NTNU
Innovation and Creativity

Segmentation of Medical Images Using CBR

Christian Marshall Rieck

Master of Science in Computer Science

Submission date:	July 2007
Supervisor:	Richard E. Blake, IDI
Co-supervisor:	Agnar Aamodt, IDI
	Toril A. Nagelhus Hernes, SINTEF

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Segmentation of images has always been an application dependent and difficult task.
There have been several attempts, but artificial intelligence (AI) has never been able to really solve the problem of segmentation.
This work will focus on using case based reasoning (CBR), a recent AI method, for controlling the segmentation and improve it if necessary.
The challenges that must be met are comparison of cases, judging the segmentation result automatically and to choose the appropriate action to correct it should the result be poor.

Assignment given: 12. January 2007
Supervisor: Richard E. Blake, IDI

Preface

This report is written as a Master's Thesis at the Norwegian University of Science and Technology (NTNU).

I would like to express gratitude to my supervisor Richard E. Blake (NTNU) as well as my co-supervisors Agnar Aamodt (NTNU) and Toril Hernes (Sintef) for helping me with strategic choices in system design and for good advice in writing of this report. I also want to thank Frank Lindseth (Sintef) for helping me with technical issues in ITK as well as system design.

In addition, a huge "thank you" to my invaluable proof-readers.

Trondheim

Christian Marshall Rieck.

Abstract

This paper describes a case based reasoning system that is used to guide the parameters of a segmentation algorithm. Instead of using a fixed set of parameters that gives the best average result over all images, the parameteres are tuned to maximize the score for each image separately. The system's foundation is a set of 20 cases that each contains one 3D MRI image and the parameters needed for its optimal segmentation. When a new image is presented to the system a new case is generated and compared to the other cases based on image similarity. The parameters from the best matching case are then used to segment the new image. The key issue is the use of an iterative approach that lets the system adapt the parameters to suit the new image better, if necessary. Each iteration contains a segmentation and a revision of the result, and this is done until the system approves the result. The revision is based on metadata stored in each case to see if the result has the expected properties as defined by the case.

The results show that combining case based reasoning and segmentation can be applied within image processing. This is valid for choosing a good set of starting parameters, and also for using case specific knowledge to guide their adaption. A set of challenges for future research is identified and discussed at length.

Contents

1 Introduction

The computer has made life a lot easier in a number of ways, but in a few areas it still has a way to go. When it comes to analyzing images and retrieving semantic information, it can not do all the work for us. It is good at acquiring and displaying the images, but has little idea of what it is. Because of this somebody has to evaluate the image and attach the desired semantical information to it, often proven to be very time consuming. When the information needed is intended to guide noninvasive surgery, for example, the patient is left waiting while this is taking place. If a computer could do it in a matter of minutes or seconds the operation would begin sooner.

The area of research that deals with interpreting images to give semantic information is called machine vision. It can be divided into a number of sequential steps with its own task, preparing the data for the next step. A common pipeline is shown in figure 1.

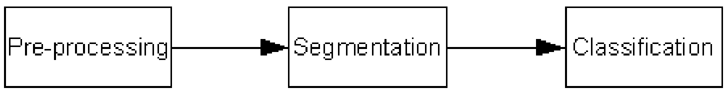


Figure 1: In machine vision it is common to send the input image through a series of steps before determining what the image represents.

The first step is usually image enhancement where the objective is to prepare it for segmentation. The most common operation is to reduce any noise present. Noise is artifacts introduced into the image by the image capturing device and is not part of the object being captured. This might obscure edges or boundaries and make segmentation harder. Another operation that might be done is to normalize the image with respect to color, or center it around a known point in the image. The next step in the pipeline is the segmentation. Its objective is to divide the digital image into regions, or segments, that correspond to objects in the natural world. This can be a car from a road scene, a tree from a picture of the country-side or a cruise missile from a surveillance satellite image. At this step it is not known that it is a car, tree or missile, it is only known that this set of pixels defines an object that is not the same as the neighboring one. The result is an image where each pixel is assigned to a segment. The last step takes the different segments and uses a classifier to determine what real world objects they represent. This step is now considerably easier because of the segmentation. The problem has been divided into several smaller problems, each problem being one segment. Without this the classifier would have to identify the image as "a room with a chair and a desk with a phone on it". After segmentation the classifier can determine "room", "chair", "desk" and "phone" separately, a much easier job. The outcome of this classification is heavily dependent on the outcome of the segmentation step. If the regions correspond poorly with their real life counterparts the task of identifying them may be impossible. It is therefore very important that the outcome of segmentation is as good as possible.

There has been much research into segmentation algorithms, and many attempts have been made to build systems that utilize them best. A problem with this is that the algorithms often come with a number of parameters to be set, and these parameters influence the result significantly. If a system is to work without human intervention it has to set these itself. As even experts find this task hard it is not evident how to get the computer to do it. In the 1980's a popular attempt was to use a rule-based approach. An analysis of the domain in question would give a set of rules that determined what should be done in different situations. Depending on the domain, this may be very hard and time consuming. Since then Case Based Reasoning (CBR) has emerged. Instead of relying on rules extracted from a domain CBR tries to use specific, relevant lessons learned earlier. This makes it appropriate for domains where the theory is not fully understood and thus such rules are hard to extract. Segmentation is one such domain.

This work will focus on using CBR to realize automatic segmentation. The parameters needed will be provided by earlier cases, possibly with some modification to suit the new problem better. The specific questions it hopes to answer will be presented later, in section 1.3, followed by the research methodology. However, to make the reader familiar with both segmentation and case based reasoning these two topics will be introduced. If this is familiar ground they can safely be skipped.

1.1 Segmentation

According to Pal [29], there is no single best segmentation algorithm. To further complicate it, there might not even be *a* best segmentation of an image, because it depends on what you are looking for. When finding the car in the road scene mentioned earlier, is it satisfactory just to find the car? Or are the different parts of the car such as wheels, doors and windows of interest? Different levels of detail may require different algorithms or at least different settings of parameters for one.

There are many different segmentation methods. To give the reader an introduction to segmentation some different categories will be explained, based on [43]. All algorithms or systems mentioned are explained in this book unless any other references are given. Anyone familiar with segmentation can skip this section. Surveys on the topic can be found in [29, 33].

1.1.1 Intensity-based

The simplest of segmentation algorithms consider nothing but the intensity of a pixel. This can be done by setting a threshold and consider all pixels with less intensity to be background, and the rest as foreground. In a controlled environment this is a quick and efficient way of segmenting dark objects on a light background. Such a setting is common in simple quality control in factories where the silhouette of the object is found and compared to a known norm.

The key to getting good results is to set the threshold correct. This can be tricky, but any *a priori* knowledge can guide the search. If it is known how many percent of the image belong to the object the threshold can be set so that this percentage is met. Different objects may have different colors and thus a sensible threshold is somewhere between these colors. This can be done by inspecting the image histogram.

1.1.2 Edge-based

A segment can be defined both by its region and its borders. If the borders are known the segment is within these, and if the region is known the border outlines it. So one method of finding a segment is to finds its borders, or edges. Such techniques are called edge-based and are some of the oldest in segmentation. They examine the image and look for discontinuity in gray level, colors, texture, etc assuming this indicates the end of a region. In color and gray level the derivative of the image will reveal borders, and several operators to find these have been defined. They are convolved with the image matrix and some examples can be seen in figure 2.

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

(a) Roberts operator

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

(b) Laplacian operator

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

(c) Sobel operator

$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

(d) Mexican Hat

Figure 2: Different edge detection operators

Many images have some noise present that will show up as edges in the result and hence images are usually smoothed prior to convolution to produce better results. After the convolution the result is usually thresholded to remove spurious edges. When only the strongest edges remain a post processing step is executed to connect edge parts that correspond with objects in an image. The Hough-transform is an example of this, designed to discover lines or circles in an image based on a partial match after edge detection. It has later been extended to find arbitrary, predefined shapes.

1.1.3 Region-based

These methods try to locate regions in an image that are homogeneous with respect to some property. In the end the result must satisfy these conditions:

- All pixels in a region are homogeneous.
- The merging of two adjacent regions will not result in a homogeneous region.

Two simple approaches are to start with the image as one region and split it recursively until the conditions are met, or to start with each pixel as one region and merge as much as possible. Large homogeneous (in gray levels) regions appear as flat areas in a landscape where the height is defined by the color value. This observation has led to the development of watershed algorithms, modeled after how water flows. When it rains all the water will flow downhill until it finds a minimum where it gathers and forms a pond. If the rain continues these pools will grow and eventually merge. Where they merge a dam will be built to keep them separated, and the rain continues until the world is flooded. Each pond now represents one region, and the dams their borders. Region growing methods accept one or more seed points from where a region will be grown. If one of its neighbors passes the homogeneous test it is included in the region. This is repeated until none of the region's neighboring pixels pass this test.

1.1.4 Model-based

Some systems have left the realm of all-purpose segmentation and focused on a small area. They focus on one or more specific objects to be found and have substantial information to aid the search. The low level techniques previously mentioned consider only a local property of the image and can thus make mistakes on a larger scale. With a model several edges can be judged as to how they fit together on a larger scale than just their individual strength.

Deformable models combines both a top-down approach based on *a priori* knowledge with a bottom-up approach based on image features [26]. First a model with the size and form of the expected region is placed at the expected location. This model comes with physical properties, usually tension and rigidity. Based on the image to be segmented, desired features are extracted and used to create the image forces. These forces can be seen as gravity, trying to pull nearby mass (the model) towards its lower (darker) regions. Based on the tension and rigidity of the model the image forces will deform the model, pulling the contour towards edges in the image. When the internal and external forces are equal the model will stop to deform and the intended object is assumed segmented. For this to work it is very important that the initial contour is placed correctly, or else it may be drawn towards the wrong image features.

1.2 Case Based Reasoning (CBR)

Case Based Reasoning had its beginning in Roger Schank's work in psychology and research on how humans use experience from specific episodes when solving problems [39]. It has since been adopted by other fields and thus "CBR" has different meaning to different people. Aha [4] lists three alternatives.

1. A psychological model for cognitive processing
2. A method for problem solving
3. A design model for expert systems

It is in the second alternative CBR will be used in this project. Aamodt et al [3] divided the overall process into the four smaller consecutive tasks Retrieve, Reuse, Revise and Retain, popularly named the four RE-s. Each of these will be explained separately in the following sections, based on [3, 7]. How they interact can be seen in figure 3.

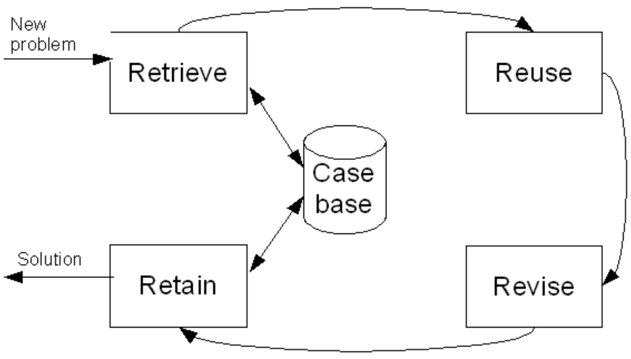


Figure 3: The CBR-cycle with the four REs. Retrieve receives the incoming case and finds the most similar case from the case base. This case is given to Reuse which applies the solution to the problem. Revise makes sure the solution worked and Retain is responsible for storing any lessons learned to the case base.

1.2.1 Retrieve

The system starts with the retrieval of an old case from the case base. The goal is to retrieve the case that is most suitable for solving the new problem at hand. This requires the user of the system to provide a description of the problem. Based on this the similarity to all earlier cases are calculated and the best one is chosen. Various indexing schemes and a multi layered approach may speed up the search and avoid comparing all cases to the original. For instance, in the PROTOS[34] system each case is associated with a *category*. The problem description is first examined and the best category is found, and then the cases under this category are examined more closely. Due to the fact that all cases are not examined the best may be missed.

There are a number of different ways to assess similarity, ranging from simple to sophisticated and knowledge-based. A very simple one could iterate through the indices and increment a counter each time the two cases have the same value for an index. The case that obtains the highest count wins and is selected. More advanced methods try to analyze the situation and understand the actual difference between cases. Based on a semantic web and a domain specific model it may be established that "car" and "Toyota" are related words and should be considered somewhat equal. CREEK [2] combines CBR with a domain specific model to explain any differences. One such explanation could be "it is OK that case A has a descriptive feature i ("flat battery") that case B lacks, because B's feature j ("left with lights on overnight") is known to cause i." In this manner two features can be matched and their actual difference computed. In this scenario the system can conclude that the two cases are more similar than they first appeared. Other systems that combine domain knowledge with CBR are CABATA [22], CABARET [37] and SAXEX [5]. For more information on these and others, see [25].

It is common to assume that the most similar case is the best case for solving the new problem, but it is not the only option. Sometimes the most similar case can be hard to adapt to the new case. This has spurred some research in adaption-guided retrieval where domain specific adaption knowledge is used to guide the search. In systems that recommend products based on a search or earlier shopping history, the most similar products may in fact be too similar. To combat this challenge one may use diversity-conscious retrieval mechanisms to ensure a good selection. Another way to go is to use compromise-driven retrieval if no cases match the initial query. Some similar cases may have important parts missing and are not acceptable and should thus not be presented to the user. No matter what similarity measure or search strategy is used, Retrieve results in a case that will be used to solve the new problem, which is the task of Reuse.

1.2.2 Reuse

After a case is selected from the case base it is time to apply its solution to the new case. This can essentially be done in two ways. One option is to just copy the solution, the other is to adapt the solution. To copy the solution is often used in classification systems where the new case is assumed to be of the same class as the most similar.

If decided to adapt the solution, instead of copying, a new choice appears, the choice between *derivational reuse* and *transformational reuse*. In transformational reuse some changes are done to the solution before it is accepted. Domain specific knowledge is used to create rules for such changes and index them according to differences in the cases. Then the system can detect differences in the two cases and apply the appropriate rule. The making of adaption knowledge is a hard task. In fact Khan et al [18] states that creation of the necessary adaption rules is a major bottleneck in development of CBR systems. Leake [20] says the problem is so hard that many drop the adaption step altogether. Therefore there has been some research into automating it based on the contents of the case base. See the survey be Mantaras et al [7] for more.

The other option when using adaption, derivational use, is to use the problem solving method itself instead of the solution. For this to work each case contains how it arrived at a solution in addition to or instead of the solution. This can be traces of goal/subgoal decomposition and what operators were used to solve each subgoal. Afterwards these steps are redone on the new problem.

1.2.3 Revise

After Reuse the system has generated an answer to the given problem. It is the responsibility of Revise to make sure this solution is correct. Some systems like CHEF[36] have an internal model which can simulate the solution, but often this step is done in the real world. The result must be entered into the system when it is known whether it works or not. Depending on the nature of the problem, it may take months to verify this. In the meantime the case can be stored and tagged as non-verified and not be used. It may also be tagged as “unverified” and get a less favorable status in the system, but it will be available if no other case is better.

If the solution is wrong there is a good chance that the system may learn from this experience. Given an explanation to its failure, from internal simulations or external users, rules can be created to avoid making this error in the future. These rules will then be utilized in the Reuse-step of later executions of the system. After Revise fails the solution is modified to provide a correct answer. This is usually followed by a second Revise-step to make sure the new solution is correct.

1.2.4 Retain

Retain is the step where the system learns. Every case encountered gives the system a possibility to use this experience later. Properly indexed failed cases can tell the system never to try a particular solution on this problem again. This information will be given be the Revise step if it concludes the initial proposed solution cannot be adapted to solve the new problem (and thus through similarity assessment, solve similar problems). Another option is to change the best case found to indicate that it also solves problems of this new kind. Sometimes the user must specify a solution in the Revise-step. Here the incoming case with its new solution can be saved as a new case and thus expand the number of available cases and solutions in the future. Sometimes the most important thing is to save how the solution was created and modified from another solution.

The next problem is to decide what indices will be created to help the system find the new case later. How similarity is determined plays a large role in this part. Where the case is represented as a vector with key/value pairs saving this vector is adequate. If the case base is hierarchical, such as in Protos, the case must be analyzed and put in the correct place. Creek has weights that tell which features are most important when judging similarity for a case. These weights must be updated.

When CBR was in its infancy it was believed that every case learned would improve the system. As more systems were implemented this assumption proved to be false. The *utility problem* appeared as the increasing size of a case base would degrade the systems performance, most notably at retrieval time. At some point the new experience learned is of less value than the increased cost in matching. At this point it is time to prune the case base and delete unnecessary cases. Care must be taken to ensure that the overall competence of the case base is not reduced as a result. Examples of case base maintenance can be seen in [21, 42].

1.3 Goal

The goal is to investigate segmentation guided by CBR principles. A CBR system will be built where the case base is a collection of earlier images and the necessary parameters to be able to reproduce the segmentations. When confronted with a new case one solution from the case base will be retrieved and used as a basis for solving the new problem. The system will be fully automatic from the user has entered the incoming problem and until the solution has been presented.

Retrieve and Revise will be given special attention, and in particular there are two questions that must be answered:

1. How to assess similarity between two cases?
2. How to use *a priori* knowledge extracted from a case to
   (a) Help Revise judge the output from Reuse
   (b) Guide the adaption of the solution

For question 1 the usefulness of the best case found is what matters. The time it takes to search the case base will not be an issue. Question 2 contains two subproblems that affect each other. The goal is to show that by a joint effort between Reuse and Revise the program can correct its own mistakes and execute proper repair methods.

Retain is not focused in this work. The system may create a new case based on the newly solved problem, but any case base maintenance will be excluded.

1.4 Methodology

The questions stated in the last section will be answered by empirical research. Two different ways of matching cases will be tried and their ranking of cases will be tested to see whether they sort cases in the correct order. If one method orders cases 2,3,5,4 (in that order) as most similar to case 1, will the result of using case 2 to solve 1 be better than 3,5,4? How the different methods perform will be analyzed in light of the domain chosen. The conclusion may not hold for other types of images.

In the matter of *a priori* knowledge the investigation will start with no adaption at all. Based on how the attempts fail new *a priori* information will be added to help Revise determine if Reuse's output is plausible or not.

How the system is performing will be monitored by comparing the output with a ground truth for each image.

1.5 Outline

The next section will present previous research that focused on CBR and segmentation. It will also include research originally intended for other areas but can be used here, such as image comparison. Section 3 will present the system that is being built. It will show how the experiments are set up, conducted and the results measured to help any interested party in reproducing the results. This section includes what segmentation algorithm was chosen and why, contents of each case and an introduction to the Repair-cycle that will enable the system to react to segmentation failures and take countermeasures. This is followed by a presentation of the cases in the case base. How Repair is implemented will be presented in section 5. It also shows how this part of the system performs, with scores given for all cases and with three of them being explained in detail. Section 6 contains results from the comparisons of two different ways to match cases during retrieve. The combination between CBR and image processing is fairly new and many thoughts emerged throughout the project. These are presented in the discussion in section 7 and some final words on future work are given in section 8.

2 Related Research

The system as defined will use CBR in image segmentation. Broken down this means finding a solution to the four RE-s. For retrieval it is interesting to see what can be found on image comparison, and if it can be used as a similarity metric. More on this in section 2.3. The reuse will simply be to use the parameters found in the best case in the already determined segmentation method. More interestingly is Revise, who's job it is to see if the proposed solution is in fact a solution. It will need a way to judge the outcome of Reuse in an objective manner. Earlier research will be discussed in 2.2. But first earlier approaches to a combined use of CBR and image segmentation will be presented.

2.1 CBR and Images

It is evident that there is a lot of literature on image processing in the field, and an increasing number of papers on CBR are published. However there is surprisingly little overlap between these fields, only a few papers and systems have been found. In [31] Perner arguments for why CBR should be used to segment images and also describes the architecture for such a system. She claims that compared to what CBR can offer, the recent segmentation systems are lacking in both flexibility and robustness. Traditional systems try to get a set of optimal parameters that work well with its chosen domain. This will guarantee a best fit over the available test data, but does not guarantee an optimal segmentation per image. The SCINA system [12] was designed to interpret SPECT[1] images and assess coronary artery disease. The end result is not a segmented image, rather it is a diagnosis for the eye. No segmentation takes place. In the case base each case is stored as a 6×6 matrix of integer values that corresponds to the level of tracer fluid in the eye. For the input image this matrix is compared against all other matrices to exclude all but a few cases for further comparison. The matrix to matrix similarity measure uses weights for the importance of areas in the eye and combines this with use of a polar data map. After this the remaining cases can be adapted to look more like the input case. This part is rule driven.

Ficet-Cauchard et al [9] built a CBR system to reuse existing image processing knowledge on new images. The sought output is a plan to be executed on the image in order to get the desired result. One case does not contain every step of a plan, nor does a plan reside in one case alone. Some cases contain a plan for a particular small task while others define a task/subtask decomposition in a tree structure and let other cases solve each subtask. This may go on for several decompositions. While matching it is the top-level case of a plan that is matched. After this the tree is traversed and subtrees might be changed. To allow this every case has information regarding its position in the tree, under what circumstances it is applicable and what it does. If the retrieved plan has a step for "smoothing noisy images" and it is known that the new image is already

[1]Single Photon Emission Computed Tomography. The patient is injected with a radioactive isotope which decays and emits gamma rays in the process. These rays are used as a source of information to render a 3D image [13].

smoothed, the plan is adapted to remove this step. The similarity assessment is done using a weighted average of the similarity between criteria, see equation (1).

$$\Phi_t(S,T) = \frac{\Sigma(\alpha_{Cr} \times \phi_{Cr}(S,T))}{\Sigma\alpha_{Cr}} \tag{1}$$

α_{Cr} is the weight for criteria Cr and $\phi_{Cr}(S,T)$ is the similarity function for Cr giving a similarity $[0,1]$ between S and T. In Ficet-Cauchard's model several ϕ_{Cr} are implemented, such as:

1. If T=S then 1, else 0
2. A gradual score from 0 to 1 based on some distance
3. The distance between S and T in an ordered set
4. The number of common elements in a set

In this way different criteria might have different similarity functions.

A system for the interpretation of Computed Tomography (CT)-images has been developed called ImageCreek[11]. It does not segment the image but rather works on a segmented image and tries to give semantical meaning to the segments. The system is divided into two layers where one is driven by the other, both using CBR. On top is the holistic layer which is responsible for the interpretation of the image as a whole. Beneath is a segment layer that receives a segment and works on this as an isolated case. The result will be reported back to the top layer. It is the job of the holistic layer to ensure that these results fit together according to domain specific rules. Both layers work in a Propose-Verify-Critique-Modify manner. Propose proposes a set of solutions to the task at hand. These are verified in the next step to make sure they actually solve the problem. If the verifying fails it is the job of the critique step to figure out how to adapt the failed solution so it can be verified. The last step, modify, executes the adaption strategy from the critique step.

In [24] Lorenzo-Valdes et al describe how to segment the left and right ventricles of the heart. Previous images of the heart are segmented and the results are combined into an atlas where each region is defined. This is then warped to look like the incoming image, usually by registration (will be explained in section 3.4). Each region in the new image will now overlap with the regions in the atlas, and thus the segmentation can be copied into the new image. It is important that the warping produces two similar images for this to work, and therefore they cannot be too dissimilar at the start of the process. With hearts and other human organs this is not always easy as there can be large differences between individuals. Lorenzo-Valdes solves this by creating a population-specific atlas by basing the atlas on "normal adults", same as the test subject. If this is to be used on a child a different atlas must be used. This can be seen as a weak form of CBR where the problem is mapped to the best available *a priori* knowledge of problems of this sort.

In [30] Perner has designed a system to give a ratio of brain vs liquids in CT images. A case base of previously processed images is created, and along with the original image the case contains the optimal parameters for how to segment it. The segmentation algorithm is the same in all cases, but the parameters vary. Case matching was done by using both image and non-image data. For the non-image data the sex and age of the patient together with the slice thickness and number of slices (in the image) were used. Similarity of image data was computed with a distance function by Zamperoni et al [46], and one was based on statistical features. In the statistical approach the eight features given in table 1 are extracted and stored within each case.

Mean	$\bar{g} = \sum_g g \cdot H(g)$	Variance	$\delta_g^2 = \sum_g (g - \overline{g})^2 H(g)$
Skewness	$g_s = \frac{1}{\delta_g^3} \sum_g (g - \overline{g})^3 H(g)$	Kurtosis	$g_k = \frac{1}{\delta_g^4} \sum_g \left((g - \overline{g})^4 H(g)\right) - 3$
Var. Coeff.	$v = \frac{\delta}{\overline{g}}$	Entropy	$g_e = -\sum_g H(g) \log_2[H(g)]$
Centroid x	$\overline{x} = \frac{\sum_x \sum_y x \cdot f(x,y)}{\overline{g}S}$	Centroid y	$\overline{y} = \frac{\sum_x \sum_y y \cdot f(x,y)}{\overline{g}S}$
g = intensity value, N(g) = # pixels of color g, S=total number of pixels			

Table 1: The eight statistical features extracted from an image in Perner's image CBR system.

Upon retrieval these features are extracted from the new image and the distance between the cases is calculated with the following equation:

$$dist_{a,b} = \frac{1}{k} \sum_{i=1}^{K} w_i \left| \frac{C_{iA} - C_{i,min}}{C_{i,max} - C_{i,min}} - \frac{C_{iB} - C_{i,min}}{C_{i,max} - C_{i,min}} \right| \tag{2}$$

where C_{iA} is the feature value of feature i for image A and $C_{i,min}$, $C_{i,max}$ are the minimum and maximum values for that feature over the entire case base. w_i is the feature weight, all set to 1 in her paper. The statistical similarity assessment has a better result in addition to being much faster. No comparison with other systems is offered but the author claims it gives "superior performance".

2.2 Evaluating a Segmentation

Unfortunately it is not straight forward to judge the output of a segmentation. In addition, both [29] and [6] states that research in this area has received much less attention than the segmentation algorithms themselves. These evaluation methods can be divided into three groups. One is the analytical group, where the algorithm itself is analyzed in terms of functionality, principles and properties. The two remaining groups, goodness and discrepancy methods, are both types of empirical methods. Goodness methods examine the output for different features known to a good segmentation such as low intra region variance or high inter region variance. Based on how the result complies with such measures a score is calculated. The third group, discrepancy methods, uses a known "ground truth" or "gold standard" that is the best segmentation of the image in question. (A best segmentation for the application at hand). This is often a segmentation done manually by an expert.

Somewhere in the middle of discrepancy and goodness methods is a system by de Graaf et al [6]. They describe a method based on the cost of editing a result from an automatic segmentation, *segment distribution*, to a gold standard, *object distribution*. This requires that the segmentation algorithm has the capability to create a fixed number of segments. An example of this is region growing methods. Two types of editing are permitted, splitting and merging of regions, with an individual associated cost k_M and k_S. These costs should be based on the available tools for these operations, as the object is to minimize the time needed for manual editing of the result later. The segmentation algorithm is run with an increasing number of segments. Each result is judged by the equation (3).

$$Q_s = \frac{1}{N} \sum_{i=0}^{N_s} \sum_{j=0}^{N(s_i)} T(j \subset s_i) \tag{3}$$

N_s is the number of segments and $N(s_i)$ is the number of pixels in segment i. These pixels may overlap with a number of segments from the *object distribution*. One of these, segment j, has the biggest overlap. T() then equals 1 if the pixel in segment s_i overlaps with a pixel from segment s_j, 0 otherwise. All, if any, of the segmentation results with Q_s higher than a threshold Q_{thr} will have its cost of manual editing computed. The one with the lowest editing score is deemed the best. In the paper ([6]) an example is shown with a slice from a Magnetic Resonance (MR) scan of a person's head. The *object distribution* has three segments and the best *segment distribution* found has eight. The quality of the print makes it hard to judge if the result would be good if these eight are combined in a sensible fashion and reduced to three.

Huo et al [14] have a fairly simple evaluation, they use the percent of overlap between the automatically segmented regions and an expert segmentation.

$$Sim(A, B) = \frac{A \cap B}{A \cup B} \tag{4}$$

This is not used to evaluate a specific segmentation as good or bad and hence request a new segmentation, rather they use it to evaluate the segmentation method itself for a particular domain. Three radiologists segment 96 mammography images by hand and evaluate if any mass lesions are malign or benign. For the automatic segmentation the algorithm is given a 512^2 region of interest centered on the abnormality in question. From there it segments the image based on a region-growing technique. From each segment/lesion in these images four features are extracted and combined with an artificial neural network to give a probability of it being malign or not. The same is done with the automatically computed segments. A comparison on a segment-to-segment basis shows that the computer undergrows the regions, on average only marking about 75% of the pixels the radiologists do. Despite this the results from the feature extraction and classification based on the computer generated segments are not much worse than the ones based on the manually marked segments. Based on this they conclude that even if the segmentation results are not perfect the algorithm used is adequate for solving this problem. ImageCreek used domain knowledge to judge the result from the segmentation. Its overall goal was to give semantic meaning to the segments, which corresponds with the classification step in the image processing pipeline. If, however, the system was unable to infer something reasonable from the output it could ask for a new segmentation as it believed the current one to be flawed. This can be seen as a success/failure judgement of the result.

Through the *a priori* nature of CBR both goodness and discrepancy method are candidates for Revise. Features from a particular output can be extracted and stored with the case in addition to the solution. Based on this features from the outcome of the new problem can be compared to the expected values. There might even be a possibility to store the segmentation result itself in a case and use this as ground truth when judging the new case.

2.3 Comparison of Images

As with segmentation, a comparison of images is not straight forward. To provide an overview of the literature some systems and methods will be discussed from three different approaches to the problem. These are matrix-based, feature-based and structural similarity.

In the matrix-based approach the data to be compared are the intensity values of the image. The simplest is just a subtraction of one image from the other. If the result is an image where all pixels have intensity 0, the images are equal. Another example is the Hausdorff distance [15] [45]. It is defined as

$$H(A, B) = \max(\max_{a \in A} \min_{b \in B} ||a - b||) \tag{5}$$

where $|| \star ||$ is an arbitrary function (here $a - b$) on the points in A and B, e.g. the Euclidean distance. For each coordinate the distance from a pixel in A to the nearest pixel in B is calculated. In the sense of Euclidean distance, the distance will in a 2D image be the distance in the 3D space based on coordinates and intensity. To save time it is common to only compare pixel $a \in A$ to the pixels $b \in W_B$ where W_B is a subwindow of B. The distance is also calculated for each pixel $b \in B$ to $a \in W_A$. The largest of all these distances will be the distance between the two images. Mutual information [8] uses statistics to assess the similarity. Given two images A and B to be measured, mutual information tries to establish if there is a dependency between the intensities of pixels at coordinates (x, y) in A and B. If the images are equal then an intensity x in image A will always yield intensity x in image B as well. A major advantage is that the proximity of the intensities in a color space does not matter. For instance, say you invert the colors of image B. There is still a strong dependency between them, the intensity i at (x, y) in A will still always predict intensity j at (x, y) in B. And thus they contain the same information and are semantically the same.

The feature-based approaches try to extract either numeric or symbolic features from the image and use existing machine learning algorithms to evaluate similarity. Santini [38] developed a similarity measure called *Fuzzy Feature Contrast* based on Tversky's *Feature Contrast* [44]. Whereas in Tversky's measure the features were binary, Santini would represent the absence or presence of a feature on a scale $[0, 1]$ where 0 is not present and 1 is present. The objective of the research was to provide a similarity measure that assessed similarity in the same way as humans do, that it ordered a set of images in the same way human test subjects did. They claim it is the *perceptual distance* that matters, as if they are to be called “intelligent” they must give the same answer as a human would. In [32] Perner describes a system for automatic classification of airborne fungi. From the image attribute/value pairs such as *color = brown* or *contour=double contour* are extracted and fed to a CBR system. This will then match the new case to the cases in the system and classify it. If it is too dissimilar from existing cases it might be determined that it is a new type of fungi and give it a new classification and thus learn. Mehrotra [27] retrieves images with similar shapes as the query. All images are processed in advance to obtain the shape boundaries. These are processed and turned into a scale, translation and rotation invariant point in multidimensional space. The user can then search for an exact match (useful for finding shapes in CAD-models, computer graphics or other non-noise environments) or a similar match (useful in real life data). In [40] Smeulders et al give an extensive survey of content based image retrieval systems designed before 2000. This contains an in depth discussion on similarity between features, object silhouettes, structural features and salient features. Puzicha et al [35] give an empirical study of nine dissimilarity measures for color and texture and conclude that they found no over-all winner or loser. Each task requires its own tool.

In structural similarity one may use an attributed graph to represent the image. VisualSEEk [41] extracts information on the size, color(s), relative and absolute position of different regions and saves this in a database. A user may later build a query by positioning different regions on a grid an let the system find similar settings in the database by graph matching.

3 System Architecture

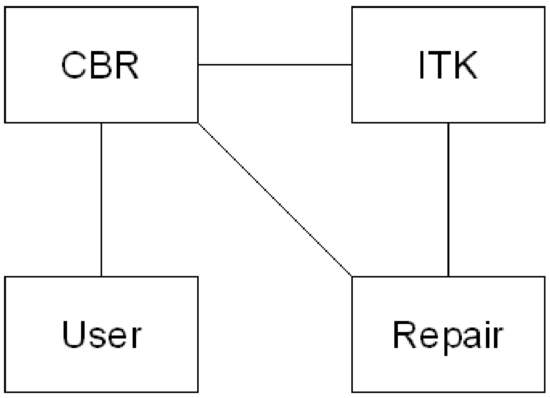


Figure 4: How different parts of the program are connected.

The system will contain three parts, a case based reasoning part called CBR, an image processing part named ITK and one Repair part. How they are connected can be seen in figure 4. CBR is responsible for accepting a problem description from the user and to retrieve the case it judges as best. Named after the image processing library, ITK will be responsible for all image processing. The third part, Repair, is responsible for coming up with adjustments to the proposed solution should Revise decide that the attempt failed. It takes whatever information needed from the case and analyzes the segmented image to decide what should be done. Repair will be further explained in section 3.6. How they interact during the execution of the program will be explained in the rest of this chapter.

3.1 Program Flow

Figure 5 shows the flow through the system. The system will to some degree follow the traditional lines of CBR systems. Each new case, in this setting a 3D MRI volume, is given to the system as the problem description. The task is now to find the solution s from a set of solutions S that segments the new case best. S contains tried and tested solutions that are known to segment a previously encountered volume in a satisfactory way. How these solutions are found and how they work is elaborated in section 3.7.

The way to get s is to go through the image volume s is connected to. This is illustrated with the first arrow in the figure, the matching of the new volume to the others. Further details on matching will be explained in section 3.4. The most similar volume, *msv*, has a pointer to a solution s. Since it is known that s segments *msv* in a satisfactory way, and the new case and *msv* are similar it can be assumed that s segments the new case in a satisfactory way as well. s is then retrieved and given to ITK for execution. This is illustrated by arrows 2 and 3 in the figure. How ITK applies the solution is detailed in section 3.5.1. After this the result is sent to Revise to be approved.

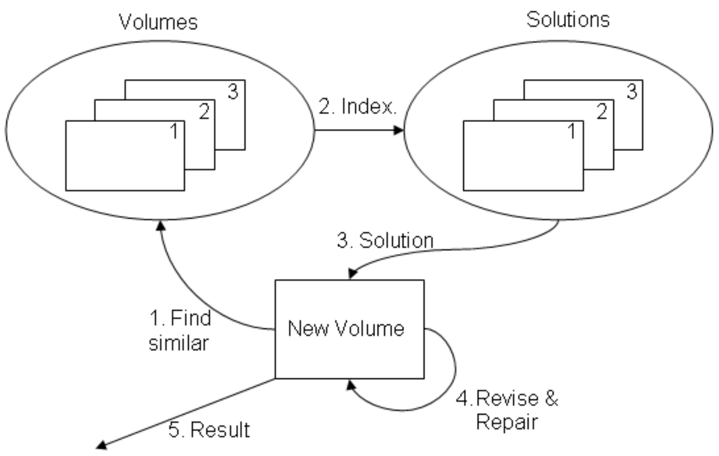


Figure 5: A simplified flow chart of the system. Each new case will be compared to older cases and the solution associated with the most similar case is retrieved and used. Thereafter the system will make changes to the solution if deemed necessary until a satisfiable result emerges.

Before the rest of the system is explained it should be mentioned what kind of data the system will work on. The volumes used were downloaded from the Internet Brain Segmentation Repository (IBSR) [2]. The IBSR is a web resource that offers real life data sets of medical images. Because of the requirement for patient privacy such images are in general hard to obtain. In addition, a hand segmented version of each volume is provided. As stated in section 2.2, an automatic evaluation of the result without a ground truth to compare with is difficult. With the inclusion of the segmented version researchers can test their proposed methods on the raw data and get an evaluation of its accuracy by comparing it with the solution. In fact the purpose of IBSR is to give researchers real life data and a solution with which they can verify their research. The other option would be to segment the raw data and then get a trained radiologist to comment on the accuracy at a later stage.

3.2 Image Processing Module

All image processing will be done with the Insight Toolkit (ITK) [1]. It is an initiative of the US National Library of Medicine of the National Institutes of Health and was started in 1999 and is still in development. Since it is open source software anyone can use and expand it as they like.

3.3 Case Architecture

A case needs two obvious elements. It needs a description of the problem it exemplifies, and the solution to this problem. The description of the problem

[2]The 20 normal MR brain data sets and their manual segmentations were provided by the Center for Morphometric Analysis at Massachusetts General Hospital and are available at `http://www.cma.mgh.harvard.edu/ibsr/`.

must be in a form that makes it possible to compare it to another description and determine if they are equal. If a particular problem is "open jar" the solution attatched to a problem indexed as "how to shower" is of no interest. Only cases indexed as relevant to jars are of interest. This system deals with segmentation of images and hence all cases stored in the case base are going to be examples of and solutions to "how to segment this image"-problems. Because of this the chosen description of a problem is the image the solution segments.

The solution is in the form of a set of parameters for the segmentation method. This set is known in advance to segment the image in the case in the best way possible. Finding these solutions and to build a case base is an important step in developing CBR systems, as without previous cases to be reused the system can not function. Section 3.7 describes how the sets were found.

There are two additional parts of each case beside the image and the solution. One is information used to verify the solution in Revise. This will be explained in section 5. The last piece of information stored in each case is the ground truth segmentation of the image. This is not used in the CBR process itself but rather in the evaluation of the system, explained in section 3.8 and is placed there for convenience. Figure 6 gives a graphical explanation to the content of each case. Note that only a 2D slice from the 3D data set is pictured in the figure.

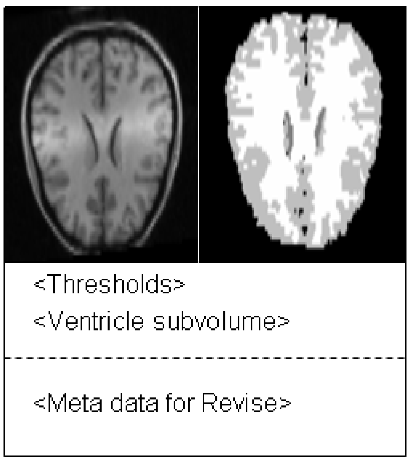


Figure 6: The contents of a case.

3.4 Case Matching

If the proposed solution is to be of any help, the input volume has to be similar to the volume just presented as a new case. As an example of this, see figure 7. In this case data set 1[3] to the right has been segmented using a threshold. The marked region (white) maps good with the ventricles, the desired structure. To the left data set 4 is segmented with the same threshold, and the failure is

[3]Unless otherwise stated, data set # refers to an IBSR data set. See appendix A for a mapping from number to name of file.

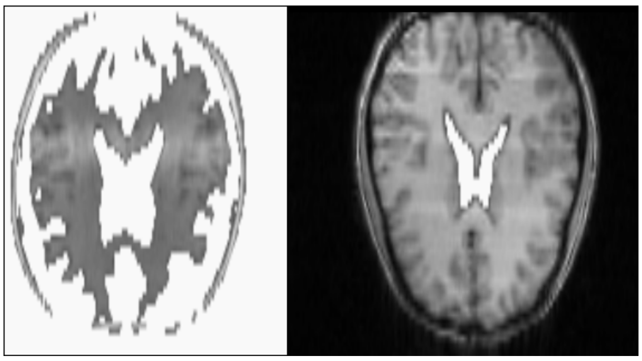


Figure 7: An example of what can happen if the proposed solution is tuned to an image too different from a new image. Left is wrong, right is correct. The same threshold is used in both images.

obvious. Voxels not members of the ventricle have been falsely marked as such. The cross sections chosen were number 140 in data set 1 and 152 in data set 4. A different slice from each volume is necessary to show the same view in both volumes, as the content on slice x from one volume does not necessarily match with the contents of the same slice in another volume. If this was true it would be known that the ventricles were to be found on slices XX through YY, simplifying things greatly.

The fact that slice x in one data set is not at the same location in the head as slice x in another data set introduces some problems. In voxel-to-voxel based matching two identical heads positioned at different coordinates in the encapsulating volume will not be judged similar at all. The modified Hausdorrf distance by Zamperoni et al [46] takes this into account by matching a voxel to a nearby set of voxels in the other image. This problem with displacement will not occur with feature based similarity measurement. The values extracted will be identical as long as the features are not position dependent. The features suggested in Perner's work [30] are based on gray levels and are position independent.

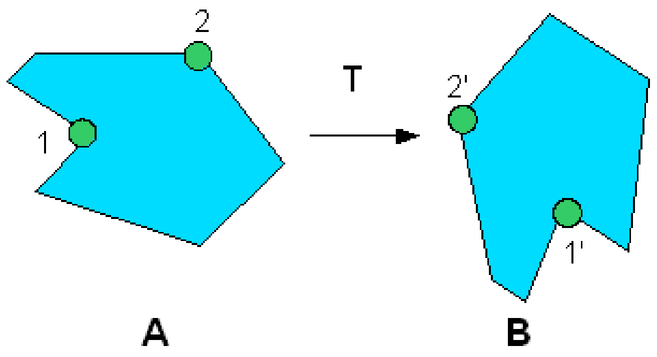


Figure 8: Points in image A are mapped onto points in image B by transform T.

Transform	VersorRigid3DTransform
Optimizer	VersorRigid3DTransformOptimizer
Sim. function	MeanSquaresImageToImageMetric
Interpolator	LinearInterpolateImageFunction
Max iterations	200

Table 2: The classes from ITK used for registration.

As will be evident in section 3.5.1 it is important that the two volumes are not only similar, but that the heads' position in them are aligned. This can be ensured by a process called registration. Registration means to find a transform T that transforms a set of points in one image to a set of points in another image. A simplified example of this is shown in figure 8. In this example T is a simple rotational matrix that needs to rotate image A 90° to correspond to image B.

In registration-specific terms image A is known as the moving image and B as the fixed image. This is because it is A that is transformed to match image B, which is stationary. Although it is easy for a human to spot a simple transform, like the one in the example, it is much harder for a computer. A computer has to guess an initial transform and apply it to the moving image. Then it must decide if the alignment is a better or worse match than before the transformation, and this is done with an image similarity function as explained earlier. Until the moving image has a close enough similarity with the fixed image this process is repeated. A natural question is to ask how to choose a new guess for the transform at any given point in time. It is the job of an optimizer function to decide this.

So regardless of the matching function chosen the new image and the image from the most similar case have to be registered. This will be done with ITK. The ITK Software Guide [16] page 400 describes the set up for a registration in 3D. This can be seen in the file `Examples/Registration/ImageRegistration8.cxx` that is included in the ITK installation. All registrations done in this work are based on this file. The details of the function are given in table 2. The transform handles both translation and rotation, meaning that the moving image can be moved around in the encapsulating volume and/or be rotated. The optimizer is a special optimizer designed for this transform. To compute the distance between two images the similarity function is used. The distance between two images are here computed as the square of the difference in intensity values voxel by voxel. Since voxels are seldom aligned between the two images after the moving image has been transformed an interpolator is used. A voxel in the fixed image ends up intersecting one or many voxels in the moving image. The interpolator computes a value to be compared to the voxel of the fixed image based on these intersecting voxels. Max iterations is the maximum number of iterations the registration process can make before it gives up and accepts its current transform as its best.

Since the registration process starts by computing a similarity, makes adjustments, computes similarity etc until it is satisfied it will at the same time give a measure of how similar the images are. The score given after the registration has been finalized can be used as a similarity measure between cases and is given automatically by a process that is needed anyway. Therefore this score is one option for computing similarity between cases. The other that will be used in this work is the one from Perner [30], extended to three dimensions. They will not be used at the same time but will rather be given the same tasks and have their results compared.

3.5 The Segmentation Method

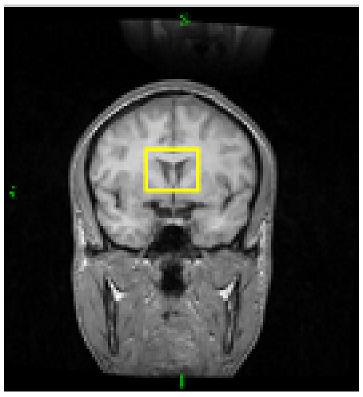
When choosing a segmentation method two aspects were taken into consideration, results and simplicity. The results a method produces are obviously important. If it can not solve the task at hand in a satisfactory manner nothing else can justify its use. Secondly the simplicity of the method was considered. Upon failure Repair has to make adjustments to the parameters of the method in a way it deems fit. When done manually this is done by an analysis of the output and based on the operator's prior experience some parameters are adjusted. This is usually a trial-and-error based search for a good result. When done automatically there must be some structures that decide how these adjustments are done. In order to design these the system designer has to understand the problem domain fully, and be able to implement this knowledge in the system. The simpler the segmentation method, the simpler this task will be.

Based on this a thresholding technique was chosen. The results were adequate and the method is fairly simple in its design. Other rejected methods are mentioned at the end of this chapter, section 3.9, where the rejection will be justified by the two considered aspects.

3.5.1 The subvolume

One of the parameters from the case is the coordinates of a subvolume of the entire volume. Why is this necessary? Thresholding works by judging each voxel against a threshold, hence its name. In this case it is appropriate because the ventricles have on average a noticeable lower intensity than its surrounding tissue, see figure 9(a). But other brain tissues do have as low intensities. Some of these are marked in figure 9(b). If the whole image was thresholded it would contain non-ventricle parts as foreground, which would be an undesirable effect. One solution, and the one adopted here, is to only threshold a part of the image. This part should be set such that all voxels of intensity lower than x belong to the ventricles. Note that an optimal setting would be the ventricle itself, as you are guaranteed not to include anything outside of it. In this case the problem of segmentation would be solved already and no thresholding would need to take place. To conclude, the subvolume should be easy to calculate and contain as few uninteresting voxels as possible.

Unfortunately it is not always as easy as to just use the bounding box blindly. The problem is illustrated in figure 10. The bounding box of the ventricles in image A may not enclose the ventricles in image B. (The size of A's ventricles was determined by the author, not a qualified radiologist.) This will happen if the bounding boxes have different size or are located differently. Since A


(a) Ventricles are darker than surrounding the tissue

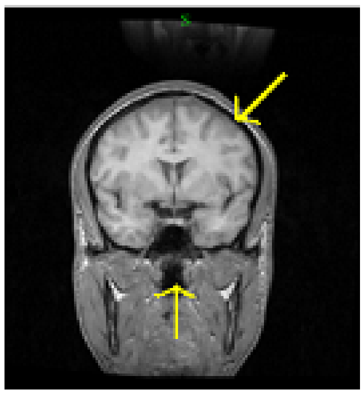

(b) Dark regions that are not part of the ventricles

Figure 9: Examples to illustrate some problems with thresholding the images. The image is frontal slice 28, data set 112.

and B have been through the registration process they should be aligned and hence their ventricles in corresponding locations. At this point three outcomes are possible, the proposed subvolume is too small, perfect or too big. The perfect one is of course not a problem. The second best alternative is the too big subvolume. It may include a number of non-ventricle voxels which the segmentation algorithm can misclassify, but it is theoretically possible to get a perfect segmentation. When the subvolume is too small however some parts of the ventricle will have no chance of being detected as they would be outside the area of interest.

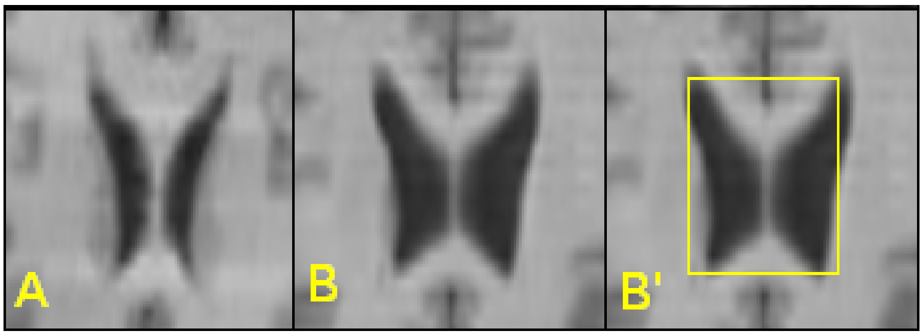


Figure 10: The bounding box of the ventricle in one image does not necessarily match another. Leftmost image is axial slice 124 from data set 1. The center and rightmost is axial slice 174 from data set 202.

The problem with different size in bounding boxes is recognized as a problem with this approach but will not receive any focus in this work. Two amendment strategies are either to include the necessary information in each case to be able to detect it mathematically, or to investigate along the boundary to see if it splits obvious regions. In the rightmost image in figure 10 the right border, top half, can be seen to have equal intensity on both sides and this could indicate an

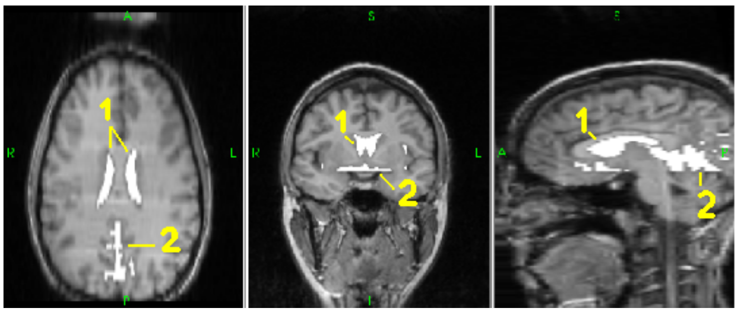


Figure 11: The white region marked 1 is a correctly marked ventricle. The region marked 2 is falsely marked as ventricles and is the region targeted for removal.

error. But as unwanted regions *should* be cut in such a fashion it is not straight forward.

3.5.2 The First Threshold

The chosen method was developed through a series of modifications to the original idea of thresholding the image. Even with the use of a defined subvolume, dark intensity voxels not belonging to the ventricles were present. They are largely present in one corner of the subvolume as can be seen in figure 11 and combined they usually outnumber the voxels of the ventricle. Once this was established a task was introduced to remove this region from the image before searching for the ventricles. The solution found was as follows:

1. Threshold the image with threshold T_{bg}.
2. Identity all voxels belonging to the biggest region.
3. Set those voxels to white.

It is important that as much as possible is removed, which means setting the threshold T_{bg} so high that the resulting region contains as many unwanted voxels as possible. A hidden danger is that there are a number of small resulting regions from this first step, among them part of the ventricles. If T_{bg} is too high these will melt together and become part of the biggest region and will thus be removed. Figure 12 illustrates this. In a) T_{bg} is 101 and in b) it is set to 102. Neither of them is perfect, however, as in a) the white region in the center of the image is in fact part of the ventricles and should not have been marked. Still, it is far better than b) where much more is falsely marked white.

Why white? At first this may not be obvious. Remember that the thresholding accepts values below a certain value. As white has full intensity the only possibility of these regions to be found is by thresholding at maximum intensity. This will set the whole image to foreground and will never be done. Thus the white regions will not be taken for ventricles ever again.

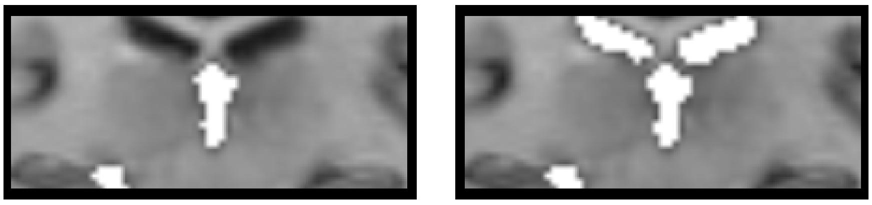

(a) Threshold set at 101 (b) Threshold set at 102

Figure 12: The effect of increasing the first threshold. At some point in time the ventricles will be considered part of the unwanted region.

3.5.3 The Second Threshold

After the first threshold the largest dark region in the image contains the desired object. Now the threshold T_v has to be found in such a way that it captures as much as desired, but not more. The fine thing however is that because of the first step T_v can be higher than it was before that step. Remember that T_{bg} was set just before the unwanted, big region BG and the wanted region V merged together. This means that what separates them are one or more voxels at $T_{bg} + 1$, a set of voxels called D. If the first step was not taken D separates two dark regions, V and BG, that will fuse at threshold $T_{bg} + 1$. If the first step indeed was taken, D now separates one dark region V and one white region BG. Increasing T_v to $T_{bg} + 1$ will only fuse D and V, not BG as it is set higher than any threshold that will be used. In this fashion T_v can be increased until the next time it is too large and accepts neighboring tissue.

3.6 Repair

Repair is inspired by the automatic revision of plans in Chef. It is not a step in CBR like the four RE-s, it is rather the name given to the extended cooperation between Reuse and Revise. After a solution to the initial problem has been found Revise will determine internally if it is sound or not. If not sound the solution has to be modified. Since this system uses derivational reuse the problem solving method will be altered, not the actual result. This is reapplied and the new result will be tested again and if necessary new modifications to the segmentation algorithm are made. Until the result is deemed acceptable the result will not be shown to the user. How Repair fits into the CBR-cycle can be seen in figure 13.

This focus on Repair is not customary in CBR. Also uncommon for CBR systems, the cause of failure will not be identified and incorporated into the case base. Properly indexed, knowledge from such failures can aid the retrieval of new cases in the future. Through a failure it can be learnt that case X does not solve problem Y and the next time Y or a similar problem comes along X is not used, even though it may be the most similar case. During the discussion in section 7 this topic will be dealt with again. This project will not focus on either detecting, utilizing or saving such knowledge. It is because the cause of failure is too hard to detect that this solution for Repair is used. Chef can

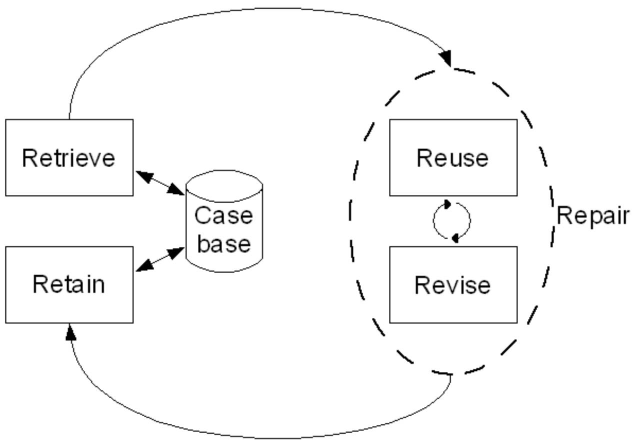


Figure 13: The Repair-module in the CBR-cycle

rely on lessons learnt from old failures in adaption and avoid making the same mistakes again.

This model for iterative improvement is influenced by earlier knowledge based systems such as [23]. The intention is to make the system aware of its own mistakes and to act upon them. This is the minimum requirement for a system that hopes to be seen as intelligent. Instead of relying on general domain knowledge as the aforementioned system does, knowledge will be based on case specific information. So instead of knowing that feature X is generally found in an area Y, X is now expected to be at a specific location. This may of course be allowed to vary due to individual differences, but with good case matching this difference should not be too great.

Adaption will be guided by a small set of rules found by trial and error. If the mass center of the region found is correct but it is too small, a threshold can be incremented to capture more voxels. If the mass center is far from the expected location a wrong region may have been marked. Based on earlier experience it can be decided how to react. How this is done will be explained in section 5.

3.7 Building the Case Base

Before a CBR system can be used to any degree the case base has to be constructed. Since a case is a problem/solution pair, a solution to the problem at hand has to be found. This takes form as the parameters needed in the segmentation algorithm. The chosen segmentation algorithm is fairly simple and has only two parameters, the thresholds. Also, the thresholds must be in the same range as the gray values it is possible to encounter in the image. Since the intensities in the images are scaled to be within the range 0, 255 this is a small enough parameter space to be brute forced. The best thresholds will be the set that maximize the overlap [14] of the proposed solution and ground truth. That is;

$$T_{bg}, T_v = \underset{T_{bg}, T_v}{\operatorname{argmax}} \frac{A \cap B}{A \cup B}. \tag{6}$$

where A is the segment from the thresholding and B is from ground truth.

An easy way to find the optimal thresholds is to try them all and compute the overlap for each. To the contrary, which part of the entire volume to apply the thresholding to can not be found in this way. The ground truth is used to do this. By traversing the voxels of the ground truth and noting the indices where the voxels belonging to the ventricles are it is an easy task to compute the bounding box. The space between the minima and maxima of the ventricles' position in the X,Y and Z dimensions defines the subvolume to use. It can not be smaller because then the sought object would not be segmented entirely. To increase the size will only include more voxels outside the sought object that potentially can be misclassified.

To summarize, the solution associated with each case will be found accoring to the following steps:

1. Locate bounding box of ventricles in ground truth.
2. Set this as the region of interest in the corresponding raw data.
3. For all possible thresholds, test and compute overlap.
4. Save the optimal set of thresholds.

3.8 Evaluation of the System

The evaluation of segmentation results are not always easy and as mentioned earlier, it usually employs a kind of ground truth. But as Kapur [17] discusses, the ground truth itself is not always easy to acquire. In the case of brain tissue segmentation the ground truth does indeed exist, but in a form that is hard to extract. Inside the patient's head the brain tissue has physical borders, but we are unable to extract them perfectly. With this in mind, she lists five different methods of segmentation validation;

1. Visual inspection.
2. Comparison with a manual segmentation.
3. Testing on syntactical data.
4. Use of fiducial marks on patients.
5. Use of fiducial marks on cadavers.

Visual inspection is the simplest of the five methods. It is subjective and has problems when it comes to segmenting volumes as people cannot see the inside of the object. A comparison with a manual segmentation is also relatively simple. One drawback with this method is that the manual segmentation is not necessarily correct. Kapur claims different experts can have as much as 15% variation in classification. With syntactical data the boundaries are known exactly, but it is difficult to generate models that capture the full complexity of the brain. As for alternative number four and five, fiducial marks can be inserted into patients/cadavers in known places. These can later be used as known samples to compare against the segmentation. This approach is highly invasive.

At the moment no trained radiologist is available to comment on the result, and this excludes visual inspection. The obvious lack of patients and/or cadavers also eliminates these options. This leaves manual segmentations and syntactical data as the only real options. Since manual segmentations were provided with the raw data sets from IBSR, method number 2 will be used. It will not be used in an application where an actual difference in the manual segmentation and absolute ground truth is catastrophic.

When the type of method is determined, the use of it remains to be explained. The system will be evaluated with leave-one-out cross validation. Each case will be removed from the case base and will be used as a new and unseen case. The score for each segmentation will be computed with the same formula used to build the case base, see equation (6).

3.9 Rejected Algorithms

The Inside Toolkit comes with a large manual [16] describing various aspects of its design and it explains how ITK works in great detail. One of its chapters is dedicated to segmentation algorithms and contains an introduction to segmentation as a field as well as how to accomplish this in ITK. The methods considered were taken from this chapter and experimented with, starting with a region growing method.

Region Growing Method
This attempt was based on the `itk::NeighborhoodConnectedImageFilter` class, which is a region growing method. Like the thresholding method it has two thresholds, but also requires a seed point from where the region will grow. This was found manually by looking at the different images. Depending on the input image, several seed points were needed.

Although fairly simple, an automatic adaption, or movement, of the seed points is not easy. The seed points are supposed to be put into the ventricles, a dark part of the image. If placed outside the ventricles, in a light part of the image, the seed point is often outside the thresholds itself and nothing happens. It is easy to deduce that something went wrong, but where to move the seed point for a better result is hard. If it was known where the ventricles were it would be simple to move them here, but as the task is to find the ventricles this is not known.

Registration
The process of registration was explained earlier in this chapter. In this setting it was suggested that one image A was to be registered with another image B, where the solution to B was available. As the images now are aligned the solution to B could be copied into A and used directly. Because of limited time available it was decided to only try rigid registration. That is, the incoming image was only rotated and moved and not stretched or warped in any way.

(The time required to get sufficient knowledge of non-rigid registration was the limit,since the running time of the program has never been an issue.)

The results were not acceptable even after manual adjustments had been tried. Thus a case base with proper solutions could not be built and the method was abandoned. A more advanced version was tried.

Registration and Level Set Methods
The solution from B copied into A did not fit perfectly and could not be expected to due to the rigid process. But the solution can serve as an initial guess to be modified, just what Repair tries to do. A family of segmentation methods called level set functions are appropriate for this task. Some of these functions can take an initial guess at a segmentation and try to evolve it based on the contents of the image and thus seemed like a natural choice to use. One problem with this approach was that the level set methods required its own set of parameters, more than six, making the task of optimizing them very hard.

Like the results from the pure registration the scores were not satisfactory. This is most probably caused by inexperience and wrong parameters. In addition to the parameters of the registration the method topped 10 parameters, too high a number. As a result no usable sets of parameters for each case were found and the method was abandoned.

For the record, the level set methods tried from ITK were:

- `itk::ThresholdSegmentationLevelSetImageFilter`
- `itk::CannySegmentationLevelSetImageFilter`
- `itk::LaplacianSegmentationLevelSetImageFilter`

For a more thorough introduction the reader may consult chapter 9.3 of the ITK Software Guide [16].

4 Case Library

The next step after finding the segmentation method is to put it to use and build the case base, or case library. This means finding a solution to the available problems and put them in a form that allows them to be used later.

4.1 The Subvolume

The subvolumes were easily extracted from the available solutions. A quick script iterated through the file and checked the indices of each ventricle voxel, noting the minimum and maximum along the x, y and z axes. These values are stored in each solution.

4.2 The Thresholds

After the subvolumes were found what was needed was the two thresholds b_{bg} and b_v. Because the segmentation takes a short time it was decided to try all possible combinations of b_{bg} and b_v. The best thresholds for each case can be seen in table 3.

Dataset	First Threshold	Second Threshold	Full Score	Score
1	100	116	0.59	0.71
2	80	94	0.50	0.63
4	64	72	0.60	0.67
5	0	54	0.71	0.74
6	0	54	0.69	0.75
7	125	133	0.58	0.71
8	106	122	0.46	0.58
11	106	122	0.67	0.76
12	112	118	0.65	0.76
13	113	130	0.41	0.52
15	0	52	0.68	0.74
16	0	52	0.62	0.65
17	23	87	0.47	0.63
100	119	124	0.53	0.67
110	119	135	0.44	0.50
111	82	112	0.47	0.59
112	94	124	0.53	0.68
191	125	132	0.47	0.67
202	42	91	0.65	0.73
205	78	102	0.66	0.74

Table 3: The thresholds and scores for each dataset

There seems to be no clear pattern. The first threshold varies from 0 to 125, the second from 52 to 135. For dataset 12 the difference between them is six, in 17 it is 64.

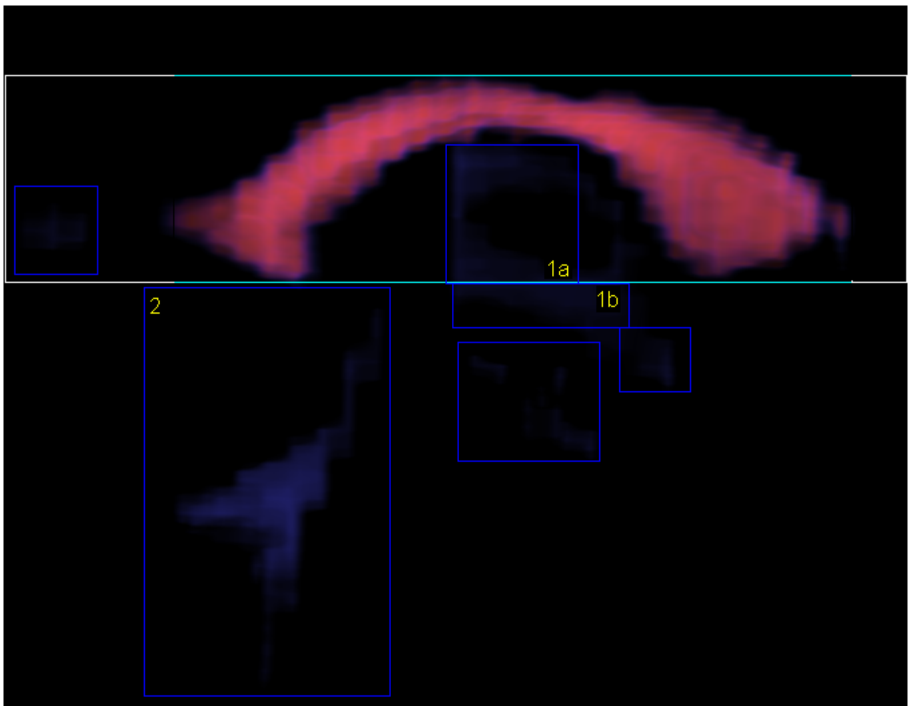


Figure 14: The white square is the parts used to compute “score” while the rest is used for “full score”. The blue squares are just to highlight regions that might be hard to see.

Datasets 7, 15 and 16 have very low scores and this can to some extent be explained. Remember that only the biggest region after the second threshold is considered ventricles and is marked as such. In these instances there are two halves that will not fuse into a single region before the last threshold is a bit too high. At this point it will also have some leaking and mark neighboring tissue as ventricle. By having a lower second threshold and keeping the two largest regions the score will be higher. This was not done in order to keep the method simple and thus the adaption will be easier.

4.3 Two Scores?

Table 3 shows two scores called “Full Score and “Score”. The difference can be seen in figure 14. The thresholding method has its shortcomings and some sacrifices were made to optimize the score for the biggest region.

In its current state the segmentation method with two iterations of thresholding will only detect the largest segment. This will of course affect the resulting score, as shown in the table. An attempt was made to extend the method so it could detect the two largest missing regions as well. For the volume in figure 14 one is marked in square 2 and the last one in both 1a and 1b. The remaining blue squares outline other, smaller parts of the ventricles that may be hard to see in the image/printout. This quickly became a trade off between complexity and score.

In the second threshold more regions than the largest can be kept. There did not seem to be a clear pattern as to which (sorted by size) these were. In one image it was 1,3,9 and in another 1,5,7. It could also be accomplished by a third or fourth thresholding with their own thresholds. This would complicate Repair even more, and that was not wanted. It was decided to keep the segmentation method at its current complexity and accept the loss of these regions. The focus will be on the detection and automatically repairing of segmentation results using *a priori* knowledge from a CBR environment, and not the development of a segmentation method that will be better than the ones already developed.

The score marked “Score” is the result of comparing only the computed segment with the parts of the ventricle in the solution that are within the subvolume defined by the white square in figure 14. The resulting score is higher, but on a global scale not more correct. If the part sought covers 75% of the ventricles a perfect segmentation would give a score of 0.75 when comparing the result with all of the ventricles. By only comparing with the solution in the subvolume a score of 1.0 can be achieved. Thus more of the scale from 0-1 is used with this method. Due to individual variations the parts shown in the blue squares vary in size from person to person. In the datasets where the Full Score and Score are fairly equal these parts are small and does not affect the score as much.

5 Repair

The following chapter will present the implementation of the Revise/Repair cycle. Although this cycle happens after matching chronologically it would be natural to compare the matching results both in terms of the initial score and the score after repairing. Thus the Repair must be presented before the matching results to put it into context.

5.1 Metadata

Before the results from Repair are presented it is in order with a presentation of the information in each case that makes Repair possible. Some information from the two regions to be found are present, size and mass center. The mass center (centroid) is represented as a 3D coordinate and "the ventricle coordinates" will refer to this. Another descriptor for the ventricles tries to say something about its shape, as they normally look the same. Three dimensional moments [10] were chosen because they are invariant to translation and rotation. Under registration the moving image may be rotated for a better fit so it is important that the descriptor can handle this. The selected moment is computed as follows:

$$J_3 = \mu_{200}\mu_{020}\mu_{002} + 2\mu_{110}\mu_{101}\mu_{011} - \mu_{002}\mu_{110}^2 - \mu_{020}\mu_{101}^2 - \mu_{200}\mu_{011}^2 \tag{7}$$

where

$$\mu_{pqr} = \sum^{X}\sum^{Y}\sum^{Z}(x-\bar{x})^p(y-\bar{y})^q(z-\bar{z})^r f(x,y,z).$$

where X,Y and Z refer to the three axis in the image, $\bar{x}$, $\bar{y}$ and $\bar{z}$ are the centroid of the object and $f(x,y,z)$ is the image intensity at x,y,z.

5.2 Detecting Failures

The above mentioned metadata is used by Revise to verify that a region is somewhat reasonable. Based on this Revise may determine the following errors:

- Region is too large
- Region is too small
- Region's mass center is not reasonable
- Region has funny shape

A region is considered fine if it is between a factor of either 0.75 and 1.5 or 0.5 and 2.0 from the size given by the used case (when each factor is used will be explained shortly). This is a trade off between wanting to give the system a small as possible range to evaluate sizes in, and having to accept differences in size between cases. Unfortunately images with big ventricles do not always register best with other images with big ventricles. This can be seen in appendix F where the ventricle size from the incoming case are compared to its best matches.

Furthermore, a centroid is reasonable if $|\phi_{new} - \phi_{old}| < 5$ for all $\phi \in x, y, z$. If it is the centroid it is said "to be found". Shape is determined by the value of J_3 and is also approved if it less than double the predicted value. No test for if this is too small has been devised as it was not necessary (at least with the test cases available). The limits used were decided after some trial and error and seem to work fine, but no thorough investigation has been conducted.

5.3 First Attempt

The first attempt at an adaption strategy was based on the average image intensity. Cases with high thresholds tended to have a high average intensity in their images, and vice versa. Thus it seemed reasonable that the thresholds could be adjusted with the difference in their average value. It was simple, quick and gave bad results. They can be seen in appendix E.

The next attempt tried to analyze and classify the final output into a set of outcomes, such as "found ventricles, but also much more". Each such outcome would have an amendment strategy that modified one or both of the thresholds. This did work better than the last attempt in some cases but also wreaked havoc in others. The main culprit found was that more than one chain of events could lead to a similar outcome. Therefore more than one strategy was needed in some outcomes and there was really no way to choose from them. Should both thresholds be adjusted, as was needed in case X, or only the last one as in case Y? To make this work the problem had to be broken down into two subproblems.

5.4 Second Attempt

In the second attempt it was decided to divide the original problem into two smaller subproblems. Instead of judging the final output and arrive at a sensible amendment strategy for both thresholds, each threshold will now be approved independently. The first step locates the unwanted region and removes it before letting step two find the ventricles, as illustrated with the following pseudo-code:

```
do{
        Attempt = Threshold(Image, Case)
        Verdict = ReviseFirst(Attempt, Case)
} while( Verdict != ''OK'')

        SecondImg = Threshold(Image, Case)

do{
        Attempt2 = Threshold(SecondImg, Case)
        Verdict = ReviseSecond(Attempt2, Case)
} while( Verdict != ''OK'')
```

ReviseFirst and ReviseSecond is responsible for modifying the thresholds in the case on each iteration.

5.4.1 ReviseFirst

ReviseFirst takes the thresholded image along with the case under consideration and returns its verdict on the image. On the first iteration the case is classified with respect to four common situations and is placed in a state designed to improve this particular situation. Which situations these are and how the classification is done can be seen in figure 15. From the starting state there are four possibilities, "Increasing at chance", "Avoiding Ventricles", "Increasing Region" and "Decreasing Region". "Increasing at chance" is the least tangible of the four and happens when the first threshold does not produce any region at all, or if it is very small and at a location that does not make sense. For the first situation and most common for the second the correct thing is to increase the threshold. "Avoiding Ventricles" may sound like the wrong thing to do, after all it is these structures we want to find. But that is the task of the second step, after this step has found and removed the unwanted region. Because of this, when a region found is believed to cover parts of them steps must be taken to leave them alone. The third, "Increasing Region", happens when the correct region is found, but is deemed to be too small. Its opposite, "Decreasing Region", has found the correct region and is trying to decrease it. Four of the cases do not remove a first region and can thus not have any information on its center or size. When these are used ReviseFirst finishes immediately.

Actually, when the region is deemed too big or small based on size, the size is compared to the size of the ventricles from the best case. This may seem strange, but it does work better. A short analysis suggested that the first region's size varies too much between cases to be useful. Also, in ReviseFirst the factors 0.75 and 1.5 are used to define the acceptable range.

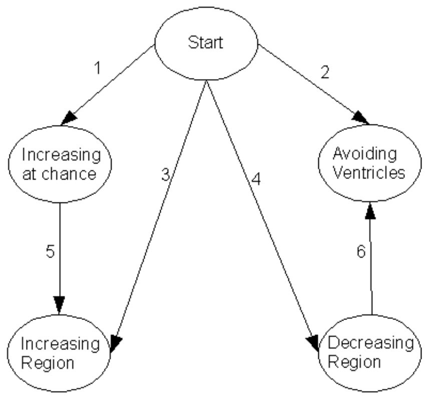
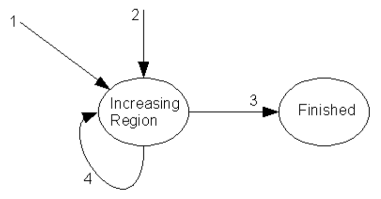


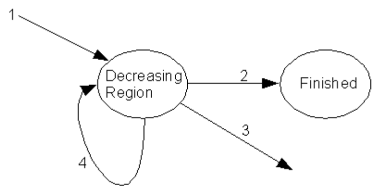
Figure 15: The state machine representing the Repair-cycle for the first threshold. 1: No useful coordinates found and a small region OR no region found. 2: Coordinates for ventricles found. 3: Found correct coordinates, but small region. 4: Too large region found. 5: Found correct coordinates. 6: Coordinates for ventricles found.

"Increasing Region", see figure 16(a), is responsible for increasing the size of a region if it is deemed to be in the correct place. To increase the size the threshold must be increased, allowing more voxels to join this region. Hence, for each iteration in this state the first threshold is increased by one. It is possible to come to this state at once if the centroid of the region is reasonable, or from "Increasing at chance". It will stay in this state until one of three things occur: 1. Region now became too big. 2. The coordinates now indicate ventricles. 3: Coordinates do no longer make sense, and indicate that the region have expanded in a strange fashion and has pulled the centroid with it. In all cases ReviseFirst uses the threshold from the last iteration (it passed these tests) and quits.

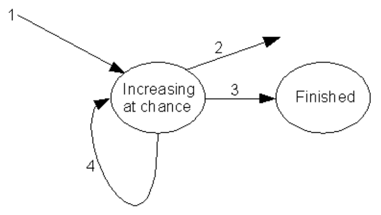
Its inverse, "Decreasing Region" in figure 16(b), can only be reached from the starting state. The test triggers on size alone and keeps on decreasing the threshold till the centroid makes sense. If the centroid moves to indicate that ventricles are found the state is changed to "Avoid Ventricles", specialized for this situation. Should the centroid now indicate that the correct region is found a second test on size is conducted. If acceptable it quits, otherwise it continues.



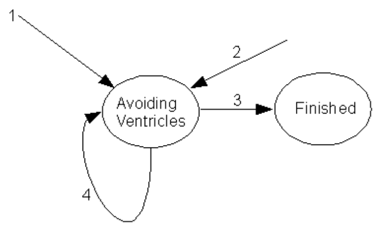
(a) 1: From Start. 2: From "Increasing at Chance". 3: Found ventricle coordinates OR Lost correct coordinates OR Region became too big. 4: Correct coordinates and acceptable size



(b) 1: From Start. 2: Correct coordinates and acceptable size 3: Found ventricle coordinates, go to "Avoid Ventricles". 4: Correct coordinates and too large region.



(c) 1: From Start. 2: go to "Increasing Region". 3: Found ventricle coordinates. 4: Default.



(d) 1: From Start. 2: From "Reducing Region". 3: Found correct coordinates. 4: Default.

Figure 16: A more detailed look on each state from Figure 15

Sometimes the centroid of the first region does not make any sense, or a region was simply not detected. "Increasing at chance", figure 16(c), was introduced to cope with these situations. If no region is found it makes perfect sense to increase the threshold to capture any voxels. Why it is correct in the first case might be hard to see. This is the last test to be performed on the region found, and will only be done if none of the other tests have triggered. Since the region is not big enough for "Decreasing Region" to be activated, the threshold is probably too low. Thus the threshold is incremented to see what happens. If the region's centroid changes to match the ventricles the threshold has become too high, and ReviseFirst quits using the last known good threshold. Should the centroid match the region to be removed the case is taken into “Increasing Region”, the state designed for this situation.

“Avoid Ventricles”, figure 16(d) is designed to make sure that the first region does not cover the ventricles. The reason is that this region will be removed and can thus not be found by ReviseSecond later. By observation the correct action in these situations is to lower the threshold, and this is done until the centroid indicates that the first region is found. As the threshold was just lowered and the correct region was found because of it, it is of no use to try and increase the threshold hoping to remove more of this region. So instead of “Increasing Region” it quits.

5.4.2 ReviseSecond

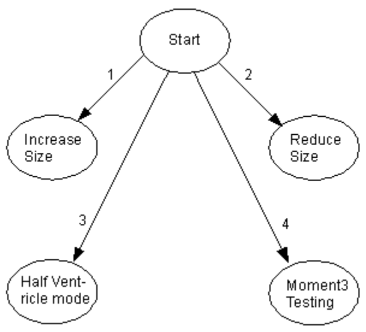


Figure 17: The choices made during the second Repair-cycle. 1: Too small size 2: Too big size 3: Y and Z coordinates fine, but not X 4: Too high J_3, but size is OK.

Unlike ReviseFirst, ReviseSecond is more of a mix between a state machine and a rule based system. The state machine part is shown in figure 17. Each iteration starts by checking size and moment and reacts to these, possibly setting the case in a new state but repeating the test next time over. Depending on the tests that trigger (too big/too small) the threshold will be updated. Too small or big here are determined by the factors 0.5 and 2.0. Only if these tests fail will the state of the case have any influence.

Reduce Size first tests to see if the centroid indicates that ventricles have been found. If this is not the case it just keeps on decreasing. If the centroid is correct it will test against J_3 to see if the shape is as expected. If it is ReviseSecond quits happily, or else continues decreasing. Increase Size does what its name implies and quits if the size or J_3 becomes too big. Moment3 Testing decreases the threshold until J_3 is below a maximum limit.

Half Ventricle mode is especially designed to fire when only one half of the ventricle is found. This is either the left or right half, seen from above, and is caused by a too low threshold. The two halves are found as two separate regions and the smallest of them is discarded by the segmentation algorithm.

Two of three coordinates of the centroid do not change from full ventricles to half ventricles, only the X element is affected. Thus if the centroid is correct along the Y and Z axes the system believes the region to be half the ventricles and enters this mode. It will stay here until the region is too large or the centroid is correct along all three axes.

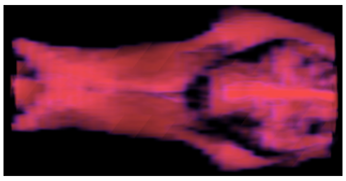
5.5 Three Examples

To show the reasoning done by the system while repairing a solution three cases will be presented and explained. One will be where the score drastically improves, one with no adaption and one where the repairing actually does damage. Many of the iterations will have an image that shows what the system is presented with and works on at that particular point in time. The 3D-nature of the images makes it hard to present them in a really good way. They are all taken from the same view and each voxel can be thought of as a small cloud. One voxel is rather small and will not be noticed against the black background, but where there are many voxels between the point of view and the background red shapes will appear. Each surface voxel is colored blue and hence blue regions indicate that you are looking along a flat side going in the depth-direction. (The program used to make the images does not allow me to alter these colors into something that may be easier to see on the black background.)

5.5.1 Repairing Case 4

Both the matching functions presented in section 3.4 agree that case 2 is the most similar case to case 4. After the registration the parameters 80 and 96 are applied giving a score of 0.0002. ReviseFirst is set to work and concludes:

```
Thresholding at 80
Computed: [30, 17, 13] To remove: [29,24,8]   Ventricle: [33,12,14]
No tests on centroid was fired. Size?
Region found is very big compared to size from case.
Suspecting white-out.
```



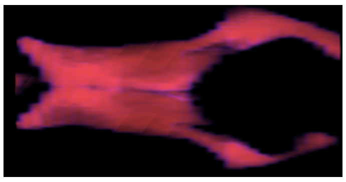
At this step the region to find is the one on the middle-right, not the ventricles on the center/left. "White-out" is an internal term indicating the first threshold is too high and the first region is too large. As this region is set to white a very large region would result in the image being filled with white, "white-out". Note that the computed centroid is close to the ventricle's centroid.

The next step is:

```
Thresholding at 79
Computed: [30, 18, 13] To remove: [29,24,8]   Ventricle: [33,12,14]
Centroid still makes no sense. Keep lowering.
```

This also happens with 78. At 77 the following happens:

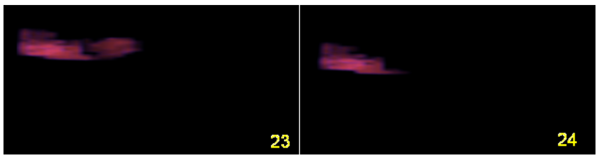
```
Thresholding at 77
Computed: [31, 13, 14] To remove: [29,24,8]   Ventricle: [33,12,14]
Ventricles found from whiteout. Enter escape-ventricles mode.
```



As the threshold has been lowered the resulting region has been modified and its centroid has now moved a little. Its new position is closer to the ventricles than before and thus the "have we found the ventricles-test" fires. This changes the state and the system works to escape the ventricles. It stays in this mode from threshold 76 to 24, where things start to happen:

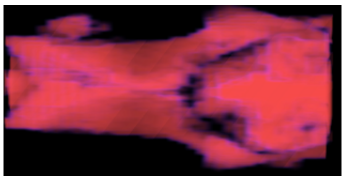
```
Thresholding at 24
Computed: [34, 8, 15] To remove: [29,24,8]   Ventricle: [33,12,14]
Still finding ventricles
---------------------------NEW First ROUND----------------------------
Thresholding at 23
Computed: [34, 6, 13] To remove: [29,24,8]   Ventricle: [33,12,14]
No tests on centroid was fired. Size?
Region found is very small compared to size from case.
Increasing (without just cause).

---------------------------NEW First ROUND----------------------------
Thresholding at 24
Computed: [34, 8, 15] To remove: [29,24,8]   Ventricle: [33,12,14]
Increased and found ventricles, backing down and leaving it at that
```



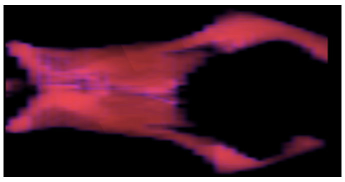
At threshold 24 most of the region has gone, but the centroid still indicates ventricle. This is not the case at threshold 23, even though it is clearly the same region, only smaller, that is found. At this point ReviseFirst quits after having reduced the first threshold from 80 to 23. Seen from the system a higher threshold finds ventricles and a lower threshold gives a very small region. At this point ReviseSecond starts with its revision. The first attempt with threshold 94 gives:

```
Thresholding at 94
Score:  0.284048  Intersection/Union: 4847/17064
Moment: 338537 vs wanted: 76234.6
Computed: [29, 18, 12]Ventricle: [33,12,14] (To remove: [29,24,8])
Big leak suspected detected by size. Lowering.
```



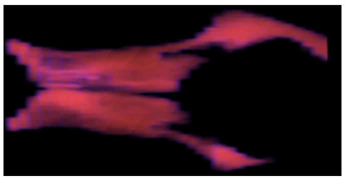
Note that the score is now printed, and the moment J_3 is presented. Already the score has gone from the initial 0 to 28% but the system believes the resulting region is too big. This result is very much the same as the initial region at threshold 80 at the start of ReviseFirst, only with a little bigger region. It keeps on decrementing the threshold and at 74 the following is found:

```
Thresholding at 74
Score:  0.630408  Intersection/Union: 4138/6564
Moment: 179519 vs wanted: 76234.6
Computed: [31, 13, 14]Ventricle: [33,12,14] (To remove: [29,24,8])
Big leak suspected detected by size. Lowering.
```



The score is now at its peak at 63%. Unfortunately the system is not happy with the size of the region. The reasoning agent is of course not allowed to see the score and does not detect a slight drop in it at the next iteration. Finally it comes to rest at threshold 63, having lost some of the score:

```
Thresholding at 63
Score:  0.588541  Intersection/Union: 3513/5969
Moment: 126492 vs wanted: 76234.6
Computed: [32, 13, 14]Ventricle: [33,12,14] (To remove: [29,24,8])
Size ok, 1008 <= 3992 <= 4032
Moment ok, 126492 <= 152469
Reduce leak-mode
Centroid approved.
```



The size is now approved, followed by an approval of the centroid and J_3. In this case the result came out a lot better than what was the case before the repairing took place. This is in spite of ReviseFirst not doing its job properly. Exactly how this affects the score will be further explained in the discussion, section 7. Figure 18 shows how the score evolves from the initial try and through the course of ReviseSecond. The jump about halfway through happens when the troublesome region is removed.

5.5.2 Repairing Case 112

The next example chosen is one where the case match and parameters are excellent to begin with. It is where the image from case 112 is presented to the system and matched (with the registration based matcher) to case 110. A case where everything works may seem somewhat of little interest, but it does introduce a problem that must be discussed. Without any form of adaption the result is:

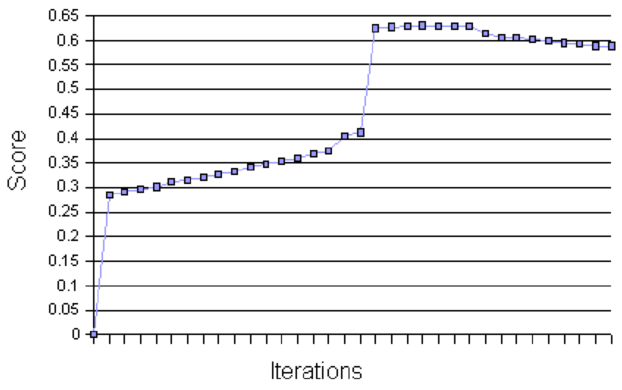
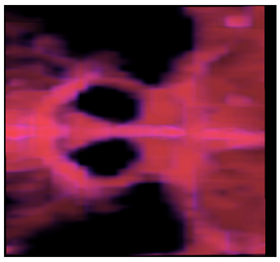


Figure 18: The score obtained with case 4 as it changes with ReviseSecond's revision and repairing.

```
Thresholding at 119
Removed: 9021
Thresholding at 135
Score:  0.658905  Intersection/Union: 3056/4638
```

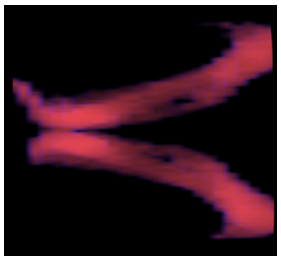
This is a very good result in itself, but the only way the system may know this is if Revise says it is. ("Removed" indicates how many voxels were set to white after the first threshold.) Therefore ReviseFirst starts and investigates.

```
Thresholding at 119
Computed: [36, 10, 9] To remove: [33,12,11]   Ventricle: [28,16,26]
Found to-be-removed at first try. Too much?.
Size looks fine, increasing.
```



This image may be hard to interpret, but the region depicted is a large region not containing the ventricles. The region found is classified as the correct one and its size is also within acceptable limits. As the goal of the first step is to maximize this region ReviseFirst tries to increase the size. This is done until the size-test fails and says that the region is now too big. It backs down and uses the last known good threshold.

```
Thresholding at 135
Score:  0.658905  Intersection/Union: 3056/4638
Moment: 319670 vs wanted: 5.46247e+006
Computed: [34, 13, 33]Ventricle: [28,16,26] (To remove: [33,12,11])
Size ok, 2851 <= 3591 <= 11404
Moment ok, 319670 <= 10924940
```

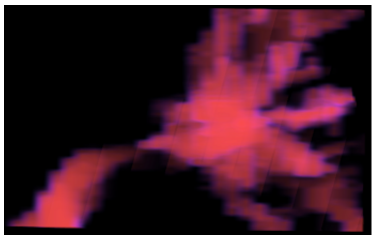


This is the result as it is accepted. Since none of the size or moment-tests flag the case is not put in any state in ReviseSecond. It runs to completion and returns. In this situation the acceptable size covers a very wide range, giving the test very little to work with. This is of course not desirable. The moment, J_3 is below its acceptable limit meaning the shape is probably fine. During the discussion, the topic regarding the large allowed size will be given emphasis.

5.5.3 Repairing Case 15

The last example shows that the system may misinterpret a situation completely and do more damage than good during Repair. It is the situation where case 15 is matched to 16, which is the best match for 15 according to the registration based matcher. Before anything is done, the following is produced:

```
Thresholding at 0
Removed: 0
Thresholding at 52
Score:  0.3141  Intersection/Union: 2221/7071
```

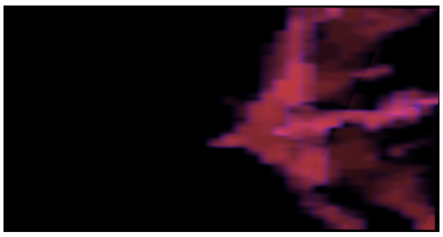


After this the system tries to verify that this is something useful and ReviseFirst starts. But case 16 has the first threshold at 0 and does not remove any region and hence has no information available to verify any first region with. ReviseFirst recognizes this as a special case and quits. ReviseSecond starts and gets the same result at its first try. To help with an interpretation of the image it can be said that it contains the same parts as the image from “Repairing case 4, thresholding at 94” on page 38. The ventricles are to the left, in this image only visible in the lower half of the picture, and there is a region to the right that is not supposed to be there. ReviseSecond concludes

```
Thresholding at 52
Score:  0.3141  Intersection/Union: 2221/7071
Moment: 3.39467e+006 vs wanted: 209927
Computed: [29, 10, 15]Ventricle: [29,14,21] (To remove: [99,99,99])
Size ok, 1584 <= 5471 <= 6338
Leak detected by Moment 3. Lowering
```

The moment J_3 is very high compared to what is wanted, which is good. It does indicate that the shape of what is found does not resemble ventricles which it does not. This is usually fixed by lowering the threshold so fewer non-ventricle voxels are included in the region. Here the case is set in the “Moment3 Testing” state and the threshold reduced. This does improve the results somewhat, as the score rises to 0.323 over the next two iterations. Then the failure introduces itself.

```
Thresholding at 49
Score:  0.00333916  Intersection/Union: 21/6289
Moment: 193675 vs wanted: 209927
Computed: [25, 8, 5]Ventricle: [29,14,21] (To remove: [99,99,99])
Size ok, 1584 <= 2489 <= 6338
Moment ok, 193675 <= 419854
```



Here the threshold is set so that the link between the wrong region and the ventricles is severed, but instead of the desired effect the wrong region is removed. The ventricles are at this point the smallest region and are removed. Interestingly, a test for the centroid on this region would clearly have revealed that ReviseSecond was wrong. The reason why this is not done is due to the fact that the case is in “Moment3 Testing”-mode where the centroid is not part of the tests. Again, it is usually not necessary, but in this case it clearly would be. Another thing to take note of is that J_3 now is OK, even though the shape clearly does not resemble any ventricles.

So this case actually introduces three important issues:

1. The final centroid does not make sense.
2. J_3 approves something it should not.
3. Case 15 registered to case 16 fails.

What caused these issues and what can be learned from them will be analyzed in the upcoming discussion in section 7. How Repair performs in other cases is shown in appendix B with the graphs of the scores during the adaption. This shows the evolution of the scores. All the final scores will be presented in the next section. The figures in the appendix are divided in three. First is the ten cases where both matchers agree on the best case. The two last are the ten remaining cases and the scores they get when using the best case from both matching functions. The rationale for dividing them in short and long repair-cycles is that when put together the short ones are hard to discern in the graph. Almost every graph points upwards. There are some examples where Repair does not work, as in the example above.

6 Retrieve

In section 3.4 two different methods of matching cases were introduced. The first was based on differences in values in statistical features extracted from each image, before the most similar image was used as the fixed image in the registration. This is referred to as the statistical-based matcher. The second proposal was to use the similarity between the two images after registration as determined by the registration process itself. This will be called the registration-based matcher. Both have been applied to the case base and the results can be seen in table 4. Unsurprisingly they return a lot of the same results. For ten of the twenty cases they return the same case as the best one. In only nine out of forty cases are one function's best not present in the other's top three. The actual distance from the input case to the best case along with distances to all other cases can be found in appendix C.

Case	**Statistical**			**Registration**		
1	17	8	191	17	7	5
2	4	17	5	17	4	16
4	2	17	5	2	5	16
5	16	17	15	16	6	4
6	16	5	15	5	16	17
7	8	191	202	1	17	8
8	7	112	191	7	191	1
11	12	110	111	12	13	111
12	11	13	111	11	13	111
13	110	111	112	111	11	110
15	6	5	16	16	6	5
16	5	6	17	5	2	4
17	7	191	202	2	1	5
100	110	111	202	110	202	112
110	13	111	100	112	100	205
111	110	13	12	110	202	13
112	13	110	111	110	205	191
191	7	8	17	205	112	8
202	191	205	7	205	111	100
205	202	191	7	202	191	112

Table 4: The three most similar cases found in the case base when using the statistical measure and the registration distance.

More interesting is how the retrieved case solves the new problem. This part is divided in two sets, the scores before and after Repair respectively, showing if the matching function found a similar case and one that is easy adaptable. Table 5 shows these values for the best match for each case together with the optimal score for each case. Both before and after Repair the statistical-based matcher does a better job, on average. The difference is not that great, however. Both matchers have one incident where the repairing does more damage than good and lowers the score. This is of course very unfortunate and should not happen. One of these cases, 15's descent from 0.31 to 0, was explained in detail in section 5.5.3.

	Statistical		Registration		
Case	Before	After	Before	After	Original best
1	0.54	0.54	0.54	0.54	0.71
2	0	0.50	0.58	0.58	0.63
4	0	0.59	0	0.59	0.67
5	0.62	0.67	0.62	0.67	0.74
6	0.71	0.71	0.72	0.72	0.75
7	0.64	0.64	0.35	0.65	0.71
8	0	0.38	0	0.38	0.58
11	0.38	0.38	0.38	0.38	0.76
12	0.07	0.68	0.07	0.68	0.76
13	0.03	0.24	0.05	0.26	0.52
15	0.44	0.74	0.31	0	0.74
16	0.59	0.59	0.59	0.59	0.65
17	0.03	0.41	0.03	0.55	0.63
100	0.17	0.45	0.17	0.50	0.67
110	0.27	0	0.19	0.32	0.50
111	0.01	0.50	0.01	0.50	0.59
112	0.63	0.62	0.66	0.66	0.68
191	0.62	0.62	0.03	0.02	0.67
202	0	0.62	0.03	0.51	0.73
205	0.49	0.61	0.49	0.61	0.74
Average	0.31	0.52	0.29	0.49	

Table 5: These are the results from the segmentation when using the statistical matching function. It shows the score both before and after the Repair-cycle has been utilized.

It is not easy to draw any conclusions based on the scores. Out of the cases where they disagree the statistical-based has the highest score in only 40% of the situations, yet it produces a higher score on averge. Both matchers have bad results in case 110. Upon inspection it has a high resulting registration score to almost every case. Could it be that the case base does not have the necessary competence to deal with this case? Overall the results are not very far from the best possible. The top results are close to the best possible, but they are all missing some points for perfection. This is unfortunate but can to some extent be explained. Repair can only detect failures and try to fix them. It has no means of telling which of two acceptable segmentation that would be best. Problems with fine tuning to get the optimal result is discussed in section 7.8.

6.1 Internal Ordering

Both matchers in general find a case that can be used to segment the new case, and with the help of Repair the score ends up decent in the majority of the cases. But is the case with the best match in fact the best case to use? To investigate this the best three matches for each matcher were used to solve the new case, and the results can be seen in appendix D. With registration the average scores before Repair are 0.29, 0.11 and 0.17 respectively, meaning that the cases ranked third are better than the ones marked second. After Repair the scores are correctly ranked with 0.49, 0.44 and 0.42. With the statistical-based matcher both rankings of average scores are correct, 0.31, 0.31 and 0.16 pre processing and 0.52, 0.46, 0.34 post. However, for each case the ordering may not be correct. Case 5 when matched on statistics has the following scores for top three: (0.62, 0.67), (0.18, 0) and (0.73, 0.73). The third case is best, the second worst and the first in between the other two. With registration the for case 191 reads: (0.03, 0.02), (0.02, 0.60) (0.58, 0.58). So it is clear that the matchers indeed do not order the retrieved cases in the correct order. But overall, they seem to work acceptably.

One advantage of using feature-based versus voxel-to-voxel similarity assessment is speed. When using registration it takes between three and ten minutes to get a score, depending on how well the images match. At the moment the case base only contains 20 cases, giving 19 comparisons with each new case. Some hacks can be introduced to save time like abort if the starting registration score is too high. Worst case runtime remains the same though, at about $19 \times 7 = 133$ minutes. This will improve as computers become faster, but it still takes a very long time. One solution can be parallel computing where each node does one comparison. Even though this system does not save any new cases, that is how CBR systems learn. Over time the case base will grow with more cases and number of comparisons done in matching increases. If the average time for one comparison is seven minutes it puts a substantial limit on the case base size.

6.2 Conclusion

Both matchers return a case which in most instances solves the incoming case adequately. There are some unfortunate exceptions meaning that neither of them are perfect. Furthermore, both have some trouble with retrieving the theoretically best case. This is evident as sometimes the second or third best case does a better job than the assumed best. The average score for both matchers are pretty much equal, with the score from the statistical matcher slightly better. This matcher is also vastly quicker and that tilts the scales in its favor. Based on this the conclusion is that Perner's [30] matching function based on extracted statistical features is the best choice for matching images, of the two considered.

Case	Score	Tested	As good or better
1	0.54	4900	1508
2	0.58	4900	223
5	0.62	7925	327
6	0.72	7925	92
7	0.64	7427	323
15	0.44	7925	512
16	0.59	Data lost	
112	0.66	10775	771
191	0.62	6125	660
205	0.49	4900	52

Table 6: How many combinations of parameters tested for each case, and the number of them giving the same or higher score than what was produced by CBR.

7 Discussion

This chapter includes thoughts on a number of topics that can be divided into two parts. Is the CBR method applicable for controlling image segmentation parameters, and can the system retrieve enough information from earlier cases to detect and fix segmentation failures? Secondly, which lessons were learned from the experiments conducted? These issues will be answered in that order and to sum up there will be a conclusion to the questions that were stated in the project's goals in the introduction.

7.1 Is CBR Applicable?

Can the usual methods from CBR be applied to segment images, i.e. Retrieve, Reuse, Revise and Retain without the focus for an iterative improvement through Repair? Based on the initial scores presented in the last chapter, before Repair was put to work, it is our opinion that yes, it can. This is based on comparisons to the best results found in the process of building the case base. These results are the optimal results for this segmentation algorithm, and serve as a benchmark. Between the two matchers ten cases have a score above 0.40, and eight cases above 0.50. Could this be incidental? Table 6 suggests the opposite. When building the case base each image was segmented with a number of different thresholds to find the optimal pair. From this it is easy to count the number of combinations that give the same or higher score than what was obtained after using the best case. Only the cases with high scores have been included. Based on this it seems highly unlikely that the high scores are incidental.

With Repair there seems to be no doubt that a new image can be segmented automatically based on information in previous cases. Repair is able to intervene when the first attempt at a segmentation is wrong and take actions to correct it. Sometimes the system produces wrong results. What this actually means is that Repair is not capable of adapting all solutions to any given new image. If all scores were perfect regardless of which case was chosen to solve the task there would be no reason to use CBR. This would imply that Revise is able to classify segmentation failures independently of any information (presently) stored in the cases since the choice of case, and hence information, does not matter. But since the choice of case does matter the case based nature of Revise is proven. It was not designed to work with all cases, it was designed to work with the best case.

With this is mind there is no doubt that the methodology of CBR is well suited for controlling the parameters of a segmentation algorithm. The rest of this chapter will be about how the system can be improved, and lessons learned that other researchers should be aware of when constructing their own systems.

7.2 Known Short-comings

This system has some short-comings. ReviseFirst does not always work. The main reason is that all cases do not react the same when the first threshold is changed. The tests devised covered most of the situations encountered while analyzing the different failures, but it is hard to detect them all and make rules for them. Fortunately the consequences in this system are not too big. The highest possible score for a case drops somewhat, but the system may still perform an acceptable segmentation.

While Repair is highly iterative in nature, there is no communication between the iterations. Sometimes the size of a region is cut in half by adjusting the threshold, but stays within the acceptable size. The centroid remains in the same position and thus Revise does not really detect any changes. But such big leaps in size indicate that something strange happened and in fact should be investigated. Another example is where, during ReviseFirst, the ventricles are found. ReviseFirst ends when the centroid has moved sufficiently far away from the centroid given in the incoming case. This happened in the first example in section 5.5.1 on page 38 between thresholds 23 and 24. At threshold 24 the ventricles are detected and at 23 they are not, according to the centroid. But by comparing the two regions it is clear that they are the same, however with a slight variation in size. Based on this the system should have concluded that the region found was still ventricle and continued to lowering the threshold. This shows that more than just the information from the best case can be used.

Any failure in the registration of two images has much more dire consequences. Figure 19 illustrates the problem. This shows the result when case 15 is registered to case 16, the best case based on the registration based matcher. It can be seen that the ventricles are outside the subvolume in every dimension. This possible outcome was mentioned in section 3.5.1 and it was stated that this system would do nothing to assure that the alignment is a good one.

Figure 19 can to help explain what happened in the Repair section where this situation was the last of the three examples to be presented. The region found initially is both the region to be removed and the ventricles, merged to one single region. When the second threshold was lowered sufficiently this region splits into two regions, where the region corresponding to the ventricles is the smallest. In effect this removes the ventricles and not the larger, wrong region. This is not

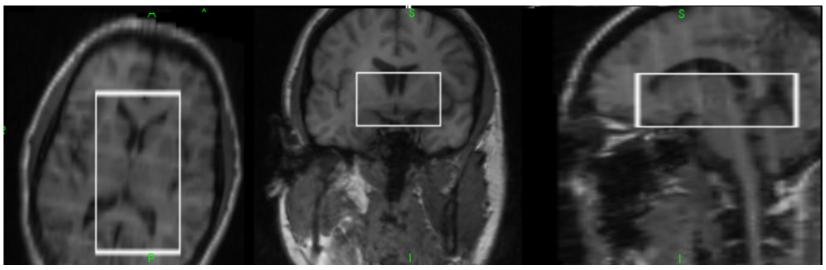


Figure 19: When registering 15 to 16 the resulting subvolume is wrong.

surprising when the middle and the rightmost images clearly show that a large portion of them are not in the subvolume which the system is working in. Even if the unwanted region was removed the resulting region would be very small and hence not be approved by ReviseSecond, based on the size. Increasing the threshold to make the ventricles bigger would not be an option either as that would bring the unwanted region back.

Since the wrong subvolume was extracted the system can not produce a satisfactory result. With this subvolume, no thresholds will be correct and the only amendment is to get another subvolume.One solution could be to modify the current subvolume to include the ventricles and expand it in all directions hoping it will have the desired effect. Or the system could try to analyze the subvolume and expand it in the correct direction only. Maybe a better solution would be to accept that this subvolume will never work, and learn from this experience.

7.3 Learning by Doing

The authors of Creek say that every case presented to the system will give it a chance to learn. The same should be attempted here. If a retrieved case is successfully adapted and solves the new case the system has done its job. However, what the system did, and equally important, why it acted the way it did, ought to be learned. In the opposite case when it failed miserably the system should learn something it can use to avoid this failure later. To continue from the scenario above where the wrong subvolume is found, what to do when the system fails will be discussed first. What to do in case of success follows thereafter.

7.3.1 Learning by failure

In Chef the program that verifies the recipe does not only say what went wrong. It also says why. The example given is a modification of a stir fry dish where chicken is replaced by beef to satisfy the user's needs. But contrary to the chicken, when the beef is stir fried it will result in much fluid in the pan. This fluid affects the vegetables in the pan and makes them soggy. This sogginess is what makes the adaption fail since the recipe specifies that the vegetables should be crisp. But the reason is that beef leaves fluid in the pan. Based on this CHEF learns that beef and (in this case) broccoli should not be cooked in the same pan at the same time. This information in cause and effect is saved and the next time the system tries to adapt a recipe and is just about to make the same mistake, this incident is remembered and appropriate actions are taken.

What would be required to accomplish something similar in this work? From the example of solving case 15 with 16 the system ended up with a result that made no sense. In this situation a simple check for centroid would reveal the error, but it may not always be that simple. But for now, let us assume such a check exists, and the result just produced failed it. A task must be defined to learn from this experience and avoid it in the future. One possibility could be to save the subvolume that did not work as a special type of case. This information are ignored during matching and are brought into play after the registration between the new image and the one from its best case. The subvolume that now will be extracted and used must undergo a screening against the "known to fail"-cases. If the proposed subvolume is more similar to a "known-to-fail"-case than some limit the system has a good reason to believe that this will only lead to failure. The correct action could be to reject the best case. Instead the second best case is promoted to best case and used as a basis for the new case. This process is repeated until the subvolume passes the test. Compared to the original example from CHEF, this would be like saving the recipe with the soggy vegetables as a failure, but not knowing why it failed. What is known is that if you want beef and broccoli, use another case since this will fail. Of course, there is no guarantee that this set up will end with a perfect result. However, one thing that indicates success is the following observation: Amongst the top three cases from each matcher one case usually gives good results when used as the best case. Based on registration distance case 191 is solved with case 205 with failure as the outcome. The second best case, 111, finishes with a score of 0.60. Based on statistical features case 5 is the most similar one to case 15. After Repair the score is 0. The second best case is number 6, giving a score of 0.74. With this in mind it seems reasonable that a final check should be conducted to make sure that the adapted region is correct, and use the second best case if not. By saving the "known-to-fail"-subvolume and reject incoming subvolumes with a similarity to this, time could be saved because the segmentation and adaption will not take place. In systems where the segmentation algorithm takes longer time to run, this might by substantial.

To verify the final result makes perfect sense and should have been introduced in this work. It would have been the last test to ensure that the result is valid. In the normal CBR-cycle this is the job of Revise. However, in this work Revise is part of Repair and as some examples show, Repair sometimes provides a wrong result if the current best case is not applicable to the current problem. To conduct a final verification after Repair would in fact be to introduce a second Revise step. But since the Revise in Repair just approved the result for some reason, the second Revise must test for some different property than (in this work) ReviseSecond used to approve the result. With this in mind a new CBR-cycle emerges, seen in figure 20. Revise has been moved out of Repair and has assumed its usual responsibility of checking whether the result in fact is a solution to the problem. To keep up with the naming tradition a new step called React has been introduced in Repair. React is a more appropriate name for this task than Revise since it is given an external stimuli (Reuse's result) and

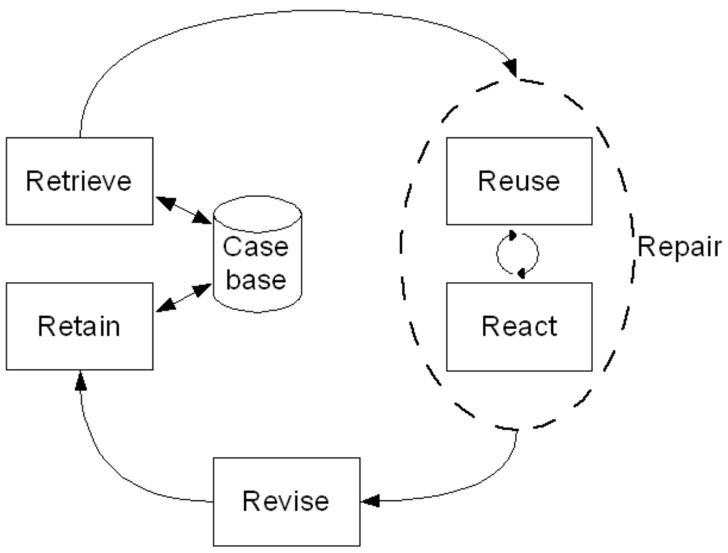


Figure 20: The new CBR-cycle suggested used for image processing CBR.

reacts to this by adjusting some parameters. However, to avoid confusion, for the remainder of this discussion "Revise" will still be referred to as the second part of Repair besides Reuse.

7.3.2 Learning by success

As with normal CBR, solved cases can give valuable information for the future. If the system segments an incoming image correctly it can save this case for later use, together with the relevant parameters and necessary metadata. When the next image is fed into the system this new case is now a part of the case base and is available as basis for new solutions. This is the usual way for systems to learn from experience.

Under the right circumstances a success/failure detection can be used to guide retrieval. For the moment all statistical features are weighted the same, as they were in Perner's [30] work. But let us say that case X is matched with case Y and the end result is a success. If one feature (Z) has a smaller difference between the two cases than another feature (W), then this can indicate that Z is more important than W for determining similarity. Based on this Z can be weighted to have a greater impact than W in future matchings. For a failure the features that are most similar do in fact not produce a good match and could be weighted down. If there is some hidden dependency between the features this approach may not work, but it is food for thought.

7.4 Harmful Cases

The example where case 16 is used to solve case 15 introduces yet another pitfall. Let us say the result passes a final test and is deemed acceptable. Let us also say that the new image, the coordinates for the subvolume and the thresholds are retained as a new case along with any metadata the system requires. (This is a hypothetical situation since the system in its current state does not retain any new cases). The best possible scenario now would be that by manual inspection the result reveals the failure and the case can be deleted by the operator. Worst case would be that, the case remains in the case base and wreaks havoc. Any case matched to it is very likely to fail, since it literally is a recipe for failure.

Such harmful cases must be kept out of the case base. An easy remedy is to let an operator approve upfront any cases to be retained. The operator would recognize a bad result without problems and take actions. It could be added to the "known to fail"-cases if implemented or simply deleted. However, this removes some of the automaticity from the system, and does not solve the problem in any fashion related to artificial intelligence. Once a harmful case has been retained an automated process would have an impossible task ahead, i.e. to remove any cases where the subvolume does not contain ventricles. To know where this is the case the process must be able to segment each subvolume and determine if the ventricles are there or not. If this process knew how to do that it should tell the rest of the system and save everybody a lot of time. However, since the system presently does not retain cases this has not been an issue, but may become one for future research in this topic.

7.5 Size of Case Base

The utility problem aside, more cases mean more solutions to pick from. As the number of cases increases the probability that one of them solves a particular problem rises as well. It also gives the matching function more responsibility, but that is not considered here. A perfect matcher is assumed. Another issue that is affected is the adaption strategy. It can go from general to more specific as it can assume that the proposed solution is not completely off to begin with. A specific example is how Revise uses the ventricle size from the old case in the adaption of the new case. An exact match in size is not expected and thus a range around the given size must be accepted. Here this accepted range is either 0.75/1.5 or 0.5/2.0 times the size from the case base. From the case with the biggest ventricles (case 11) this means that sizes between 6430 and 12861 will be deemed acceptable. This is a very big range and prohibits Revise to use the size test as effectively as it should. Maybe a fixed difference could have been used instead? What would such a constant be? For some of the cases $\pm$ 200 would suffice, for others $\pm$ 3300. With more cases it could be assumed that the best case in fact would match better in size. And for that matter, as would the moment J_3 and any other descriptor. A better match means a smaller allowed range and hence gives the reasoner a more detailed plan to work from. Generalization is traded against specialization. As an example, test were done to narrow down the acceptable range in size in ReviseSecond to the same factors as in ReviseFirst. The results were as expected. Some of the results improved while other became worse. Is this because those who improved had a better match than the others?

A somewhat general adaption strategy worked for a lot of the cases in this work. Is this good enough? Could a specialized adaption step have been applied after this general adaption? Yes, it could and it was tried with a variance-based test in ReviseSecond (section 7.8). But the whole point of case based reasoning is to select a similar looking case and work from this. If we start to fix every retrieved solution from rubbish to slightly better, then adequate, then good, then the point of using CBR is gone. The point is not to begin from scratch every time. Sometimes it has to be done because no other option is available, but we would rather not.

7.6 Extending the System

At the moment this system is geared towards segmenting MRI images of heads and to extract the ventricles. What would it take to make it able to handle images of knees, shoulders, abdomen or, for that matter, cars? This is best answered by the 4 RE-s from the CBR-cycle.

For Retrieve the number of domains covered does not really matter. It extracts the necessary statistical features and computes similarity over these. A voxel-to-voxel based similarity measure should also cope with this nicely, given that the two images have the same size. To help Retrive along cases could be divided into sets which the user can choose from when entering a new case. “I am looking for knees”, and thus only cases marked with knees are searched.

Reuse would mind. As the number of domains covered increases it is probable that the existing algorithm will not work for all. Instead of only parameters for the segmentation algorithm, the solution would have to specify its own algorithm as well, or contain a plan for how to use the existing one in a new way. With a large internal library of algorithms the case could contain a pointer to one or a set of these. It will resemble the work of [9] where the plan is a tree of tasks solved by a particular algorithm. This will not affect Reuse’s position in the CBR-cycle, only give it more power to do its work.

Revise must be altered as well. Currently it uses mainly size and centroid to detect failures and classify them. But in other tasks the goal might be a set of regions, not just one like here. Then it could be useful to consider the scene as a whole and not a collection of regions with nice centroid and size. This introduces a lot of new situations besides “too big” and “too small”. And the correct action to take regarding parameters is highly dependent on the algorithm Reuse is using at the moment. One possibility is to have failure classifications and amendment strategies sorted by topic as was suggested above in the discussion on Reuse. Something similar is described in [4] where CBR is used to control a robot. This robot is supposed to be autonomous and must handle mechanical failures on its own. This is done by a CBR system that monitors what parts are functional and finds the optimal set of rules for this configuration. Should something break down then the system will use the remaining functional parts to index a new set of rules that operates the robot without the missing component.

Retain would not need much alteration. As long as the matching is done image-to-image no indices need to be found and created. If a partition of cases into sets is used, then at the start the user will specify which set this new case belongs to, and Retain can store this along with the image. This happens only for storing new cases as they are entered and successfully solved. Retaining experiences from adaption and failures is another matter. It was discussed in section 7.3.

To conclude, Reuse and Revise must be updated to handle new domains. Revise must have a set of rules to handle the different situations. This could be solved by having a set of rules and choosing the appropriate one for each domain. The question is whether this would be just several systems intended for different domains packed together to look like one entity. Optimally the system should be able to, or at least have the option to, use and adapt knowledge from other domains in the current one. Such cross-use might not have been the intention of the system's creator but may still make sense. One possibility for such use can be by adding a new and different type of cases to the system.

7.7 Case Based Adaption

In [20] Leake describes DIAL, a CBR system that focuses on how to learn from experiences in adaption. DIAL is a tool for disaster response planning where the cases contain plans of action in various disasters such as chemical spills. When a new case is retrieved and the need for adaption is present then the system first searches among specific *adaption cases* that handles a similar adaption. If this fails the system falls back to regular rule-based adaption. If the rule-based approach produces a satisfactory adaption this is saved as a new adaption case for later use. Their example shows how the system deals with an air quality problem at a school and retrieves a similar problem from a factory. One step in that plan was to inform the workers' union. But the school-children do not have a union and the rule-based approach suggests alerting their parents instead. This transformation is saved and is put into use at a second example with a chemical spill in a school. The system detects that the children do not have a union and bases its adaption on the adaption case previously created.

A major difference in these scenarios is that with images the final adaption is not very useful. In one case it would be correct to alter the parameters by x,y and z, but in another situation the correct alteration would be by χ, Ψ and ζ. In the system presented here this problem was solved by adjusting the parameter by the smallest modification possible and test if the need for alteration was still there. So all the system can learn through an adaption is which alterations to which parameters that will give a small change in the desired direction. This is assuming there is such an adaption to the present algorithm, and that there is only one way of doing it. If there are several ways, more control information must be entered to choose the correct one. This leaves any adaption cases with the information about how to affect the outcome of a segmentation in a small direction. And this is exactly what the rule/state machine driven adaption in Repair contains now. With each new segmentation algorithm entered into the system a set of rules or cases would be added to tell the system how it should behave. Indexed by algorithm this effectively becomes the system discussed in the last section with a set of rules for each algorithm or situation. What was desired was a system that could use knowledge from other algorithms should it be of benefit. To do this with the case based approach sketched above new adaption would be required, now to adapt the adaption knowledge. If the case needed for a current algorithm does not exist, is there a case for another "edge-based, region-growing, noise-suppressing" algorithm in the case base? Could it be adapted to adjust the current algorithm for the new case?

One other point Leake[20] makes in his paper is that the learned adaption cases can affect Retrieval. When the retrieval criterion is similarity it is due to the fact that it is assumed a similar case fits the new problem best and is the easiest to adapt. With specific knowledge available about what can easily be adapted this could mean that less similar cases are easier to adapt. It is not known how to incorporate this into the similarity assessment used here.

7.8 Perfecting the Result

Repair seems to be able to detect big errors and find the correct region. Once found though, it has problems with the fine tuning and to stop at the correct threshold. This is evident in cases 4, 8, 13, 111 and 202. (See graphs of their scores in appendix B.) When ReviseSecond ends the score is lower than it was at its maximum. It is also reasonable to believe that the scores that end while increasing are not at their maximum. The likely reason is that the available information from the cases are not adequate. This topic was touched upon whilst discussing the case base size, i.e. that more cases may allow smaller ranges to check against and thus render possible higher accuracy. A second reason is that Repair only quits in a transition from bad to good, or the other way around. This means that the accepted results are always very close to not being good enough. A solution to this is not known. One could attempt to note such a transition from bad/good and continue with the present adaption (increasing, decreasing the threshold) a number of times and hope the results will improve.

Another option is to add more specialized *a priori* knowledge to the cases. It was tried at the end of ReviseSecond to assume that the region was found correctly, but that the final tweaking was somewhat insufficient. The choice fell upon the nine statistical features used in matching because of their availability. But instead of extracting them from the whole volume or even the subvolume, it was extracted from what was believed to be ventricles. That is, only the voxels determined to be ventricles were used when extracting the features. To find which ones to use the final segmentation from each case was matched with the solution from IBSR to show which voxels were correct, false negatives or false positives. If the correct voxels have a rim of false positives the final threshold was probably too high, and vice versa. After it was manually determined whether the final result was too big or too small it was compared with the statistical features gathered to look for a connection. The feature that seemed to agree best with the manually found result was the variance. Therefore a new state was introduced in ReviseSecond, "Variance Checking". It was placed as the last step, when ReviseSecond was finished Variance Checking became active. The state can be seen as a ReviseThird step. The case had no way out of this state until the computed variance was between a factor of 0.8 and 1.2 of the variance given from the best case. This helped in some cases but did the same amount of harm to others and was dropped.

Maybe each case can store the result of the segmentation. Then when the case is used to solve a new problem the two results can be compared. This, like with the other methods, depends on how good the initial match of cases are. If there can not be established a clear connection between the similarity in shape of ventricles as the similarity between cases increases, the allowed difference between the results would have to be so big that it would be useless. A different problem is how to make the comparison. How to compute similarity between 3D objects has received a lot of attention lately, mostly for use in search engines that aim to find similar objects. The main problem with using these is that the algorithms offered try to distinguish between different classes of objects and not necessarily between objects for the same class. As an example, Novotni et al [28] compute similarity by aligning 3D models and then compute their distance histograms. They conclude that the system delivers a measure on the *overall* similarity.

A final idea is to use the result of this segmentation as the starting point for another segmentation. One of the rejected segmentation algorithms (section 3.9) tried to use a level set function based on the solution of the best case. Here the correct solution from IBSR was planned to be used, but in reality two options are available. One is to use the result from the best case, as planned. The other would be to use the outcome from Repair. The level set function would easily correct any mistakes where the subvolume cuts off part of the ventricles, given that Repair has found the rest. If the parameters for this level set function should be set by another round of CBR or not is not known.

7.9 General Domain Knowledge

There are few systems under development today that try to be exclusively a CBR system. As more research has been conducted, advantages of domain specific knowledge outside the cases have been noted. By combining different research areas a synergetic effect is achieved as one area's strengths may compensate another's weaknesses. So by combining rule-based and case-based systems the best from both can be harvested. During the development of the system presented here care has been taken to avoid using other information than what is found within each case. Sometimes Repair increases one of the thresholds to a value that is outside any reasonable limits. Why it does this may vary, but until one test says "enough" it will continue. Based on the thresholds found, one could conclude that the second threshold will never be above 200. There is presently no test in Revise that test for 200 and flags an error should this value be detected. The test to see if half a ventricle has been found relies on a comparison between centroids. An observation made is that J_3 is negative in such situations. Such general knowledge is not case specific and therefore not stored in any cases. Hence it is neither the basis for the test it would have been perfect for.

Creek uses domain knowledge during Retrieve to investigate the semantic difference between two indices, not the lexical difference. This cannot be done here since the cases are matched on extracted features, and no knowledge exists to explain what differences here actually mean for the similarity between images.

7.10 Reaching the Goals

At the start of this project three questions were presented, to which this work wanted to answer. For convenience they are reproduced here:

1. How to assess similarity between two cases?
2. How to use *a priori* knowledge extracted from a case to
   (a) Help Revise judge the output from Reuse
   (b) Guide the adaption of the solution

When it comes to similarity assessment both proposals that were presented on average produced good results. The one based on feature extraction was deemed the best both in terms of speed and the results its retrieved cases produced. For comparing images in CBR it is a very good choice for a similarity function. It is however not perfect and sometimes retrieves suboptimal cases and on rare occasions cases that do more harm than good. One of the cases it deems as top three usually does a good job, and a plan to take advantage of this was presented. The size of the case base is not very large and how it affects the retrieval is not known. Without further research in this area it can only be speculation whether more cases would add confusion or more clarity.

There is no doubt that information can be stored in a case that lets Revise classify an output as good or bad. By finding some properties a good result should have, and if the properties can be represented internally so Revise can reason with them, it is easy for Revise to judge the output of a case. Based on an expected centroid, size and a statistical moment,J_3, this system can usually conclude that something has gone wrong. It is harder to say when something has gone as it should, though. Due to individual variations the values are not expected to match perfectly, but within an acceptable range. As long as it stays within this range it is hard to know if one result is better than another. Some thoughts and possibilities were offered on the subject in the preceding discussion. When it comes to guide the adaption based on information from a case alone, there are two options. One is to store it in the cases that exist in the system already. It would be in the form of "if region is too big compared to X, reduce the second threshold". This would then be duplicated into every case in order to give each case the *a priori* knowledge needed to guide adaption. If one test must be adapted it would have to be done in all cases. Compared to the present system it will accomplish the same, but in a less elegant manner. The second option is to use adaption cases. Indexed by the segmentation algorithm and problem at hand the information on how to adapt is in fact coming from a case.

In concluding remarks on adaption, a final point should be made. Until a deeper understanding of segmentation is available, adaption can not be indexed to solve a particular problem like "lower threshold by X". X can not be determined by analyzing the images alone and must therefore be found by search. This is the essence of Repair, small repeated steps until a set of criteria is met. What these steps will be is the job of the adaption knowledge.

8 Further work

The discussion introduced many areas where CBR for image segmentation could be improved. Some of these are active research areas for improving the CBR methodology itself, like adding case based adaption, incorporating general domain knowledge and letting adaption knowledge influence retrieval. Since the mix between image processing systems and CBR is fairly new it should be explored how they could work together in a more “standard” CBR cycle first. Based on this further work, either with this system or others, should focus on:

- Verifying the final result.
- Rejection of wrongly retrieved cases.
- Fine tuning.
- Retention of new cases.

With the new CBR-cycle from figure 20 a new Revise-step has been introduced to examine the final result. It must be based on other properties than what is used in React, and this introduces some problems. Say there is a test Θ in Revise that is not used in React and which is able to judge success/failure. Hence there is some information that React does not have, and which could possibly allow it to do a better job. The same can be said for all tests proposed for Revise. So unless Revise is done manually by an expert, React must renounce some tests for the greater good of the system. Research should be conducted in what React must sacrifice in order to have a Revise step. The second task of Revise must not be forgotten either. It is unlikely that Repair can give a perfect result independent of image and the assumed best case. When a failure occurs it must be used to prevent the same failure again. As argued the exact reason for failure is not likely to be found, so inferior solutions must be devised instead. Here it was proposed to save the subvolume as an example of something that did not work. For other segmentation algorithms this is not feasible, they would need their own solutions. If a system is to be general and learn both new domains and algorithms during its operation an algorithm independent solution must be found.

Because it is not realistic to develop a perfect similarity function, controls that deal with cases retrieved out of order are needed. This is in image processing hard to do since it is not known what features you should look for in advance to detect this. How Retain solves knowledge of failure, as mentioned above, plays a major role in this. To compare with “known-to-fail” subvolumes was introduced as an idea, but it is only applicable with this segmentation algorithm. For others it is not known, and more research is needed to improve this topic.

React only checks to see if the result is wrong and thus can not say how correct it is. It is either correct or belongs to a failure category. To find a measure for correctness is what every researcher in segmentation wants, and thus is not readily available. Maybe a larger case base means that better solutions will be found and that fine tuning is not needed. This raises another issue, namely how case base size affects results is not known. Overall, the impact of case base size should be investigated. Optimization of parameters is not a new problem, nor is it restrained to CBR. It is a research field of its own and many attempts have been made, simulated annealing [19] being one of them. A variation of this might be applicable where React adjusts the acceptable range a property can deviate from a given value depending on the number of iterations Repair has done. In the beginning the property is allowed to vary by a (reasonably) large amount. As more adaption is done, the parameters are assumed to match better and the allowed deviation is reduced. Another possibility is to hand over the result over to a very specialized segmentation algorithm for further processing. This may or may not be part of another CBR cycle.

This work has not done any work in retention. Retain saves any information gathered by Revise regardless of success or failure. Failure was discussed above for Revise, so only successful cases are mentioned here. Research in this CBR-step is much more than just saving a new case. Dumping the new image together with metadata and the correct parameters is not hard to do. What is more difficult is to avoid the utility problem, storing too many cases and degrading the system. More intelligent control for storing is needed, and the theory behind it. When are two images so similar that only one needs to be in the case base? Or should the cases somehow be merged? The need for work in retention is obvious, but this work does not provide any answers to it. Answering such questions must be done if a future system is to operate over a long period of time, learning from each problem solved.

8.1 The Bigger Picture

How does this system fit into the image processing pipeline from the introduction, reproduced below for convenience? Grimnes built ImageCreek to

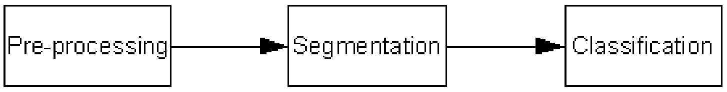


Figure 21: The steps in machine vision, reproduced from section 1.

interpret segmented images and his work is thus placed in the classification step. Each segment is classified independently and later their internal positioning and properties are viewed together. If this holistic view does not match any domain knowledge the segmentation is deemed to be wrong and a new one is asked for. This is the last resort of the program and it is not specified how the new segmentation is done differently from the previous one. To summarize, it was a classification system that could go back to segmentation in the pipeline. This work represents a tighter coupling between the two last image processing steps. Neither the segmentation algorithm nor the classification step has been very advanced, instead the focus has been on the cooperation. How would it be to use this system and ImageCreek together? Put in sequence the synergy is lost. Reuse produces a first attempt and expects React to inform it of failures and guide it further. ImageCreek expects a near perfect segmentation and asks for a new only as a last resort. Two important assumptions, one in each system, are not met. Instead ImageCreek could be modified to work in an iterative way. It is perfect for determining whether the segmentation was correct or not, but does at this point in time not guide the resegmentation. If put in React together with the proper adaption knowledge the result given to Revise would be a finished segmentation together witn an interpretation of the image. What is left is a system that merges segmentation and adaption to help one guide the other.

This merging must be given special attention. What is needed is an appropriate control structure that can modify the segmentation based on the classification's output. This is, in our opinion, the single most important step for the success for such an iterative approach, whether used in a CBR-system or not.

A IBSR data sets and names

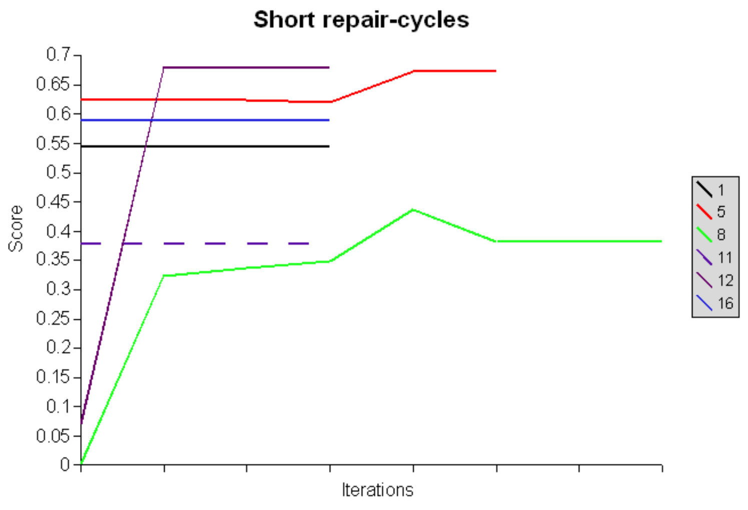
Unless otherwise stated a referense to “data set #” refers to an IBSR data set. More spesifically the IBSR data set is downloaded from `http://www.cma.mgh.harvard.edu/ibsr/` (registration required) and is marked “20 Normal Subjects: T1-weighted MR Image data with gray/white/other expert segmentations (3.1mm slice thickness)” The mapping from “data set #“ to an actual data set is:

1	1_24.img
2	2_4.img
4	4_8.img
5	5_8.img
6	6_10.img
7	7_8.img
8	8_4.img
11	11_3.img
12	12_3.img
13	13_3.img
15	15_3.img
16	16_3.img
17	17_3.img
100	110_23.img
110	110_3.img
111	111_2.img
112	112_2.img
191	191_3.img
202	202_3.img
205	205_3.img

To view these data sets one can use imageJ or VolView (amongst others) and open the accompanying .hdr-file for each .img-file.

B Full Scores

Below are figures that show how the scores for each data set evolve through the course of Repair. The reason for dividing the short and long cycles into two figures is to show the short ones clearer.



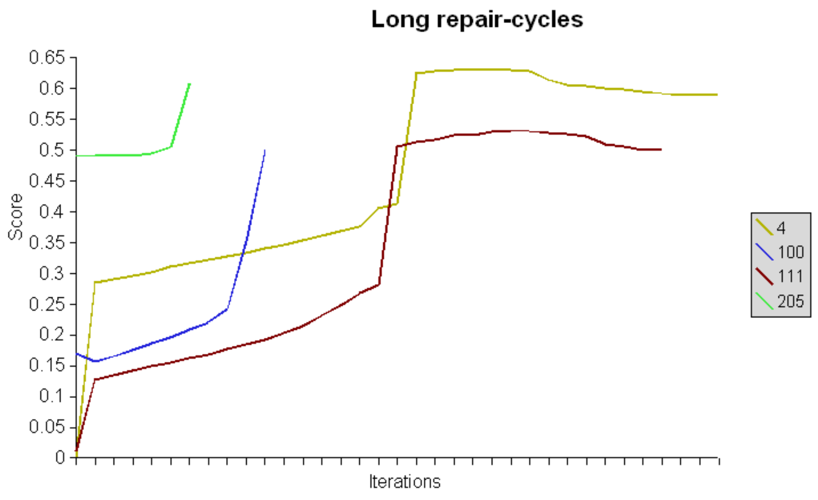
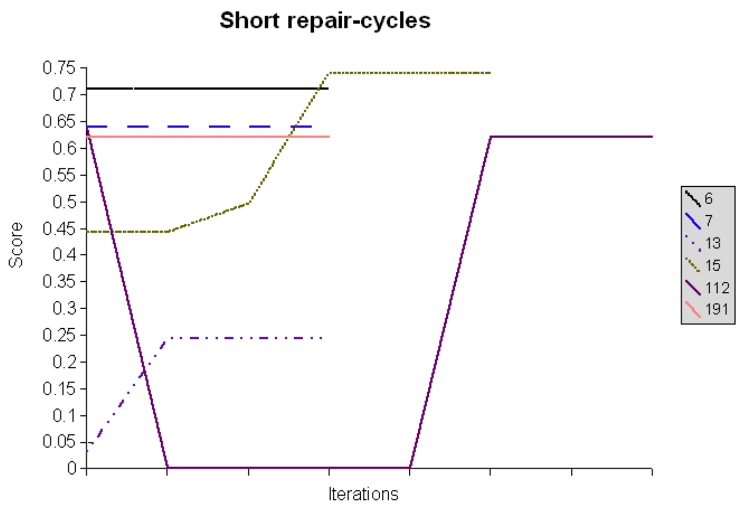


Figure 22: The scores during Repair for the best cases where both matching functions agree.



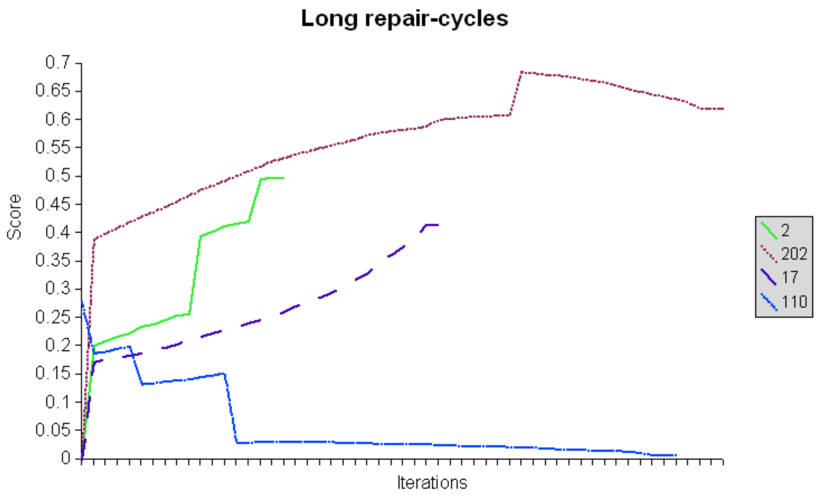
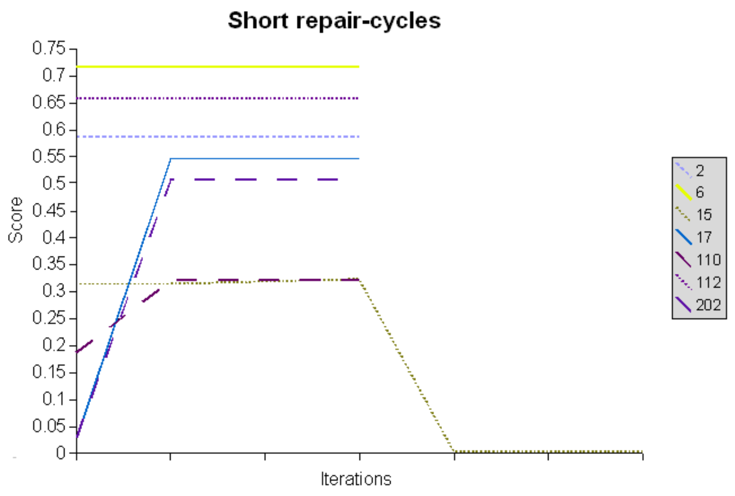


Figure 23: The scores during Repair for the best cases as determined by the statistical-based matching function.



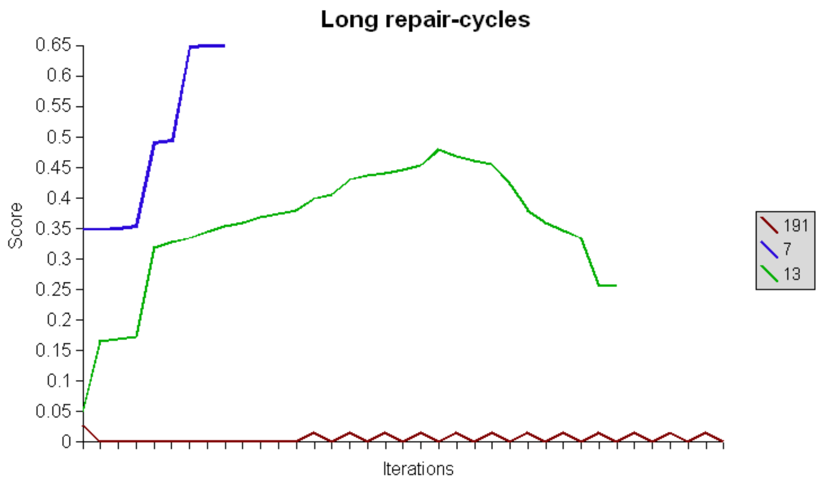


Figure 24: The scores during Repair for the best cases as determined by the registration-based matching function.

C Matching Scores

The following tables show the similarity between all cases as determined by both matchers. It can be seen that some cases have a very high score towards almost every other cases, and some cases have low scores on average.

	1	**2**	**4**	**5**	**6**	**7**	**8**	**11**	**12**	**13**	**15**	**16**	**17**	**100**	**110**	**111**	**112**	**191**	**202**	**205**
1	×	798	932	**723**	892	**716**	1052	2258	2315	2191	1693	982	**568**	1799	1791	2043	1527	1242	1646	1302
2	830	×	**545**	697	951	1036	1427	2564	2579	2605	1605	**611**	**537**	2116	2171	2226	1915	1304	1623	1319
4	957	**539**	×	**630**	932	1339	1561	2796	2926	2893	1554	**644**	820	2287	2434	2408	2177	1624	1938	1590
5	753	710	**641**	×	**547**	1114	1426	2663	2855	2670	1220	**492**	788	2016	2198	2324	1980	1592	1854	1531
6	923	947	935	**547**	×	991	1294	2366	2472	2445	1159	**725**	**920**	1791	1976	2006	1684	1264	1665	1328
7	**720**	1007	1318	1085	972	×	**901**	2185	2198	2068	1928	1438	**821**	1737	1657	1971	1435	1151	1723	1271
8	**1050**	1412	1527	1384	1268	**888**	×	2014	2082	1897	1465	1747	1165	1594	1485	1707	1193	**931**	1491	1275
11	2205	2563	2724	2596	2312	2184	2010	×	**946**	**1059**	2559	3138	2242	1273	1196	**1082**	1429	1749	1371	1549
12	2283	2812	3020	2985	2431	2193	2001	**927**	×	**1148**	2519	3142	2217	1688	1542	**1276**	1652	1932	1614	1784
13	2152	2779	2869	2681	2435	1982	1830	**1036**	1118	×	2420	3084	2227	1129	**1043**	**976**	1233	1658	1179	1395
15	1736	1927	1668	**1193**	**1152**	1967	1504	2524	2544	2468	×	**1145**	1645	1926	2140	2001	1911	1649	1694	1803
16	1028	**622**	**654**	**497**	736	1481	1772	2994	3127	3098	1225	×	847	2218	2522	2493	2330	1759	1982	1693
17	**590**	**528**	815	**773**	918	843	1199	2254	2288	2296	1673	840	×	1975	1763	2051	1594	1184	1633	1188
100	1826	2058	2329	2012	1762	1710	1668	1228	1659	1124	1892	2256	1915	×	**861**	1108	**1066**	1308	**955**	1028
110	1797	2133	2567	2292	1938	1676	1518	1193	1565	1081	2077	2594	1734	**876**	×	**931**	722	1138	990	**927**
111	2046	2210	2361	2389	2038	1998	1738	1069	1288	**997**	1962	2638	2012	1121	**932**	×	1034	1475	**939**	1121
112	1525	1875	2183	1988	1655	1413	1180	1395	1664	1260	1872	2533	1559	1066	**722**	1038	×	**845**	1002	**797**
191	1243	1280	1591	1569	1240	1150	**938**	1736	1966	1703	1678	1660	1154	1320	1148	1491	**848**	×	1126	**733**
202	1645	1612	1902	1871	1628	1788	1493	1375	1626	1192	1679	1934	1559	**975**	993	**947**	1000	1123	×	**734**
205	1341	1331	1589	1523	1316	1281	1313	1579	1821	1444	1794	1715	1188	1064	941	1154	**810**	**759**	**757**	×

Table 7: The distanse after registering the images with each other. Read as the final distanse when registering 1 with 2 is 798. The three best for each image are in bold.

	1	2	4	5	6	7	8	11	12	13	15	16	17	100	110	111	112	191	202	205
1	×	34.6	39.5	37.9	33.5	30.7	**29.1**	61.5	56.1	52.5	48.1	38.8	**27.8**	43.4	51.2	54.0	43.1	**29.2**	42.0	45.2
2	32.1	×	**15.0**	**22.3**	31.2	36.2	42.3	59.4	65.8	60.8	45.7	24.3	**21.2**	49.6	51.8	56.7	53.1	34.0	38.4	43.2
4	32.8	**12.7**	×	**28.1**	38.5	41.0	49.5	69.3	73.4	70.2	51.6	30.7	**25.9**	59.1	61.4	66.2	62.2	34.9	44.3	47.2
5	30.3	24.6	32.0	×	19.7	29.4	28.0	54.1	49.9	53.5	**24.5**	**14.7**	**23.7**	50.7	49.3	45.5	46.2	40.1	43.4	51.0
6	30.7	33.0	41.5	**22.5**	×	33.0	30.1	47.9	44.7	46.0	**23.9**	**21.3**	24.7	39.8	41.2	43.1	45.0	40.2	43.5	51.2
7	27.6	35.5	43.1	33.0	31.2	×	**13.4**	37.7	32.9	31.8	39.3	29.9	18.9	30.8	28.3	28.7	24.1	**13.8**	**17.2**	25.8
8	24.1	43.0	53.4	34.7	29.4	**14.7**	×	39.0	32.0	28.9	38.5	34.0	28.8	28.1	28.6	30.6	**22.4**	**23.1**	27.1	36.6
11	55.8	62.2	73.5	60.0	50.4	42.8	40.1	×	**15.9**	22.7	56.5	60.6	49.6	27.7	**19.4**	**22.3**	34.0	42.7	35.3	46.4
12	51.8	66.0	75.9	55.2	44.9	36.8	32.9	**14.8**	×	**18.9**	46.4	52.6	50.8	34.5	21.9	**19.5**	30.7	43.1	42.6	54.5
13	42.8	56.3	65.3	55.1	43.5	32.0	27.1	21.8	19.9	×	56.4	51.1	44.1	21.0	**10.8**	**15.4**	**17.4**	34.8	31.0	41.1
15	41.6	43.4	47.9	**23.8**	**21.3**	37.8	39.0	57.5	50.0	59.9	×	**32.2**	34.8	57.7	56.1	49.5	57.9	50.6	51.9	57.9
16	34.8	23.6	30.1	**16.5**	**19.5**	30.0	32.1	57.0	52.7	51.6	31.4	×	**24.7**	48.6	47.4	47.9	48.2	39.4	42.3	50.0
17	23.3	22.5	31.2	26.2	23.6	**20.1**	25.0	44.2	47.7	46.5	37.1	24.3	×	36.7	38.1	42.9	38.8	**22.0**	**22.4**	28.1
100	36.8	51.5	62.9	55.2	43.4	33.9	29.3	25.7	32.5	24.2	56.7	52.4	40.6	×	**18.2**	**23.0**	24.7	33.5	**23.8**	34.3
110	43.2	49.5	58.9	50.6	40.8	29.9	26.0	16.8	20.2	**13.2**	51.5	48.1	36.9	**15.4**	×	**14.5**	18.8	29.1	24.2	35.5
111	45.4	54.3	62.8	47.0	40.7	28.9	28.2	19.4	**17.1**	**13.8**	45.0	46.4	41.3	20.7	**10.1**	×	19.4	32.7	28.5	37.3
112	33.3	48.3	56.9	47.5	41.9	23.0	20.2	31.3	30.7	**14.8**	54.1	46.7	36.1	19.9	**16.1**	**17.5**	×	27.3	22.8	29.4
191	24.2	31.1	36.4	39.9	37.9	**17.5**	**20.2**	39.5	41.8	40.5	49.5	38.6	**20.7**	32.0	32.2	37.2	32.7	×	21.1	22.0
202	35.0	37.1	45.1	41.8	41.6	**18.4**	23.2	32.1	39.0	34.0	50.7	41.4	21.9	23.8	25.1	29.8	25.8	**17.6**	×	**18.1**
205	35.7	39.8	42.1	47.7	48.6	**24.9**	32.0	42.9	51.1	39.3	54.9	48.2	26.7	30.4	32.4	34.2	26.7	**18.3**	**11.2**	×

Table 8: The distanse after matching the images with each other with the statistical-based matching function. Read as the final distanse when mathcing 1 with 2 is 34.6. The three best for each image are in bold. Numbers shown in percent.

D Top 3 Matching Results

To see if the matching functions rank the cases in a way that corresponds to their score, the three best matches for each case where put through the system. This was done for both the registration-based approach:

Registration-based matching						
	First		Second		Third	
1	0.54	0.54	0	0.54	0	0.33
2	0.58	0.58	0	0.49	0	0
4	0	0.59	0	0.54	0.50	0.50
5	0.62	0.67	0.63	0.63	0	0.66
6	0.72	0.72	0.71	0.71	0.21	0.49
7	0.35	0.65	0	0.61	0.64	0.64
8	0	0.38	0	0.54	0.36	0.41
11	0.38	0.38	0.02	0.17	0	0.34
12	0.07	0.68	0	0.67	0	0.70
13	0.05	0.26	0	0.46	0.03	0.24
15	0.31	0	0.44	0.74	0.20	0
16	0.59	0.59	0	0.36	0	0.61
17	0.03	0.55	0	0	0	0.27
100	0.17	0.50	0.03	0.03	0.26	0.63
110	0.19	0.32	0.32	0.28	0	0
111	0.01	0.50	0.05	0	0.01	0.49
112	0.66	0.66	0.02	0.20	0.70	0.70
191	0.03	0.02	0.02	0.60	0.58	0.58
202	0.03	0.51	0.05	0.62	0	0.57
205	0.49	0.61	0	0.67	0	0.21
Average	0.29	0.49	0.11	0.44	0.17	0.42

and for the statistical-based:

Statistical-based matching						
	First		Second		Third	
1	0.54	0.54	0.60	0.61	0	0.57
2	0	0.50	0.59	0.59	0	0
4	0	0.59	0.43	0.16	0.54	0.54
5	0.62	0.67	0.18	0	0.73	0.73
6	0.71	0.71	0.72	0.72	0.70	0.70
7	0.64	0.64	0.70	0.70	0	0
8	0	0.38	0.27	0.39	0	0.54
11	0.38	0.38	0.01	0.31	0.04	0.34
12	0.07	0.68	0	0.67	0	0.70
13	0.03	0.24	0.05	0.26	0.36	0
15	0.44	0.74	0.20	0	0.31	0
16	0.59	0.59	0.58	0.58	0.15	0.31
17	0.03	0.41	0	0.46	0	0
100	0.17	0.45	0.56	0.56	0.03	0.03
110	0.27	0	0.03	0.36	0.32	0.28
111	0.01	0.50	0.01	0.49	0	0.50
112	0.63	0.62	0.66	0.66	0	0.42
191	0.62	0.62	0.58	0.58	0	0.37
202	0	0.62	0	0.51	0	0.68
205	0.49	0.61	0	0.67	0	0.04
Average	0.31	0.52	0.31	0.46	0.16	0.34

E Average Intensity-based Adaption

The first attempt of adaption was to use the difference in average intensity between the two images. This did not give satisfactory results.

Dataset	Adjustment	Post-Score
2	-2	0.39
4	-14	0
5	10	0.29
6	-1	0.62
7	3	0.43
8	0	0
11	3	0.37
12	5	0.03
13	8	0.04
15	-2	0.24
16	-9	0.52
17	2	0.02
100	-8	0.10
110	-9	0.02
111	-4	0
112	7	0.27
191	8	0.02
202	-4	0.02
205	2	0.39

F Difference in Size

When matching cases the size of the ventricles from the best case does not always map very good with that of the new case. This means that the system must allow two sizes to be somewhat different, but not too different. If this happens the size metadata is useless as every attempt will be approved. Note that in the table the size given for dataset 1 is the size prior to registering to the other images. Some of the ventricles may be moved outside the subvolume and the size given to Revise to reason with may be less than stated here.

Dataset	Orig. Size	Size from reg.	Size from stat.
1	3567	2255	2255
2	2016	2255	5350
4	5350	2016	2016
5	4665	3169	3169
6	5518	4665	3169
7	3604	3567	2818
8	2818	3604	3604
11	8574	7787	7787
12	7787	8574	8574
13	3122	3518	5702
15	6858	3169	5518
16	3169	4665	4665
17	2255	2016	3604
100	3317	5702	5702
110	5702	3406	3122
111	3518	5702	5702
112	3406	5702	3122
191	2360	7070	3604
202	7284	7070	2360
205	7070	7284	7284

References

[1] The insight segmentation and registration toolkit. `http://www.itk.org`.

[2] Agnar Aamodt. Explanation-driven case-based reasoning. In *EWCBR '93: Selected papers from the First European Workshop on Topics in Case-Based Reasoning*, pages 274–288, London, UK, 1994. Springer-Verlag.

[3] Agnar Aamodt and Eric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AICom - Artificial Intelligence Communications*, 7:39–59, 1994.

[4] D. Aha. The omnipresence of case-based reasoning in science and application, 1998.

[5] J.L. Arcos, R. Lopes de Mantaras, and X. Sierra. Saxex: a case-based reasoning system for generating expressive musical performances. *Proceedings of the International Computer Music Conference*, pages 329–336, 1997.

[6] C. N. de Graaf, A.S.E. Koster, K.L. Vincken, and M.A. Viergever. A methodology for the validation of image segmentation methods. *Fifth Annual IEEE Symposium on Computer-Based Medical Systems*, pages 17–24, June 1992.

[7] Ramon López de Mántaras, David McSherry, Derek Bridge, David Leake, Barry Smyth, Susan Craw, Boi Faltings, Mary Lou Maher, Michael Cox, Kenneth Forbus, Mark Keane, Agnar Aamodt, and Ian Watson. Retrieval, reuse, revision, and retention in case-based reasoning. *Knowledge Engineering Review*, 20:215–245, 2005.

[8] Lijun Ding, Thisath Kularatna, Ardeshir Goshtasby, and Martin Satter. Volumetric image registration by template matching. In *Proc. SPIE Vol. 3979, p. 1235-1246, Medical Imaging 2000: Image Processing, Kenneth M. Hanson; Ed.*, pages 1235–1246, jun 2000.

[9] Valerie Ficet-Cauchard, Christine Porquet, and Marinette Revenu. Cbr for the reuse of image processing knowledge: A recursive retrieval/ adaptation. In *ICCBR '99: Proceedings of the Third International Conference on Case-Based Reasoning and Development*, pages 438–452, London, UK, 1999. Springer-Verlag.

[10] Bob Fisher. 3d moment invariants. `http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/FISHER/MOMINV3D/inv3d.htm`, 2001.

[11] M. J. F. Grimnes. *ImageCreek: A Knowledge Level Approach to Case-Based Image Interpretation.* NTNU - PhD Thesis, 1998.

[12] M Haddad, D Moertl, and G Porenta. Scina: a case-based reasoning system for the interpretation of myocardial perfusion scintigrams. *Computers in Cardiology 10-13 Sep 1995*, pages 761–764, 1995.

[13] The Vancouver Hospital and Health Sciences Centre. http://www.physics.ubc.ca/ mirg/home/tutorial/tutorial.html.

[14] Zhimin Huo and Maryellen L. Giger. Evaluation of an automated segmentation method based on performances of an automated classification method.

Medical Imaging 2000: Image Perception and Performance, 3981(1):16–21, 2000.

[15] D.P. Huttenlocher, G.A. Klanderman, and W.A. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.

[16] L. Ibanez, W. Schroeder, L. Ng, and J. Cates. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-15-7, http://www.itk.org/ItkSoftwareGuide.pdf, second edition, 2005.

[17] Tina Kapur. Segmentation of brain tissue from magnetic resonance images. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.

[18] Abdus Salam Khan and Achim G. Hoffmann. Acquiring adaptation knowledge for cbr with mikas. In *AI '01: Proceedings of the 14th Australian Joint Conference on Artificial Intelligence*, pages 201–212, London, UK, 2001. Springer-Verlag.

[19] S. Kirkpatrick, D. Gelatt Jr, C., and P. Vecchi, M. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[20] David B. Leake. Combining rules and cases to learn case adaption. In *Proceedings of the Seventeenth Annual Conference of the Cognitive Science*, pages 84–89, 1995.

[21] David B. Leake and David C. Wilson. Remembering why to remember: Performance-guided case-base maintenance. In *EWCBR '00: Proceedings of the 5th European Workshop on Advances in Case-Based Reasoning*, pages 161–172, London, UK, 2000. Springer-Verlag.

[22] M. Lenz. Cabata - a hybrid cbr system, 1993.

[23] H. Li, R. Deklerck, B. De Cuyper, A. Hermanus, E. Nyssen, and Jan Cornelis. Object recognition in brain ct-scans: Knowledge-based fusion of data from multiple feature extractors. *IEEE Transactions on Medical Imaging*, 14(2):212 – 229, 1995.

[24] M. Lorenzo-Valdes, Gerardo I. Sanchez-Ortiz, R. Mohiaddin, and Daniel Rueckert. Atlas-based segmentation and tracking of 3d cardiac mr images using non-rigid registration. In *MICCAI '02: Proceedings of the 5th International Conference on Medical Image Computing and Computer-Assisted Intervention-Part I*, pages 642–650. Springer-Verlag, 2002.

[25] Cindy Marling, Edwina Rissland, and Agnar Aamodt. Integrations with case-based reasoning. *Knowledge Engineering Review*, 20:241–245, 2005.

[26] Tim McInerney and Demetri Terzopoulos. Deformable models. pages 127–145, 2000.

[27] Rajiv Mehrotra and James E. Gary. Feature-based retrieval of similar shapes. In *Proceedings of the Ninth International Conference on Data Engineering*, pages 108–115, Washington, DC, USA, 1993. IEEE Computer Society.

[28] Marcin Novotni and Reinhard Klein. A geometric approach to 3d object comparison. *smi*, 00:0167, 2001.

[29] N. R. Pal and S. K. Pal. A review on image segmentation techniques. *Pattern Recognition*, 26(9):1277–1294, 1993.

[30] Petra Perner. An architecture for a cbr image segmentation system. *Journal on Engineering Application in Artificial Intelligence*, 12(6):749–759, 1999.

[31] Petra Perner. Why case-based reasoning is attractive for image interpretation. In *ICCBR '01: Proceedings of the 4th International Conference on Case-Based Reasoning*, pages 27–43, London, UK, 2001. Springer-Verlag.

[32] Petra Perner, Thomas Günther, and Horst Perner. Airborne fungi identification by case-based reasoning. In *Workshop om CBR in Health Sciences, ICCBR03*, pages 63–72, 2003.

[33] Dzung L. Pham and Jerry L. Prince Chenyang Xu. A survey of current methods in medical image segmentation. *Annual Review of Biomedical Engineering*, 1998.

[34] B. W. Porter, R. Bareiss, and R. C. Holte. Concept learning and heuristic classification in weak-theory domains. *Artif. Intell.*, 45(1-2):229–263, 1990.

[35] Jan Puzicha, Joachim M. Buhmann, Yossi Rubner, and Carlo Tomasi. Empirical evaluation of dissimilarity measures for color and texture. In *ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2*, page 1165, Washington, DC, USA, 1999. IEEE Computer Society.

[36] Christopher K. Riesbeck and Roger C. Schank. *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 1989.

[37] Edwina L. Rissland and David B. Skalak. Cabaret: rule interpretation in a hybrid architecture. *Int. J. Man-Mach. Stud.*, 34(6):839–887, 1991.

[38] Simone Santini and Ramesh Jain. Similarity measures. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(9):871–883, 1999.

[39] Roger Schank. *Dynamic Memory: A theory of Reminding and Learning in Computers and People.* Cambrigde University Press, 1983.

[40] Arnold W. M. Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Content-based image retrieval at the end of the early years. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(12):1349–1380, 2000.

[41] John R. Smith and Shih-Fu Chang. Visualseek: a fully automated content-based image query system. In *MULTIMEDIA '96: Proceedings of the fourth ACM international conference on Multimedia*, pages 87–98, New York, NY, USA, 1996. ACM Press.

[42] Barry Smyth and Mark T. Keane. Remembering to forget: A competence preserving case deletion policy for cbr systems. In *International Joint Conference on Artificial Intelligence*, pages 377–382, 1995.

[43] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision 2. Edition.* Brooks/Cole Publishing Company, 1999.

[44] Amos Tversky. Features of similarity. *Psychological Review*, 84(4):327–352, 1977.

[45] J. S. M. Vergeest, S. Spanjaard, and Y. Song. Directed mean hausdorff distance of parameterized freeform shapes in 3d: a case study. *The Visual Computer*, 19:480–492, 2003.

[46] P. Zamperoni and V. Starovoitov. On measures of dissimiliarity between arbitrary gray-scale images. *International Journal of Shape Modeling*, 2(2&6):189–213, 1996.