Bachelorarbeit

A Heuristic Algorithm for Graph-based Surface Segmentation in Volumetric Images

Verfasserin

Rosa Zimmermann

angestrebter akademischer Grad

Bachelor of Science (BSc)

Wien, 2018

Studienkennzahl lt. Studienblatt:	A 033 521
Fachrichtung:	Informatik - Scientific Computing
Betreuerin / Betreuer:	Kathrin Hanauer, M.SC. B.Sc Dipl.-Math. Dipl.-Inform. Dr. Christian Schulz

Abstract

This thesis presents a heuristic algorithm for the problem of surface segmentation that utilizes a graph theoretic approach and the experimental evaluation of the algorithm. The empirical test was conducted on ophthalmological three-dimensional, single surfaced data.

The presented algorithm is up to seven times faster than the exact approach and yields approximately 2.9% worse solutions in the single surface case, but its straightforward extension to the multisurface requires some modifications to become competitive. The algorithm's performance varied widely for different noise levels but was not in correlation with noise. Multiple versions were tested in an algorithm engineering manner, however no difference in solution quality was measured for any of the tested version, and only one version was considerably slower than the others.

Contents

1 Introduction

1.1 Motivation

Identifying the boundaries between different objects or layers in medical images has many applications. For example atherosclerosis can be detected by determining the wall and plaque borders in intravascular ultrasound images [16]. In MRI analysis layer segmentation is used as part of the vital process called *skull stripping* [9]. The segmentation of intraretinal layers can help with the diagnosis of ocular diseases such as *age-related macular degeneration, diabetic retinopathy, and glaucoma* [1].

The Christian Doppler Laboratory for Ophthalmic Image Analysis (OPTIMA) at the Medical University of Vienna applies automated surface segmentation to images obtained using optical coherence tomography [2]. These images are three-dimensional and show multiple layers across the two-dimensional slices. After the images are preprocessed they consist of a normalized greyscale value for each voxel (a pixel in any of the pictures which form the three-dimensional object). While the presence of layers is observable to the layman, the identification of the exact boundaries takes an expert and at least 3 minutes for each slice, [7] which becomes unfeasible when hundreds of slices are available, which is the case for the data OPTIMA uses.

Thus, since technologies enabling the acquisition of such large data sets have become available, the medical community has been developing strategies for automated boundary-identification [7]. The program currently used by the University of Vienna, developed by the OPTIMA Lab [2] is graph-based and exact, but not fast enough in practice, according to out personal communication with Hrvoje Bogunovic, especially since the solution has to be recomputed multiple times after some input by an expert.

These re-computations become necessary because during data acquisition artifacts and minor inaccuracies arise that lead to solutions which an expert can identify as unrealistic. In this case the ophthalmologist has to change a few points in the data and rerun the program.

This not only gives importance to the speeding up of the program, but it is also the reason we thought a heuristic approach is of interest. When the optimal solution for the given data is not the correct real-world solution, a slightly suboptimal solution might be just as good for all practical approaches.

Another potential for improvement we considered was the fact that the previously used approach applies a general purpose algorithm to the graph constructed for segmentation purposes only and has a very specific structure. The approach presented in this thesis exploits this structure and thus does not need to use general purpose graph algorithms as subroutines.

1.2 Scope

This thesis is comprised of the introduction, implementation, theoretical analysis and experimental testing of a heuristic algorithm for segmenting surfaces, specifically intraretinal layers. The introduced algorithm is based on a graph constructed so that the problem is transformed into a minimum closed set problem, and will be compared to the (optimal) solutions and execution times of an implementation of the Minimum Cut approach alongside which the mentioned graph was presented. Furthermore, a very simple heuristic is used to validate the usefulness of our approach.

1.3 Structure

Before entering into the main part of the thesis, the concepts, terms and definitions used by related works and/or in this thesis will be briefly introduced. In Section 3 related work and their important contributions to previously developed methods are briefly introduced. Section 4 provides an explanation of the approach currently used by the OPTIMA Lab and Section 5 provides a theoretical discussion of the two algorithms presented in this thesis. Section 6 presents and discusses the results of our empirical experiments. The concluding section, 7, provides an overview of all findings and describes possible further steps.

2 Preliminaries

The previous work on this topic has introduced and used certain terminology and graph theoretical concepts. In order to be able to use them without further explanation in the remainder of this thesis, this section provides the necessary information about these concepts and notations. For the most part the same notation in [10] was used.

2.1 Terminology

The problems discussed in this thesis consist of Y two-dimensional ($X \times Z$) images (also called slices), each showing a cross-section of the objects whose surfaces are to be identified. These images are arranged into a three-dimensional grid, and the grid points (voxels) are addressed with three-dimensional Cartesian coordinates (x, y, z), where $0 < x \leq X, 0 < y \leq Y, 0 < z \leq Z$. One such voxel is made up of one measured value, denoted $\mathcal{I}(x, y, z))$.

From the values obtained by the given imaging technique a cost value has to be calculated, which will be denoted $c(x, y, z)$ for the remainder of this thesis. How the cost is determined from the measured values is an question on its own, which can be solved in a multitude of ways, to different outcomes [7, 10]. However, making this decision is beyond the scope of this thesis, and the cost function c will be assumed to be given.

Definition 1 (Column)**.** *A column (of a data set) is the set of voxels that have the same x and y coordinates. These sets are regraded as a vertical column where one voxel, (x_1, y_1, z_1), is **above** another, (x_2, y_2, z_2), iff $x_1 = x_2 \wedge y_1 = y_2 \wedge z_1 > z_2$.*

Definition 2 (Neighbor)**.** *Two columns, (x_1, y_1) and (x_2, y_2), are called neighbors iff $|x_1 - x_2| + |y_1 - y_2| = 1$.*

The problem of identifying many intraretinal layers can be formulated as finding the optimal, feasible surface set.

Definition 3 (Optimal surface)**.** *A surface $\mathcal{N}$ is a function that maps every column present in a given data set to a z value. Each surface is associated* *with a cost, which is given by* $c(\mathcal{N}) = \sum_{x=1}^{X} \sum_{y=1}^{Y} c(x, y, \mathcal{N}(x, y))$*. An optimal surface is a surface with the minimum cost among all possible surfaces.*

Intraretinal surfaces have to conform to certain constraints to be feasible.

Definition 4 (Smoothness constraints)**.** *The parameters* Δ_x *and* Δ_y*, specify the allowed change in z-value within a surface between neighboring columns. More specifically* $|\mathcal{N}(x, y) - \mathcal{N}(x \pm 1, y)| \leq \Delta_x \wedge \mathcal{N}(x, y) - \mathcal{N}(x, y \pm 1)| \leq \Delta_y$

When k surfaces $\mathcal{N}^{(1)} \ldots \mathcal{N}^{(k)}$, where $\mathcal{N}^{(1)}$ is the topmost and $\mathcal{N}^{(k)}$ the bottommost surface, are to be found in the same data set, the cost of the solution is given by the sum over the individual surfaces. Moreover, in this case each surface has its own smoothness constraints $\Delta_x^{(i)}$ and $\Delta_y^{(i)}$, and additionally the surfaces have cannot deviate from the so-called interrelation constraints.

Definition 5 (Interrelation constraints)**.** *The parameters* $\delta_l^{i,j}$ *and* $\delta_u^{i,j}$ *denote the minimum and maximum distance between surface* i *and surface* j*. Thus,* $\forall i, j, x, y : \delta_l^{i,j} \leq \mathcal{N}^{(i)}(x, y) - \mathcal{N}^{(j)}(x, y) \leq \delta_u^{i,j}$ *has to hold.*

2.2 Graph Theory

Since both the approach presented in Section 3, and our heuristic approximation rely on the problem's formulation as a graph theoretical one, this section presents the graph theoretical basics to understand the following sections.

We will use a directed, edge-weighted graph $G = (V, E, w)$ and a directed node weighted graph $G = (V, E, c)$, where V is the vertex or node set, E is the arc set, that contains pairs (u, v), where $u, v \in V$, and, finally, w is the weight function. The weight function has two different usages. In case of a node weighted graph $c(u), u \in V$ denotes the node's weights, while in case of an arc weighted graph $w(u, v), (u, v) \in E$ denotes the arc's weight.

A path from $u \in V$ to $v \in V$, denoted $\langle u, v \rangle$, is a sequence of nodes $v_1, v_2, ..., v_k$, where $v_1 = u$, $v_k = v$ and $\forall 1 \leq i < k : (v_i, v_{i+1}) \in E$. Similarly v is **reachable** from u, if there is a path from u to v [3].

We the set $N^u = \{v | (v, u) \in E\}$, the neighborhood of u.

Definition 6 (Closed node set [12])**.** *A closed node set is a subset* S *of* V *for which* $(u \in S \wedge (u, v) \in E) \Rightarrow v \in S$ *holds.*

Definition 7 (Strongly connected component)**.** *A strongly connected component [3]* $S \subseteq V$ *is a maximal set of nodes were from every node in* S *every other node in* S *is reachable.*

Definition 8 (Flow graph [3])**.** *A flow graph* $G_{st} = (V_{st}, E_{st})$ *is a directed, connected and arc-weighted graph with non-negative arc weights in which there are two nodes* $s, t \in V_{st}$ *called source and sink respectively. In this thesis we assume that the source has no incoming arcs and the sink has no outgoing ones. Additionally if* $(u, v) \in E_{st}$*, then* $(v, u) \notin E_{st}$*.*

Definition 9 (Flow [3])**.** *A flow in the flow graph* G_{st} *is a function* $f : (u, v) \to \mathbb{R}$*, where* $0 \leq f(u, v) \leq w(u, v)$ *(capacity constraint) and* $\sum_{v \in V_{st}} f(u, v) = \sum_{v \in V_{st}} f(v, u)$ *for* $u \notin \{s, t\}$ *(flow conservation).*

A maximum flow in graph G_{st} *is a flow with the maximum value. The value of a flow is given by* $\sum_{i \in V_{st}} f(s, u)$*, which is equal to* $\sum_{i \in V_{st}} f(t, u)$ *by the flow* *conservation.*

Definition 10 ((S, T)-Cut [3])**.** *A cut in a flow graph bisects the vertex set V_{st} into S and T so that $V_{st} \setminus S = T$, $s \in S$ and $t \in T$.*

A minimum cut on graph G_{st} is a cut with the minimum cost. The cost, $c(S, T)$ of a cut is given by $\sum_{u \in S, v \in T} w(u, v)$.

An important concept for the computation of both a Maximum Flow and a Minimum Cut, which are, as we will later see, dual problems, as well as for the understanding of the Maximum Flow approach, is the residual graph. Two convex optimization problems are called dual problems if one is a minimization and the other a maximization problem, and both problems have the same solution [14].

Definition 11 (Residual Graph [3])**.** *Given a flow graph G_{st} and a flow f, the residual graph $G_r = (V_r, E_r)$ is a graph that shows the capacities remaining alongside the flow f. This means $V_r = V_{st}$, and for every $(u, v) \in E_{st}$, there is an arc $(u, v) \in E_r$, with capacity $w(u, v) - f(u, v)$, and a arc $(v, u) \in E_r$ with capacity $f(u, v)$. This latter group of arcs means that we can 'cancel' the previous flow by sending flow in the opposite direction.*

Definition 12 (Augmenting Path [3])**.** *We call a path from s to t in the residual graph E_r an augmenting path.*

The residual graph is important because when there is no path from s to t in the residual graph we know that the flow is a maximum flow [3]. This also is the connection to the minimum cut problem. More formally this is stated in the Max-Flow Min-Cut Theorem presented and proven by Ford and Fulkerson [5], us given in [3] as

Theorem 1 (Max-Flow Min-Cut Theorem)**.** *Given a flow f in flow graph G_{st}, the three statements below are equivalent.*

1. *f is a maximum flow in G_{st}.*
2. *No augmenting paths can be found in the residual graph G_r.*
3. *$|f| = c(S, T)$ for some cut (S, T) of G_{st}.*

When performing an exhaustive search, in the residual graph starting from s, we obtain a vertex set that does not contain t, since there is no path from s to t. The set found by exhaustive search is the S set of minimum cut problem, and the T set can then be easily computed by $V \setminus S$. Of course, under this premise, the value of the maximum flow is equal to that of the minimum cut.

2.3 Data Structures

What remains to be explained is the term priority queue and a few possible algorithms to implement it.

'*A priority queue is a data structure for maintaining a set S of elements, each with an associated value called a key.*' [3, p. 162].

The defintions of the following paragraphs are also taken from [3].

This data structure supports the `insert`, and depending on whether one is interested in the maximum or the minimum of the keys the `maximum/minimum`, `deleteMax/deleteMin` and `increaseKey/decreaseKey`, operations. For the Single-Swipe algorithm only the `insert` and `deleteMax` operations are required, so only these will be discussed here.

As the name suggests `insert` operation adds one key-value pair to the priority queue, while `deleteMax` returns a value with the maximum key and deletes that entry from the queue.

For the scope of this thesis the most important data structure that fulfills this definition is the bucket queue, which can only be used because the data is integer and bounded. It consists of an array of length C, which ideally is the span (maximum possible key k_{max} - minimum possible key k_{min}) of the elements to be inserted. When C is smaller than the span, the operation becomes somewhat more complicated. However, since for the the topic of this thesis, using C equal to the span is feasible, we will omit how other choices of C are dealt with.

Each of the C array entries contains the first element of a (doubly) linked list. When inserting a key/value pair (k, v), v is inserted at the end of the doubly linked list starting at array entry $k - k_{min}$. Thus, each list contains the values with one key and it is easy to choose between these values. We will see later that this is an important feature.

3 Related Work

The idea to use graph-based approaches for segmentation problems appears as early as 1971 [19]. Before 1992 [18] these approaches used minimum spanning tree and shortest path algorithms instead of minimum cut algorithms for image segmentation [10]. Due to the large amount of successful and generally accepted research around the optimal segmentation of two-dimensional images, three-dimensional images were segmented slice by slice when they arose [10]. However, in this case the information encoded in the neighbor relationship of slices is ignored. According to Li et al. [10] the research trying to remedy this problem was not quite successful and they were the first to provide an approach that is both globally optimal as well as computationally feasible.

This Maximum Flow/Minimum Cut approach was successfully used for segmenting intraretinal layers multiple times. For example, see [2] and [7] where one addition was made – they allowed for Δ_x/Δ_y to be a function of the position within the image. The results achieved using automated segmentation were comparable to the work done by two expert ophthalmologists.

A more recent, machine learning based approach - which has been discussed less so far - was presented in [17]. Here each pixel is classified to belong to one of six intraretinal layers based on a few specifically designed features. In contrast to the previous approaches, this one can be used only for intraretinal layers and not for other image segmentation problems.

4 The Maximum Flow Approach

The idea of this approach is to transform the optimal surface problem into a minimal closed set problem on graph G, which can then be solved (after further transformation) by Maximum Flow/Minimum Cut algorithms on graph G_{st}. The construction of G will be discussed for the one-surface case first, and the generalization for k surfaces will follow.

The first steps of the graph theoretic approach are the choices of V, E and w for G. V will simply consist of one node for each voxel $\mathcal{I}(x, y, z)$, which can be addressed using the same coordinates. The other choices are more complex. Choosing the node-weight function will transform the optimal surface problem into a minimal closed set problem, the arc set will encode the constraints.

Rather than searching directly for the cheapest surface $\mathcal{N}$, we will search for the set that contains $\mathcal{N}$ and all nodes below it. In order to find the set corresponding to $\mathcal{N}$, w has be an adequate transformation of c. This transformation is

$$w(x, y, z) = \begin{cases} c(x, y, z) & \text{if } z = 0 \\ c(x, y, z) - c(x, y, z - 1) & \text{otherwise} \end{cases}, \tag{1}$$

as given by [10, p. 121, (1)].

At the end of this section, after one last change is made to the weights we explain why this transforms the problem into a closed set problem.

So-called **intracolumn arcs** are added to ensure that a set is closed iff all nodes below a surface are inside it. One arc is added from every node to the node exactly below it (i.e. in the same column), except of course if $z = 1$, since there is no lower node in that case. Now, if a node is not in the set, but the one above it is, the arc to the lower node violates the definition of a closed set.

Intercolumn arcs are added so that the surface (made up of the topmost node of each column) does not violate the smoothness constraints. These are defined by

$$\begin{aligned} u = (x, y, z) \wedge v \in \{&(x \pm 1, y, \max(z - \Delta_x, 0)), \\ &(x, y \pm 1, \max(z - \Delta_y, 0))\} \Rightarrow (u, v) \in E. \end{aligned} \tag{2}$$

For an example of constraint enforcing, see Figure 1, and consider the nodes $u = (1, 1, 5)$ and $v = (2, 1, 2)$ and let $\Delta_y = 2$. Finally, let C be a closed set that contains $(1, 1, 4)$ and where v is the highest node of its column to be in C. In this case, the smoothness constraints would be violated if we were to add u to C, and C would no longer be a closed set. By (2) there is an intercolumn arc $(u, a = (2, 1, 3))$, where a is not in C. u cannot be added without breaking the closed set property. If however a was added first the smoothness constraints would be satisfied for the two considered columns, and adding u while maintaining the closed set property (for these two columns only) would be possible.

At this point all closed sets, with one exception, have a feasible surface at their top. This exception is the empty set, since it does not violate the definition of a closed set, but cannot contain any surface. To avoid the empty set being found as a solution at a later step, some of the weights are altered. The set $\{(x, y, 1)\}$ is called the **base set** and because of the structure of G it is contained in any nonempty closed set.

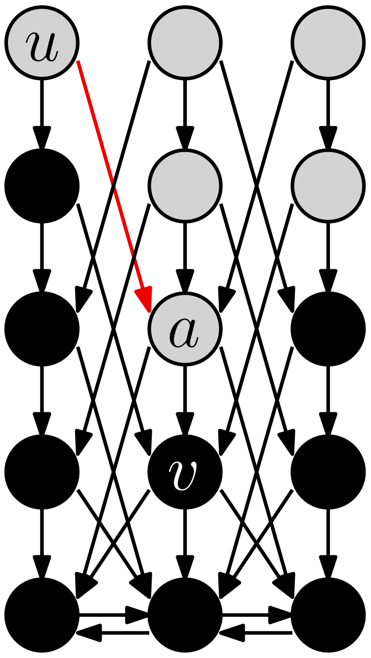


Figure 1: Graph which demonstrates the example of constraint enforcing, u, v and a mentioned previously are indicated. The nodes in C are distinguished by their black coloring. $X = 3, Y = 1, Z = 5, k = 1, \Delta_x = 2$.

Thus, to ensure that there is one feasible set with a negative cost (hereby becoming better than the empty set), an arbitrary negative weight is assigned to all nodes in the base set. Since the base set is strongly connected, because all nodes at a height $\leq min(\Delta_x, \Delta_y)$ (including height 1) are connected to height zero, there is no smaller closed set.

To get an intuition (1) transforms the problem into a minimum closed set problem, see Figure 2 and consider adding one node after the other (from bottom to top of course). Once a new node is added the cost of the previous node is subtracted from the set's cost and the cost of the new node is added. Thus, the sets cost depends solely on the topmost nodes of each column, which is exactly the surface cost.

The remaining task is to find the minimal closed set using any Maximum Flow/Minimum Cut algorithm [8].

For k surfaces, $G = (V, E)$ consists of $\bigcup_{i=1}^{k} G_i$, where each $G_i = (V_i, E_i)$ is constructed as described for the single surface case. The nodes of G_i will be denoted $(x, y, z)_i$, and in each G_i one $\mathcal{N}_i$ is given by the subset of the solution that is in G_i.

The interrelation constraints have to be modeled by adding **intersurface arcs** to E. For each pair of G_i, G_j, such that $1 \leq i < j \leq k$ we add (u, v) for $u = (x, y, z)_i \wedge v = (x, y, z - \delta_l^{i,j})_j$, iff $z - \delta_l^{i,j} > 0$ and for $u = (x, y, z)_j \wedge v = (x, y, z + \delta_l^{i,j})_u$, iff $z + \delta_u^{i,j} \leq Z$.

When $k > 1$, it can be observed that (sometimes) there are nodes that cannot be part of any feasible solution because of the minimum distance constraint. For

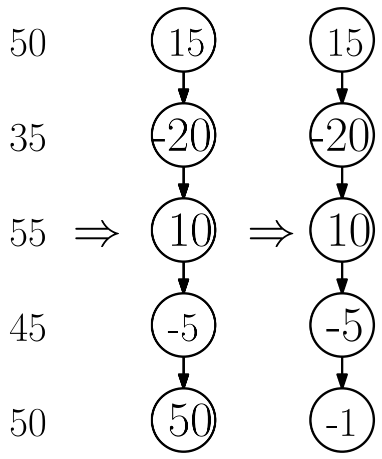


Figure 2: All cost transformation steps of an example column. The leftmost column shows the untransformed costs, the middle column shows the graphs column after applying (1), and the rightmost column shows the graph with weights after adjusting the base set's weight.

example, in the two-surface case the presence of a node from $\{(x, y, z)_2 | z < \delta_l^{1,2}\}$, which is a node in G_2, that is lower than the minimum distance between surface 1 and 2, in the solution would violate interrelation constraints. Such nodes are called **deficient nodes**, and can be eliminated from G. The base set is now the set just above the deficient nodes in V_1.

For solving this problem with a flow algorithm a flow graph $G_{st} = (V_{st}, E_{st})$ has to be constructed from G. To do so, the source and sink nodes s, t have to be added, and the graph has to become arc weighted, so that $V_{st} = V \cap \{s, t\}$ and for $(u, v) \in E$, $w(u, v) = \infty$. Finally arcs have to be added which ensure that the solution of a Maximum Flow algorithm is equivalent to a minimal closed set. For each $u \in V$, where $c(u) < 0$ an arc (s, u), with $w(s, u) = -c(.)$ is added, and the same is done for each u where $c(u) > 0$ an arc (u, t) with $w(u, t) = c(u)$. This final transformation is depicted Figure 3.

To understand why creating the graph in this way enables a maximum flow algorithm to find a closed node set consider that for a maximum flow there is no path s to t in the residual graph. This means that the flow on arcs going from the S set to the T set is equal to the capacity of these arcs. From this we can see that none of the arcs present in the node weighted graph can go from S to T, since their weight is set to infinity and they cannot have a flow equal to their capacity (at least in the presence of arcs with a weight less than infinity). Thus, S gives a closed node set in terms of the node weighted graph.

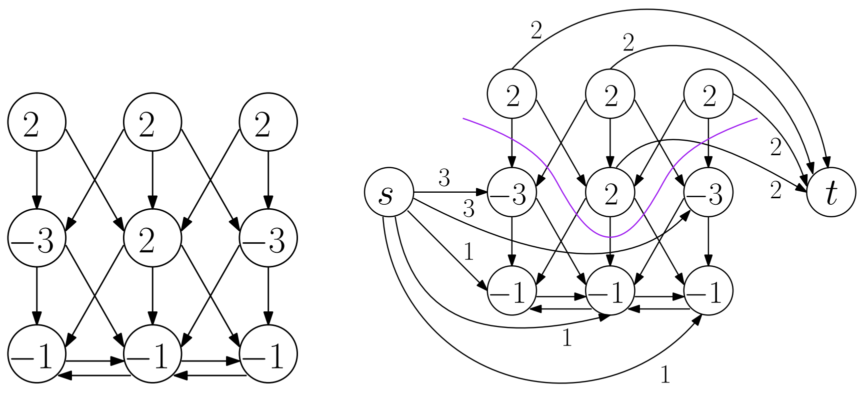


Figure 3: (a) shows a simple node-weighted example graph (with $X = 3, Y = 1, Z = 3$), while (b) shows its transformation into an edge-weighted graph. The purple line indicates which (S, T)-cut, and optimal closed set, is found by an exhaustive starting from s. In this case the graph itself is the residual graph, since there is no path from s to t in this example.

We now consider why the closed node set given by the S set is the minimal one, for an example see Figure 3. We add arcs from s to all nodes with negative weight, whose presence we welcome in the minimal closed node set, while we add arcs to t for those nodes we do not want to see in a minimal closed node set. Considering this from a minimum cut perspective we want to cut through as few of these arcs as possible, since they all have positive weight. This is exactly what we want, since cutting through arcs from s to another node means that node is not in S, even though we would want it to (given the way this graph is constructed), and cutting through an arc from a node to t means that this node is in S, even though we do not want it to (again this is because of the way the arcs are added).

The S set resulting from the execution of a Maximum Flow algorithm on this graph, and the subsequent search on the residual graph, is the minimal closed set we are looking for. Thus,

$$\mathcal{N}_i(x, y) = \max_z \{z | (x, y, z)_i \in S\}. \tag{3}$$

Since the most time is spent solving the Maximum Flow problem, it is critical which algorithm is chosen. The Ford-Fulkerson algorithm, which is based on finding augmenting paths has a time complexity of $\mathcal{O}(|E_{st}| \cdot |f*|)$, where $f*$ is the maximum flow. A running time depending on the capacity of the maximum flow is problematic as we do not necessarily know how large the bound on the node weights is. The Edmond-Karp algorithm improves the Ford-Fulkerson algorithm so that the time complexity becomes $\mathcal{O}(|V_{st}| \cdot |E_{st}|^2)$.

An important example is the Push-Relabel algorithm which is asymptotically faster than the Edmond-Karp algorithm and runs, depending on the implementation, between $\mathcal{O}(|V_{st}|^2|E_{st}|)$ to $\mathcal{O}(|V_{st}|^3)$. Since the 'slower' versions of the Push-Relabel algorithm have simple implementations and are still faster than many other algorithms they are a good choice in practice.

All information regarding Maximum Flow algorithms given in the above was taken from [3].

5 Heuristic Approaches

Two heuristic algorithms for solving the problem of surface segmentation were considered. The main algorithm, presented in this thesis, which we call Single-Swipe, was compared the most straightforward heuristic we could fathom - the cheapest level plane - in order to see whether the Single-Swipe's results are at all useful. In this section the Single-Swipe algorithm is introduced, and its workings are theoretically discussed, and the Cheapest Plane algorithm is discussed afterwards.

5.1 The Single-Swipe Approach

The idea behind the Single Swipe is to add each node to a set of previously processed nodes once, at a point in time when its addition does not violate any constraints, in a greedy manner. Starting from the base set the surface(s) move(s) 'upward' slowly during execution - all the while the storing both the current set and the cheapest set found so far.

The schematic working of the Single-Swipe is given by Algorithm 1. It uses a priority queue (PQ) that provides a fast way, `deleteMin`(), to obtain and remove the minimum stored value.

Algorithm 1: Single-Swipe

```
current ← lowest non-deficient node of each column
best ← current
insert the set directly above base set into PQ
while PQ not empty do
    next ← deleteMin(PQ)
    current ← current ∪ next
    if current cheaper than best then
        best ← current
    end
    insert all newly eligible nodes into PQ
end
return best
```

Figure 4 shows step by step how the Single-Swipe finds the same solution as the Maximum Flow approach on the graph shown in Figure 3a.

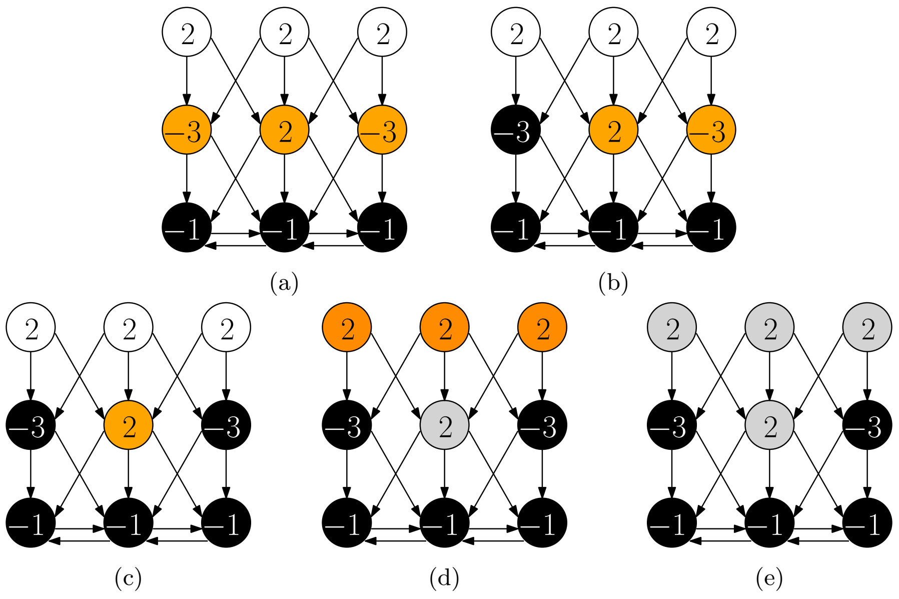


Figure 4: Internal state of the Single-Swipe. White nodes are those the Single-Swipe has not yet encountered; black nodes are part of the set the Single-Swipe considers to be optimal; orange nodes are those eligible for addition into the `current` set in the next iteration (and thus are in to priority queue at that point in time); grey nodes are those in the `current`, but not in the `best` set. (a) shows the starting state, after line 4 of Algorithm 1. (b)-(d) show the state after each of the three subsequent iterations. Finally, (e) shows the state after no more nodes are in the priority queue and the algorithm has terminated.

An example where the Single-Swipe finds a suboptimal solution is depicted in Figure 5.

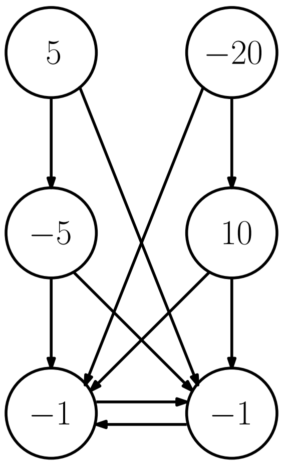


Figure 5: Graph on which the Single-Swipe yields a suboptimal solution. Inside each node its weight is indicated. $X = 2, Y = 1, Z = 3, k = 1, \Delta_x = 2$.

The two bottom nodes with weight -1 are contained in both sets at the start of the algorithm, and -5 and 10 are eligible and inserted into the priority queue. Of course the node with weight -5 is deleted, and the one with 5 is inserted. At this point 5 and 10 are in the queue. While the cost of the current set is increased by 5, when the corresponding node is deleted, the best set only contains $\{(1,1,1),(2,1,1),(1,1,3)\}$. The best set remains equally unchanged when the only eligible node, which has weight 10, is deleted. Finally, the node with weight -20 becomes eligible. At this point the current set has cost 8, while the best set so far has cost -7. Once we delete the remaining node, both sets are updated to contain all nodes and have cost -12. It can easily be seen that the optimal set is the one that does not contain node $(1,1,4)$ and has cost -17.

For an analysis we will set $N = X \cdot Y \cdot Z$ for shorter expressions.

Each node can become eligible only once, namely when the last of the nodes it has an arc to is added into the current set. Since the graph constructed above is connected, no node remains ineligible at the end of the algorithm. Thus, the bucket queue is empty after exactly kN iterations.

In each such iteration all nodes that became eligible by the previous addition, called u, have to be identified and inserted. Of all nodes only those in the neighborhood of u have to be considered. For the nodes not in the base set of any G_i, $|N^u| \leq 1 + 4 + 2(k-1) = \mathcal{O}(k)$, since, in in addition to node above u, from each neighboring column, one node might have an arc to u, and from each surface u is not in, two arcs might have u at their head. For the nodes that are in the base set of any G_i we get $|e_u| \leq 2\Delta_x + 2\Delta_y + 2(k-1)$, because of the way the smoothness constraints are added. However, since the addition of nodes that are less than Δ_x/Δ_y away from the base set cannot violate smoothness constraints, the number of nodes to be checked is $\mathcal{O}(k)$ for all nodes. Since no more nodes are checked at most $\mathcal{O}(k)$ nodes are inserted into the priority queue.

The time complexity of the priority queue operations depends on the data structure it was realized with. A common example would be a max-heap which supports both operations in $\mathcal{O}(\log n)$ [3], where n is the number of elements stored in the heap. This n has an upper bound of $X \cdot Y$, since the intracolumn arcs ensure that no more than one node in each column can be eligible.

Thus, when a max-heap is used as the priority queue, each of the kN iterations of Single-Swipe algorithm takes $\mathcal{O}(k \log n)$ and so it has a total time complexity of $\mathcal{O}(k^2 N \log n)$, or equivalently $\mathcal{O}(k^2 XYZ \log(XY))$. The bucket queue, which is used in our implementation and is also heuristic, can insert in constant time [4], and deleting the maximum takes $\mathcal{O}(C)$, where C is the span of the data. In this case the time complexity becomes $\mathcal{O}(kN(k+C))$.

5.2 The Cheapest Plane Heuristic

The provided data is flattened in a way that makes the surface resemble a level plane [2]. For this reason we chose to compare the Single-Swipe to the cheapest level plane.

This simple heuristic works directly with the costs instead of the the transformed weights. In the single surface case the Cheapest Plane algorithm sums over all nodes with a given height, starting with 0, and returns the cheapest cost found while testing all heights. The multisurface case is more complex and our version of the Cheapest Plane algorithm does not always find a solution.

The globally cheapest level plane might be ineligible because of interrelation constraints.

For example, consider a problem where Z, the maximum height, is equal 5, and we want to find two surfaces, with a minimum distance, $\delta_l^{1,2} = 3$. Now, the level plane at height 2 cannot be part of any solution. Yet, our algorithm starts by finding the cheapest level plane among all possible heights, and if this were the one at height 3, no feasible solution could be found.

Since the Cheapest Level Plane algorithm does not work for multiple surfaces, an alternative would be needed.

6 Experimental Evaluation

The experiments in this section were conducted using an algorithm engineering approach. Multiple variants of the Single-Swipe are compared both to each other and to the other two algorithms.

Thus, we start this section by explaining which parts of the algorithm could be done in multiple alternative ways. In the subsequent section the experiments' setup, their results and a discussion is presented - first for the single surface and then for the multiple surface case.

All three algorithms were implemented in C++ (11) only, and where possible the data structures found in the KaHIP library were used. These data structure's implementation details are documented up in the manual [15].

6.1 Hardware

All data appearing in this section was gathered on a 4-core (only one of which was utilized at any given time) university/provided machine. The cores are each an Intel Xeon E5-2650, which has 2.20 GHz clock speed.

The L1, L2 and L3 caches are of size 32KB, 256KB and 30720KB respectively.

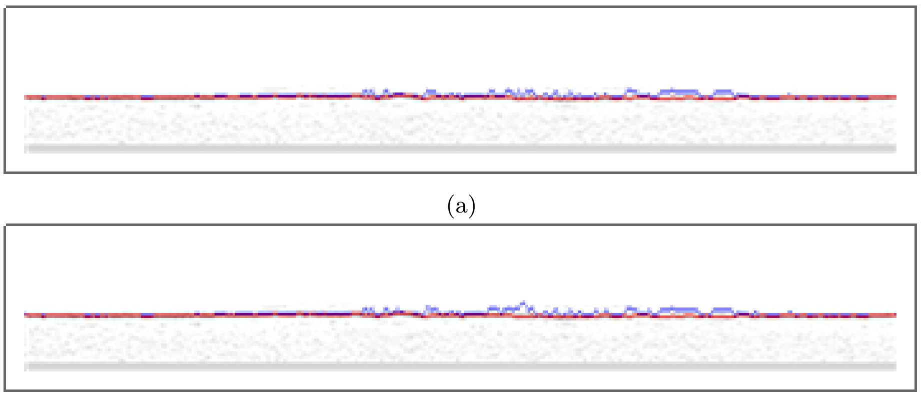
There are 15.7 GB memory available, which were not exceeded during any computation conducted for this thesis, thus disk information will be omitted.

6.2 Variants of the Single-Swipe Algorithm

Three components of the Single-Swipe algorithm were identified to be interchangeable.

The first is the data structure to serve as a priority queue. While we started out with a max-heap, we soon realized, that because the weights are integer a (in our case minimum) bucket queue can be used. It is not only fast, but allows for a lot of flexibility when choosing between multiple values with the same key. For this reason, among others, we chose to stick with the bucket queue, even though there are many more options to implement a priority queue.

This brings us to the second interchangeable part - the scheme by which one of multiple minimum keys is chosen. Here three different options were implemented and evaluated. By starting with a Last-In-First-Out (LIFO) scheme we realized that this caused the Single-Swipe to move up as far as possible within in a small neighborhood (since it is always nodes in the same or neighboring columns of the one currently processed that are inserted into the queue). For the flattened surfaces this algorithm was engineered toward this is a bad fit. Thus, we decided to try the opposite - a First-In-First-Out (FIFO) scheme and a scheme where among the node-indices to be returned one of those with minimum height is chosen. This final scheme will be referred to as LOWER in the tables given in the following sections. Figure 6, as well as the numbers discussed in the following section, show that the differences observed between FIFO and LIFO queues do no have any major influence on solution quality.



(a)

(b)

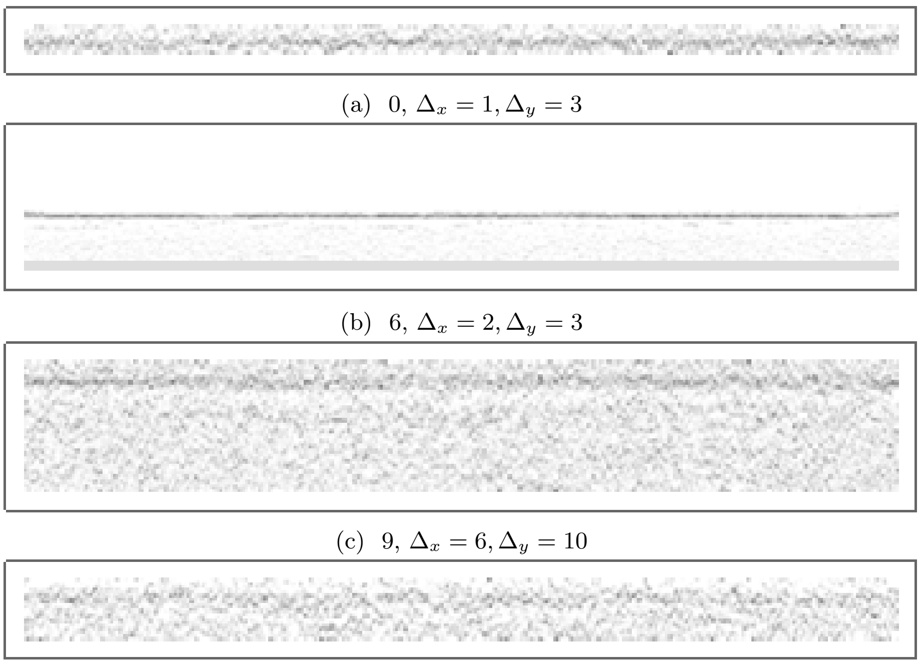
Figure 6: Comparison of Single-Swipe (blue) results with (a) FIFO and (b) LIFO versions to optimal result (red) on problem number 6783406, noise level 6 and slice 64.

In the very first implementation we encountered a phenomenon different from, yet still connected to, the one described above. The currently processed set moved up the right hand side of the image, while the left hand side was only dragged upward when absolutely necessary for progressing. This behavior arose because in combination with a LIFO queue the base set was inserted into the queue from left to right. Thus, (and because of the fact that the node in the same column is always inserted before those in the neighboring columns), always the rightmost possible column was processed. This lead us to try to randomize the order in which the base set is inserted. In the following tables the left-to-right order will be denoted using SEQ, while a random insert order will be labelled RAND.

6.3 Single Surface

6.3.1 Method

All three previously described algorithms were run using the implementation described in Appendix A, on 20 data sets provided by the OPTIMA Lab. These data sets all have $X = 512, Y = 128$, while Z varies from problem to problem. These images show five different surfaces, each with four different noise levels, which were provided under the names 0,6,9 and 10. The different versions of one such surface can be found in Figure 7. Each noise level was solved with its own smoothness constraints provided by be the OPTIMA Lab, which are also indicated in Figure 7. The surfaces are flattened and remain at roughly the same height throughout all slices, as is depicted in Figure 8. Each provided data set contains one file in CSV format with the costs normalized into the $[0, 255]$ range and another one with the node-weights (in the range $[-255, 255]$). A visualization of one such data set can be seen in Figure 9.



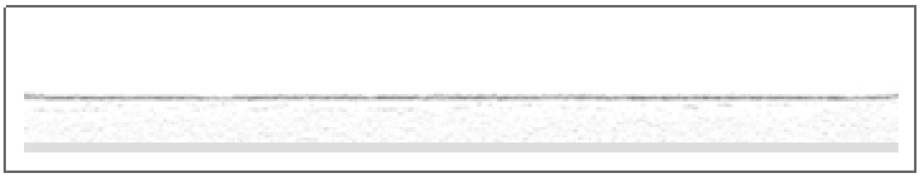
(a) 0, $\Delta_x = 1, \Delta_y = 3$

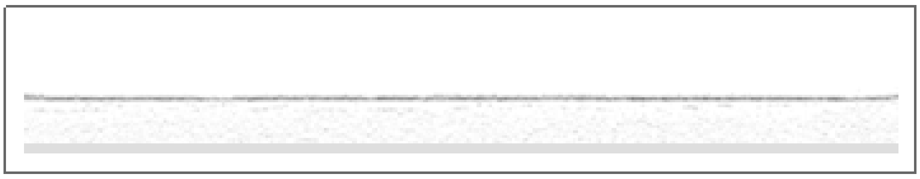
(b) 6, $\Delta_x = 2, \Delta_y = 3$

(c) 9, $\Delta_x = 6, \Delta_y = 10$

(d) 10, $\Delta_x = 6, \Delta_y = 10$

Figure 7: Slice 64 of all noise levels of problem number 6783406.


(a) Slice 0


(b) Slice 64

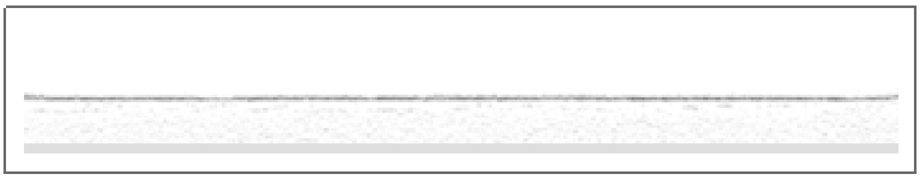
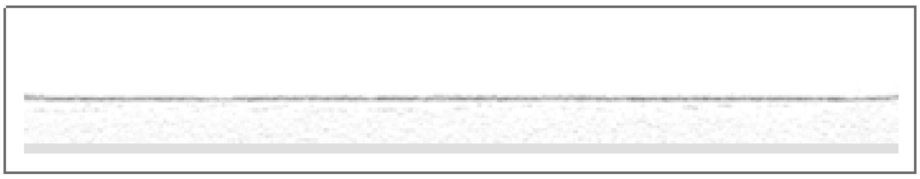

(c) Slice 128

Figure 8: Three slices of problem number 6783406 and noise level 6


(a)

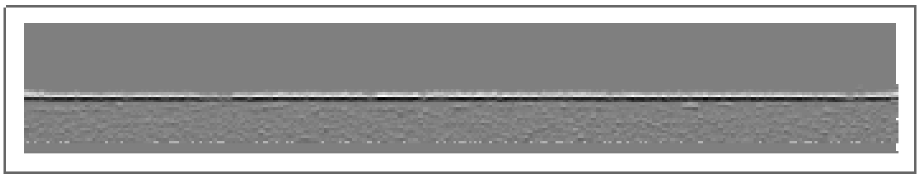

(b)

Figure 9: (a) Costs and (b) transformed node weight problem number 6783406, noise level 6 and slice 64.

In order to see effects, including interaction effects, of the inserting and queue alternatives, every problem was solved using every possible combination of alternatives. To avoid conclusions based only on outliers each run was repeated ten times. The raw data was output into files and then processed as described in Appendix A.2. The run times of the Single-Swipe and Minimum Cut algorithms were obtained to five decimal points. For the Cheapest Plane algorithm running time was not measured, as it was only used to in/validate the Single-Swipe algorithm's usefulness.

Two metrics were used to gauge the quality of a given algorithm/version. These are the relative error

$$r = \frac{c_s - c_c}{c_c}, \tag{4}$$

and the speedup

$$S = \frac{t_c}{t_s}, \tag{5}$$

where t is the execution time and c the cost of the surface found by either the Single-Swipe, denoted by the subscript s or the Minimum Cut, denoted by the subscript c.

In order to make statements about groups of runs the (geometric) mean and the variance of these groups was calculated. While the mean is less robust against outliers than the median, significance testing proves to be a lot simpler when making statements about two groups (geometric) means rather than about their median. For these statements we assumed a normal distribution, even though a Shapiro-Wilks test revealed strong non-normality for the data as a whole, because of the Central Limit Theorem and the large sample sizes (the smallest sample used is of size 110; the largest is of size 1200) involved in these tests.

When testing for a difference in means for two groups we used the Welch-test, since it does not assume equal variances. For tests involving more than two groups the p-value resulting from an ANOVA was used. Both statistics were calculated as given in [11].

When it was necessary to do so, Levene's test, as described in [6], was used to test for equality of variances.

In all cases the tests were two-sided, and the significance level was chosen to be 5%

6.3.2 Results and Discussion

First, we compare the different versions of the Single-Swipe against each other. We then discuss how the Single-Swipe compares to the Cheapest Plane approximation and the optimal solution.

	FIFO	LIFO	LOWER
SEQ	0.02766	0.02893	0.02901
RAND	0.02766	0.02891	0.02901

Table 1: Relative error of the Single-Swipe algorithm for all combinations of tested alternatives. Geometric mean over all noise levels, surfaces and runs.

Table 1 shows that there is no difference in solution quality when using a random inserting order, instead of a sequential one.

Moreover, no difference was observed between groups given by combinations of insert order and queue version.

However, the situation is very different when looking at the speedup of these versions (Table 2). While switching from SEQ to RAND (or back) does - again - not make any difference, using the LOWER decision scheme for the bucket queue is slower than the others. The four groups that on average yield speedup (instead of slowdown) are not different from each other (on average).

	FIFO	LIFO	LOWER
SEQ	2.72259	2.69551	0.61694
RAND	2.78974	2.74422	0.63790

Table 2: Speedup of the Single-Swipe algorithm for all combinations of tested alternatives. Geometric mean over all noise levels, surfaces and runs.

At this point we have reason to exclude the LOWER bucket queue for its immense inefficiency, when compared to other queue versions. Using a more complex data structure as insert key, for example a pair made up of node height and node weight, could increase the efficiency, but is not necessary, as the LOWER version did not yield better results than the others.

Among the four remaining groups no basis for decision is provided by the previously discussed metrics. One might wish to consider the variance of quality (Table 3) or speedup (Table 4) for each of these groups.

	FIFO	LIFO	LOWER
SEQ	0.00019	0.00017	0.00017
RAND	0.00018	0.00016	0.00016

Table 3: Variance of relative error over all noise levels, surfaces and runs of the Single-Swipe algorithm for all combinations of tested alternatives.

	FIFO	LIFO	LOWER
SEQ	5.90995	5.86477	0.18414
RAND	5.91206	5.66601	0.19298

Table 4: Variance of speedup over all noise levels, surfaces and runs of the Single-Swipe algorithm for all combinations of tested alternatives.

The speedup's variance is fairly large, and not different (p-value=0.978) for the four groups of interest, and neither is the relative error's variance when using a FIFO rather than a LIFO queue. No help in settling for a version is provided by the variances, and since Figure 6 also provides little help, this decision cannot be made within the scope of this thesis.

To summarize, the insert order does not seem to matter on average. Always inserting the lowest possible node makes the Single-Swipe incapable to compete with the Maximum Flow algorithms speed, and it is not clear whether a FIFO or a LIFO selection scheme would perform better in practice. Thus, for the remaining discussion we will not distinguish the measurements by inserting order and will completely omit the LOWER scheme.

While the previously presented data already contained information about how the Single-Swipe algorithm compares to the Minimum Cut algorithm, we have not yet seen the results of the Cheapest Plane algorithm. Table 5 shows how the two heuristic approaches compare on each noise level.

There was a significant difference between FIFO and LIFO for noise level 6 only, but their combined average was also significantly worse than the result produced by the Cheapest Plane algorithm for this noise level. For all noise

	0	6	9	10
PLANE	0.02862	0.00749	0.03073	0.05669
FIFO	0.02260	0.01409	0.04560	0.04032
LIFO	0.02310	0.01674	0.04540	0.03982

Table 5: Relative error of the Single-Swipe algorithm (FIFO and LIFO selection scheme) and the Cheapest Plane algorithm. Geometric mean over all surfaces, runs, and insertion orders.

levels there was a significant difference between the Cheapest Plane, and the average of FIFO and LIFO versions combined. Yet, there is no immediately apparent pattern in which cases the Single-Swipe and in which cases the Cheapest Plane produced better results. Figure 7 shows that it is certainly not noise level, or smoothness constraints. Perhaps further investigation is required to find out in which cases the Single-Swipe performs favorably.

In any case, it should be mentioned that the data used intraretinal layer segmentation is flattened in way that is quite well approximated by a level plane. Since all appearing algorithms can be used as general purpose segmentation techniques, it should be considered that for less level (not flattened) surfaces it is quite clear that the Cheapest Plane approach cannot compete.

What looking at the noise levels individually tells us, is that the high variances observed in Table 4, originate from speedups differing between noise levels (Table 6), since within each noise level the variance is much lower (Table 7).

	0	6	9	10
FIFO	0.96463	2.91462	7.45944	2.75071
LIFO	0.96501	2.86403	7.36368	2.68851

Table 6: Speedup obtained by using Single-Swipe versions FIFO and LIFO for each noise level. Geometric mean over all insert orders, surfaces and runs.

	0	6	9	10
FIFO	0.00383	0.18129	0.25311	0.00955
LIFO	0.00352	0.18362	0.25786	0.00835

Table 7: Variance of the relative error of Single-Swipe versions FIFO and LIFO, over all insert orders, surfaces and runs, for each noise level.

For the sake of completeness the absolute running times of the two main algorithms are shown in Table 8.

To summarize, the Single-Swipe algorithm yields results 1.5%-4.5% worse than the optimal minimum cut algorithm and can be up to seven times as fast - all depending on the noise level. Yet nor linear correlation between noise and the Single-Swipe's performance is discernible. The insert order does not influence the performance on this data, and neither does the choice between FIFO and LIFO bucket queue tie breaking schemes.

	0	6	9	10
CUT	9.33906	35.23472	67.72735	26.01888
FIFO	9.78264	12.13984	9.10582	9.51265
LIFO	9.66325	12.28603	9.16812	9.63257
LOWER	15.40826	158.47027	48.06535	31.68422

Table 8: Running times of Single-Swipe versions and Minimum Cut algorithm on different noise levels. Mean over all surfaces, runs, and insertion orders.

6.4 Multiple Surfaces

6.4.1 Method

Since the Single-Swipe algorithm did not perform in a satisfactory way in the multiple surface case (in this implementation), no rigorous empirical tests were conducted. Rather we will explain in the following section the behavior that leads to these problematic results.

6.4.2 Results and Discussion

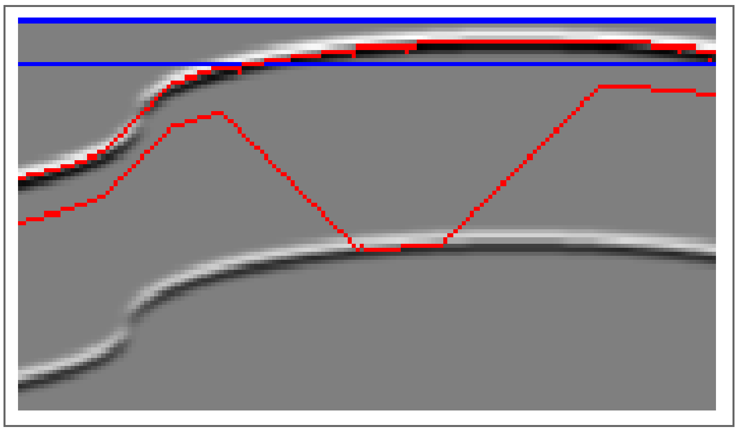


Figure 10: Final stage of the working of the Single-Swipe algorithm on a problem where it fails to find an acceptable solution. The red line is the solution surface found by the Single-Swipe. The blue line shows up to which point the nodes were inserted into the 'current' set by the Single-Swipe. Grey pixel indicate 0 weight at that point in the image, darker coloring indicates negative weight, while lighter indicates higher weights. $X = 159, Y = 0, Z = 89, k = 2, \Delta_x = 1, \delta_l^{1,2} = 10, \delta_u^{1,2} = 80$.

Figure 10 shows the Single-Swipe's state after the final iteration performed on a toy example. We will use this example to demonstrate how and why the algorithm performs poorly. It is obvious already at first glance that the lower of the two found surfaces has almost no connection to what seems the correct solution.

This phenomenon occurs because the Single-Swipe, moving from bottom to top, has to 'overcome' expensive nodes and postpones until the latest possible moment due to its greedy nature. This leads to more than one problematic behavior in the multisurface case.

For one, when the first expensive area or the most expensive surface is encountered by an upper surface the lower surface moves as close as possible (see Figure 11), since all lower nodes are (by definition) cheaper at this point. During the iterations where the upper surface has not yet 'broken through' the expensive area this continues, leaving us with a situation like the one shown in Figure 12.

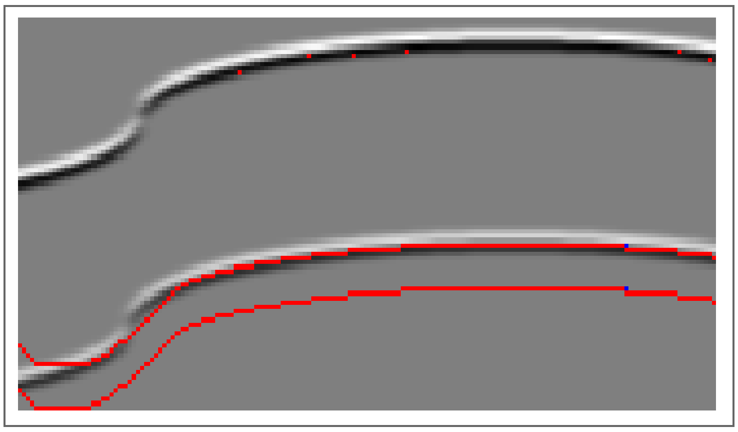


Figure 11: Stage of the Single-Swipe algorithm, where a lower surface moves up to the upper one. The red line is the solution surface found by the Single-Swipe. The blue line shows up to which point the nodes were inserted into the 'current' set by the Single-Swipe. Grey pixel indicate 0 weight at that point in the image, darker coloring indicates negative weight, while lighter indicates higher weights. $X = 159, Y = 0, Z = 89, k = 2, \Delta_x = 1, \delta_l^{1,2} = 10, \delta_u^{1,2} = 80$.

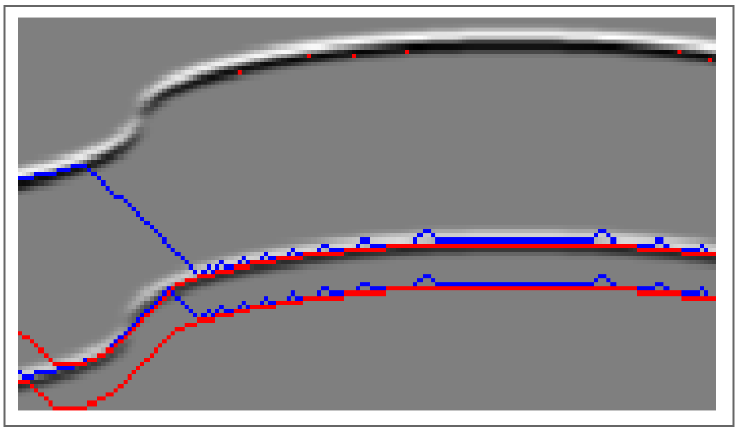


Figure 12: Stage of the Single-swipe algorithm, where a lower surface moves upward directly behind the upper one until the latter one enters a cheaper area. The red line is the solution surface found by the Single-Swipe. The blue line shows up to which point the nodes were inserted into the 'current' set by the Single-Swipe. Grey pixel indicate 0 weight at that point in the image, darker coloring indicates negative weight, while lighter indicates higher weights. $X = 159, Y = 0, Z = 89, k = 2, \Delta_x = 1, \delta_l^{1,2} = 10, \delta_u^{1,2} = 80$.

What can also be seen in Figure 12 is that the lower surface has already moved up to the expensive area the top surface broke through earlier. At this point, the concluding, and most disturbing problem becomes apparent. If it was cheaper for the top surface to break through this part than trough the other parts, it will also be cheaper for the lower surface to break through this part than for the top surface to break though another part. The exact moment this occurs is shown in Figure 13, and the result is depicted in Figure 14

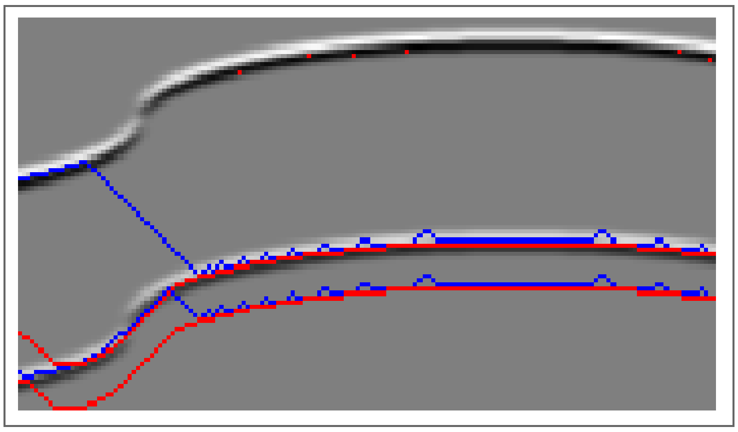


Figure 13: Stage of the Single-swipe algorithm, where a lower surface moves past the surface it should find. The red line is the solution surface found by the Single-Swipe. The blue line shows up to which point the nodes were inserted into the ‘current’ set by the Single-Swipe. Grey pixel indicate 0 weight at that point in the image, darker coloring indicates negative weight, while lighter indicates higher weights. $X = 159, Y = 0, Z = 89, k = 2, \Delta_x = 1, \delta_l^{1,2} = 10, \delta_u^{1,2} = 80$.

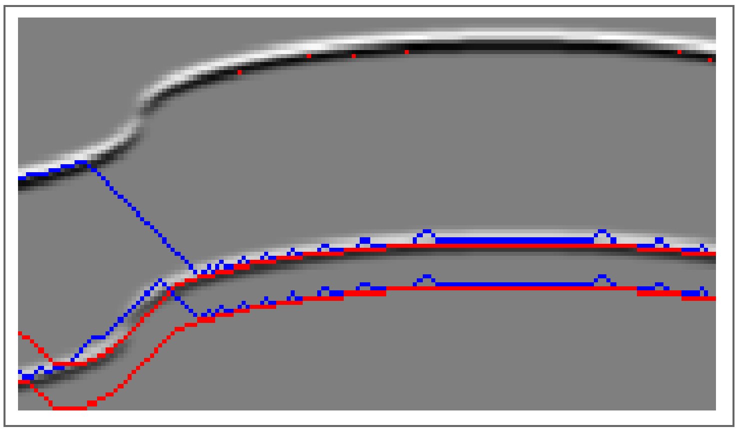


Figure 14: Stage of the Single-swipe algorithm resulting from the problem shown in Figure 13. The red line is the solution surface found by the Single-Swipe. The blue line shows up to which point the nodes were inserted into the ‘current’ set by the Single-Swipe. Grey pixel indicate 0 weight at that point in the image, darker coloring indicates negative weight, while lighter indicates higher weights. $X = 159, Y = 0, Z = 89, k = 2, \Delta_x = 1, \delta_l^{1,2} = 10, \delta_u^{1,2} = 80$.

The previously described problems have to be remedied before the Single-Swipe algorithm can be applied to multisurface problems. Finding such a remedy is most probably not a trivial matter, as no easy strategy of choosing between nodes seems to solve this problem. For instance, always moving the upper surface first means that it might move past the optimal solution before the lower surfaces find their optimal position.

7 Conclusion

Within the frame of this thesis the Minimum Cut approach presented in [10] was implemented for single- as well as multisurface problems. An algorithm that operates on the same graph, yet finds an approximate instead of an optimal solution, the Single-Swipe algorithm, was presented and also implemented.

The empirical test, conducted on ophthalmological three-dimensional, single-surfaced, data showed that the Single-Swipe's quality varies widely between different noise levels. In comparison with our implementation of the previous approach the Single-Swipe was on average over three times as fast, and the solution quality dropped by 2.9%.

A much simpler heuristic, the cheapest level plane, could compete with the Single-Swipe algorithm on this data. However, there is reason to assume that this is not generally the case.

In the multisurface case the Single-Swipe exhibits problematic behaviors that render the results useless. Further work is needed to see whether if and how these flaws could be remedied.

7.1 Further Steps

With regard to the Single-Swipe algorithm the most important task is the development and testing of ideas that solve the problem(s) described in Section 6.4. One possibility that comes to mind is the softening of the algorithm's greediness by exchanging the simple priority queue for a structure with more complex rules of choosing among the eligible nodes.

One example for such a more complex queue would be to use a look-ahead strategy when calculating a key for the nodes to be inserted.

For the problem of single surface intraretinal layer segmentation the Cheapest Plane approximation can be used to restrict the range within which the optimal surface can be found in. For problems where this range is significantly smaller (e.g. low Δ_x and Δ_y) the Maximum Flow approach can be used to refine the solution found by the Cheapest Plane. In this case a solution of very high quality could be found in reduced time. Implementation and testing of this idea were not covered in this thesis.

Appendices

A Implementation Details

This Appendix provides an Overview of how the results presented in this thesis were obtained.

A.1 Software

All relevant implementation done for this thesis can be found in the `retina_4D` class.

This class requires the number of surfaces to be found as well as all constraints to be passed to the constructor in form a file that adheres to the following structure. (All of the values given below have to be present, and no other symbols may appear in the configuration file, thus the comments after '//' should be omitted.)

k
$\Delta_x^{(1)}$ $\Delta_y^{(1)}$ $\delta_l^{1,2}$ $\delta_u^{1,2}$ $\dots \delta_l^{1,k}$ $\delta_u^{1,k}$ // topmost surface
$\vdots$
$\Delta_x^{(k)}$ $\Delta_y^{(k)}$ $\delta_l^{k,1}$ $\delta_u^{k,1}$ $\dots \delta_l^{k,k-1}$ $\delta_u^{k,k-1}$ // bottommost surface

The attentive reader has already noticed that there is redundant information in this format (the minimum distance from surface 1 to surface 2 is the same as the minimum distance from surface 2 to surface 1). Thus, care has to be taken that these values are indeed equal in the supplied configuration file, otherwise different constraints are assumed at different program points.

After the construction of the `retina_4D` object the problem's data has to be supplied in a CSV file. The Cheapest Plane algorithm requires the untransformed costs, while the others need the weights of the graph described in Section 4. The format of this data should be as follows in all these cases.

X Y Z
$w(0,0,Z), w(1,0,Z), \dots, w(X,0,Z); w(0,1,Z), \dots, w(X,1,Z); \dots; w(0,Y,Z), \dots, w(X,Y,Z);$
$\vdots$
$w(0,0,0),\ w(1,0,0),\ \dots,\ w(X,0,0);\ w(0,1,0),\ \dots,\ w(X,1,0);\ \dots;\ w(0,Y,0), \dots, w(X,Y,0);$

This file should be passed to the `read_from_csv` function. It can be called multiple times to switch out the data the object operates on.

In the multisurface case it is assumed that all G_i are built from the same data. While this is not always the case in practice, it only needs to be adjusted once the Single-Swipe algorithm has successfully been adapted for multiple surfaces.

A.1.1 Minimum Cut

For this algorithm the main work was done by the flow graph data structure and the Push-Relabel algorithm implemented in the KaHIP library.

Only the construction of the graph described in Section 4 has to be done by the `retina_4D` class. Such a `flow_graph` object of the KaHIP library is returned by the `construct_graph` function. As stated before, the three-dimensional graph given in CSV format duplicated k times and the intersurface arcs are added as specified in the configuration file.

A.1.2 Single-Swipe

The implementation of the Single-Swipe algorithm can be called with the `find_surface` function, and does not use a graph data structure. Because of the regular nature of the graph constructed for the Minimum Cut approach, the weights are stored in a one-dimensional array, which is accessed at the $y \cdot X \cdot Z + z \cdot X + x - 1$ index when the weight of the node $u = (x, y, z)$ is required.

One peculiarity is the `get_max_deficient` function, which has (more than) exponential time complexity in k. The reason for this is that in [10] it is stated that the constraints are given for all pairs of surfaces. Thus there are 2^k possibilities for the height of the deficient node set. For example, consider $k = 4$, now for the topmost surface the height of the deficient node set is

$$\max(\delta_l^{(1,4)}, \delta_l^{(1,2)} + \delta_l^{(2,4)}, \delta_l^{(1,3)} + \delta_l^{(3,4)}, \delta_l^{(1,2)} + \delta_l^{(2,3)} + \delta_l^{(3,4)}). \tag{6}$$

If considered further it could be seen that there are 2^{k-i} possibilities, where $1 \leq i \leq k$ is the index of the surface enumerated from top to bottom.

The most complex step of this algorithm is the identification of the nodes that become eligible thorough the addition of another node u. We need to consider all nodes in $\{v | (v, u) \in E_{st}\}$. However for any of these nodes there might be yet another node $w \notin$ `current`, with $(v, w) \in E_{st}$. Thus for every v the `check_bottleneck` function is called which returns $|\{w | (v, w) \in E_{st} \wedge w \notin$ `current`$\}|$. If this is not zero, the node does not yet become eligible.

As a priority queue the bucket queue provided by the KaHIP library was used, with one change. Instead of only choosing between values with equal keys using a LIFO scheme, prepossessor statements were added to switch between the FIFO, LIFO and LOWER schemes, described in section 6.3. This is done to avoid a large number of additional unnecessary checks at run time. Similarly, whether the loop index is permuted is also decided at compile time.

This bucket queue is a maximum priority queue, so in order to be able to use it without further changes, the sign of all inserted keys was reversed.

A.1.3 Cheapest Plane

For the single surface case, the algorithm is implemented exactly as described in 5.2. In the multisurface case, the algorithm runs k times and after the first iteration it starts at height $z_i + \delta_l^{i,i+1}$, where i is the number of previously found surfaces and z_i the height of the ith found surface.

A vector of the cost of each of these level planes is returned, and if the height of the problem would be exceeded the maximal integer (in place of infinity), is inserted for all remaining surfaces.

A.2 Data Acquisition and Processing

Ten iterations of one Single-Swipe version were executed in one run and the resulting cost and times were output into a file. All of the created files were read into a multi-dimensional object in the R language.

Here duplicate measurements (for example the optimal cost found by the Minimum Cut algorithm is output ten times in one such run) were dropped, in order to not artificially increase the sample size.

For the relative metrics, speedup and relative error, the geometric mean was used, in order to avoid biasing the value, while for the cost and run time values the standard mean was used.

For conducting Welch-test the `t.test` function provided by R was used, which has Welch's T-test by default [13]. For more than two groups the `aov` and for comparison of variances the `leveneTest` functions were used.

B Program Usage

Two programs were written for using the `retina_4D` class. The first to be presented here is a simple test program (`test_4D.cpp`) that outputs the solution surface found by the tested algorithm. Following this, a program for generating more advanced visualizations (`visualize_4D.cpp`) is explained.

Both these programs need the same command for compilation.

```
g++ -std=c++11 -O3 -D<queue> -D<insert> examples/test4D/[program
filename] -I lib/oneSwipe/ -I lib/ -I lib/tools/ -I lib/
data_structure/priority_queues/ -I lib/data_structure/ -I lib
/algorithms/ -I lib/tools -I lib/partition -o program lib/
algorithms/push_relabel.cpp lib/tools/random_functions.cpp
```

To avoid typing this the makefile can be used which provides the `build` and `build_visual` rules for the two programs. The makefile has variables for each of the placeholders in angled brackets. The queue placeholder (makefile variable name `queue`) can take the values FIFO, LIFO and LOWER_NODE (default FIFO) which refer to the bucket queue version described in Section 5.1. The insert placeholder can be given the RAND_INSERT value to use a randomized insert scheme, and set to anything else for sequential insertion.

B.1 Simple Test Program

This test program can run all three algorithms appearing in this thesis. For the Minimum Cut and the Single-Swipe the solution is output in the format

$(0, 0, h)_1, \ldots (0, 0, h)_k$

$\vdots$

$(X, 0, h)_1, \ldots (X, 0, h)_k$

$\vdots$

$\vdots$

$(X, Y, h)_1, \ldots (X, Y, h)_k,$

and for all algorithms the resulting cost is output on the command line.

The program should be called from the commend line as follows:

```
./program <swipe|cut|plane> <node weights CSV> <untransformed costs
CSV> <configuration file> <output file>
```

All files have to be in the format given in Appendix A.

For the two main algorithms the program can also output data to be used for visualizing the solution. In order to do so two further parameters have to be added at the end of the above line. First the zero-based number of the slice to be visualized and then the file into which the output should be written, has to be specified.

To obtain the actual picture the `visualize_retina` notebook needs to be opened and cells should be run. After this the command

```
Picture["path/to/visualization/outputFilename"]
```

should be inserted and executed. The picture can then be found in the `<outpuFilename>.png` file.

B.2 Advanced Visualization

This program can output data for depicting the problem without any solution and the problem with both algorithm's solution indicated. Furthermore, it can output data for creating an animation that shows the workings of the Single-Swipe algorithm step by step. All these data sets are transformed into actual visuals by using the Mathemtica functions provided in the `visualize_retina` notebook.

The makefile provides the option to run this program. For this there are more variables that can be set in the make command. These variables will appear in brackets where the value they should contain is described.

In order to obtain the visualization of the problem alone, the file containing the this problem (`cost_graph`), the configuration file (`config`) and finally the index of the slice to be printed (`visualize_result`).

To obtain a picture showing the solutions of both algorithms before the configuration file first the graph containing the untransformed cost (`cost_graph`) and then the file containing the node weights (`node_graph`) should be given.

For a final possibility, the creation of data for subsequent animation, two further arguments should be added to the program call used for creating a picture of the solution. First the number of the slice whose states should be animated should be given (`visualize_swipe`), and then the width up to which (from 0) the problem should be visualized (`visualize_swipe_width`).

For this program all output appears on the command line by default and should be redirected to a file (`out_file`) from outside of the program. The makefile does this automatically.

While the comparative visualization is obtained the same way as in the previous section, for the animation the second function in the Mathematica should be used. The command to be inserted after executing the present cells should look like

```
Picture["<path/to/visualization/output>", <visualization width>].
```

References

[1] Abràmoff, M. D., Garvin, M. K., and Sonka, M. Retinal imaging and image analysis. *IEEE reviews in biomedical engineering 3* (2010), 169–208.

[2] Bogunović, H., Sonka, M., Kwon, Y. H., Kemp, P., Abràmoff, M. D., and Wu, X. Multi-surface and multi-field co-segmentation of 3-d retinal optical coherence tomography. *IEEE transactions on medical imaging 33*, 12 (2014), 2242.

[3] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to algorithms.* MIT press, 2009.

[4] Edelkamp, S., and Schroedl, S. *Heuristic search: theory and applications.* Elsevier, 2011.

[5] Ford, L. R., and Fulkerson, D. R. Maximal flow through a network. *Canadian journal of Mathematics 8*, 3 (1956), 399–404.

[6] Fox, J. *Applied regression analysis and generalized linear models.* Sage Publications, 2015.

[7] Garvin, M. K., Abramoff, M. D., Wu, X., Russell, S. R., Burns, T. L., and Sonka, M. Automated 3-d intraretinal layer segmentation of macular spectral-domain optical coherence tomography images. *IEEE transactions on medical imaging 28*, 9 (2009), 1436–1447.

[8] Hochbaum, D. S. A new—old algorithm for minimum-cut and maximum-flow in closure graphs. *Networks: An International Journal 37*, 4 (2001), 171–193.

[9] Iglesias, J. E., Liu, C.-Y., Thompson, P. M., and Tu, Z. Robust brain extraction across datasets and comparison with publicly available methods. *IEEE transactions on medical imaging 30*, 9 (2011), 1617–1634.

[10] Li, K., Wu, X., Chen, D. Z., and Sonka, M. Optimal surface segmentation in volumetric images-a graph-theoretic approach. *IEEE transactions on pattern analysis and machine intelligence 28*, 1 (2006), 119–134.

[11] Maindonald, J., and Braun, J. *Data analysis and graphics using R: an example-based approach*, vol. 10. Cambridge University Press, 2006.

[12] Picard, J.-C. Maximal closure of a graph and applications to combinatorial problems. *Management science 22*, 11 (1976), 1268–1272.

[13] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2017.

[14] Rockafellar, R. T. *Conjugate duality and optimization*, vol. 16. Siam, 1974.

[15] Sanders, P., and Schulz, C. Kahip v2. 0–karlsruhe high quality partitioning–user guide. *arXiv preprint arXiv:1311.1714* (2013).

[16] Sonka, M., Zhang, X., Siebes, M., Bissing, M. S., DeJong, S. C., Collins, S. M., and McKay, C. R. Segmentation of intravascular ultrasound images: A knowledge-based approach. *IEEE Transactions on Medical Imaging 14*, 4 (1995), 719–732.

[17] Vermeer, K., Van der Schoot, J., Lemij, H., and De Boer, J. Automated segmentation by pixel classification of retinal layers in ophthalmic oct images. *Biomedical optics express 2*, 6 (2011), 1743–1756.

[18] Wu, Z., and Leahy, R. Image segmentation via edge contour finding: A graph theoretic approach. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on* (1992), IEEE, pp. 613–619.

[19] Zahn, C. T. Graph theoretical methods for detecting and describing gestalt clusters. *IEEE Trans. Comput. 20*, SLAC-PUB-0672-REV (1970), 68.