Table of Contents

1 Introduction

This chapter contains the motivation and outlining of the paper.

1.1 Motivation

Segmentation is one of the more important fields in image processing, and also the base of any interpretation of objects in the image. This field is a challenge that has been studied for many years. There are many different approaches to the problem both based on mathematics and more ad hoc solutions. The development of faster computers has helped a lot, but still can not rival a human in any way.

Genetic algorithms have been used by many fields in image processing, and are often used in “ill posed” situations. It has already been used for helping single segmentation algorithms, but never in a scale as tried in this paper. This is what makes the challenge.

The purpose of this paper is to try to improve segmentation without changing or trying to improve the already existing segmentation algorithms. This will hopefully lay some foundation in the field that can be developed further. If successful it might help taking basic segmentation one step further.

The text of the assignment is written below [Figure 1-1].

Evaluate the benefits of having more than one segmentation algorithm available, when segmenting images from different domains with a genetic algorithm.

Use the genetic algorithm with one segmentation algorithm at a time, and compare the results to the use of more than one algorithm, and the ability to use a sequence of segmentation algorithms. The comparison should be pixel by pixel, hit or miss determined by a manual segmentation of the picture. Time, quality and the consistency between time and quality should be considered. The time to get an adequate solution. The quality of the solution.

Figure 1-1 Assignement

1.2 Outlining of the paper

The paper is divided into eight chapters. The different chapters contain.

- Chapter one is just an introduction.
- Chapter two and three contain the background knowledge that is needed.
- Chapter four combines the background knowledge for making the system.
- Chapter five is construction choices and implementation.
- Chapter six contains the results and discussion about them
- Chapter seven is the conclusion.
- Chapter seven is last with the references.

2 Image processing and understanding

Image processing and understanding are processes where a computer is preparing, analysing, interpreting and understanding information about and contained in an image [1]. There are many kinds of information that can be contained by an image, and also information about an image. It can be classification, recognition and relational information and so on. The ultimate goal is to get a computer to use images as well as a human and maybe better.

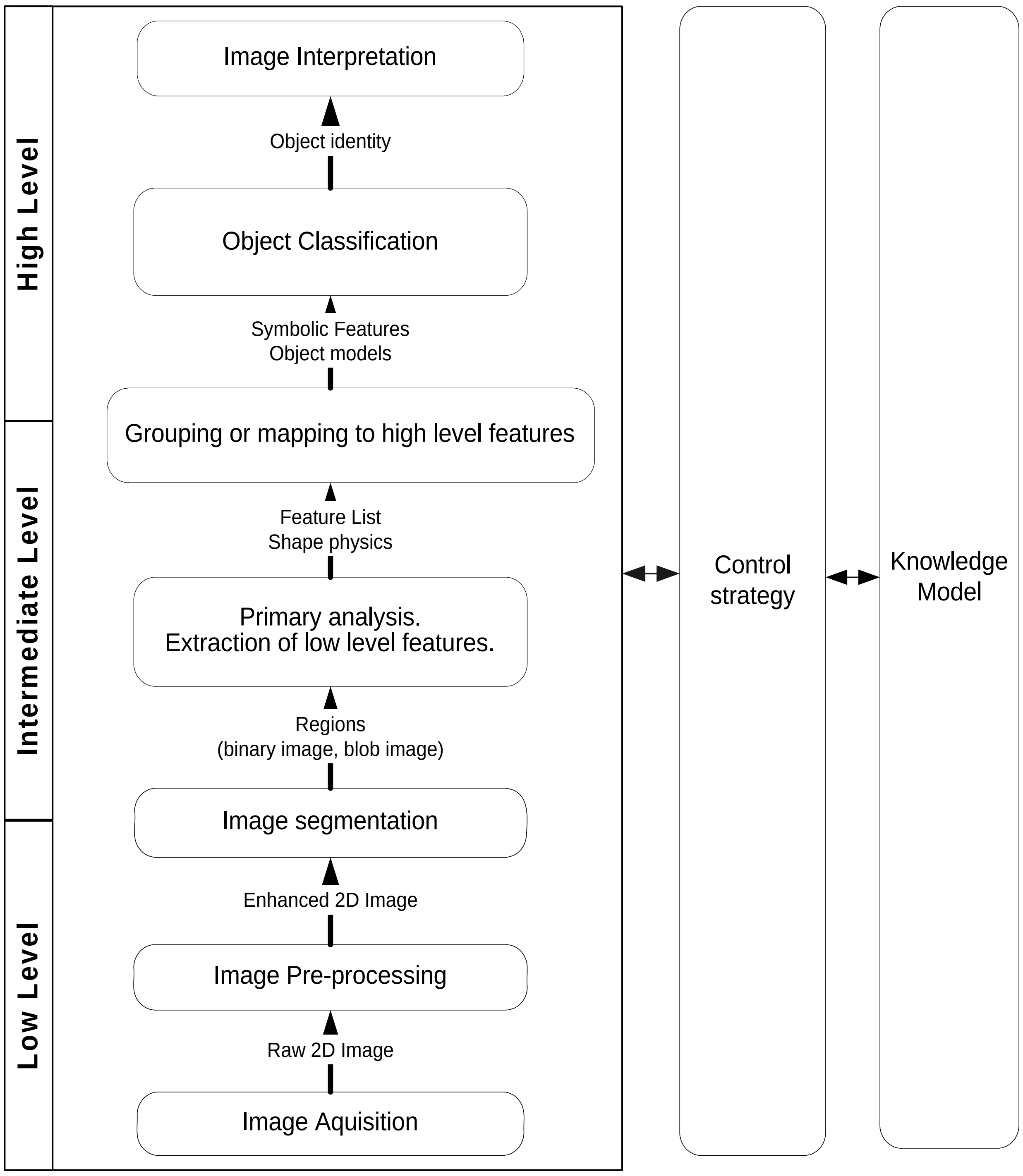


Figure 2-1 Image processing and understanding

There are many steps in processing and understanding an image [Figure 2-1] [1] and each step is built on the previous one. Even though each step has to get certain information from the previous step, the previous step can be completed in many different ways. This is as long as the step after receives the necessary information. For example step three has to give a set of regions to step four. There are many ways to segment an image without interfering much with the next steps. This is the reason that there are so many different ways to segment a picture. Some schemes have a certain domain where they are better suited than other schemes. Some need more background information than others. Some algorithms are designed to be combined with other algorithms to get a suitable result.

There is no one algorithm that is always the best choice. The success of an algorithm is always dependant on the domain and the knowledge about the domain. The domain can be everything from trees and dogs to magnetic resonance images or radar images.

2.1 Image processing techniques and representation

This chapter is about some of the principals in image processing.

2.1.1 Pre-processing

Pre-processing of images is all about preparing the image for processing. It can be everything from conversion and restoration to enhancement of the picture. Everything that is happening before the main algorithm can be viewed as pre-processing. Also including noise filtration, smoothing of areas, enhance edges and remove distortions.

Images are often represented as one channel grey images, or a three or four channel RGB images [2]. This may be dependent on the sensor or camera that captured the image. The first thing is to convert the image into a suitable colour space, for instance HIS (Hue Saturation Intensity) space [3]. Then the rest of the filtration and enhancement may be done. There is a lot of research in this area, [4][5][6] are some references. Pre-processing is often important to enhance output, but I will not go further into the matter in this report.

2.1.2 Segmentation

The segmentation process is the process of dividing an image into parts/segments. These segments are supposed to divide the image into non overlapping relatively homogenous regions. Regions that are close to each other can be merged if they have similar pixels [7][8][9]. Mathematically it can be written like this:

$$R = \bigcup_{i=1}^{S} R_i \qquad R_i \cap R_j = \emptyset \qquad i \neq j$$

This is for a complete segmentation of the image R with the final set of regions. $R_1 \ldots R_S$ [1].

Under-segmenting
Under-segmentation divides the picture into too few regions, and often removes the accuracy of the segments.

Over-segmenting
Over-segmentation divides the picture into too many regions. There are still too many details. I some cases this is a desirable starting point for segmentation algorithms.

2.1.3 Segmentation algorithms

There are a lot of segmentation algorithms with different qualities. Everything from knowledge based algorithms to statistical ones. The six first written about here are used in the system. The algorithms under "other well known algorithms" are just examples of other possible algorithms.

Threshold
The Threshold algorithm is one of the simplest segmentation algorithms [7]. There are some different versions of this algorithm, but the main idea is as following. A number is picked between the extremes of the picture values. Then all the values that are higher than that value is set to the maximum value and the other values set to the minimum value. There are ways to make this algorithm better and more complex. One way is to use more than one threshold, and thus getting more than two values in the result. This way will give more difficulties when deciding what the threshold values should be. Other schemes to improve this algorithm are dividing the picture in different parts before segmenting, or using some schemes for automatic determining a useful threshold value or values.

Watershed
The watershed algorithm is a region based segmentation algorithm [10]. An easy way to explain the watershed algorithm is to imagine the images as a landscape or three dimensional images with the pixel values as the height. Then the water level is raised step for step. The places where the pixels penetrate the surface new basins (segments) are made. These basins are expanded when the water level is raised further. When two basins merge a dam is made, so that the basins are kept separate. The problem with this algorithm is when one pixel could become a part of two or more basins. There are many ways of deciding which basin the pixel will become a part of, but no standard approach.

Edge based segmentation
An edge based algorithm uses the edges found in the image to segment it. There are some different ways to find edges. Using filers is often an easy way to do it. Both simple filters like a Sobel [7][11] filter and more complex like Gauss filter can be used [7]. What these filters do is to change the information in the image, so that sudden changes get a high value and low changes get a small value. Afterwards some other processing can be done. Like taking all the small value areas that are surrounded by high levels as segments.

Maximum likelihood
The maximum likelihood algorithm is a recursive algorithm [12]. It starts with the most likely connection between two parts often pixels. Then recalculates the probabilities between the parts/pixels and using the newly formed part as one part. This is done repeatedly until the result is satisfactory. This algorithm needs some information of number of segments or another way to stop it.

Mean Shift
The Mean Shift algorithm is a simple interactive procedure that shifts each data point to the average of data points in its neighbourhood [13]. This procedure is repeated several times to achieve the final answer. The way the neighbourhood is weighed is important. Different weightings give very different results.

Split Merge
Split merge is a recursive segmentation algorithm [7]. First it splits the image into many small and homogenous parts. The criterion for not splitting segments any further is quite strict, and divides the image into too many segments. Then the different segments are combined to remove extra boundaries. The rate of splitting and merging should be controlled when initializing the algorithm.

Other well known segmentation algorithms
There are quite a lot of other segmentation algorithms. Some of these are:

- Canny Edge Detector [7]
- Active contour models [7]
- Markov Random Fields segmentation [9]

2.2 Extraction of low level features

Features are short descriptions or properties based on the image [14]. The different features can be made from segments/regions, groups of segments or from the whole image. Low level features are very often numerical and represented in a feature vector like this $x=[x_1, x_2, \ldots x_D]$. The different features can be used to compare trial segmentation with a perfect set of features.

2.3 Grouping or mapping to high level features

High level features describe relation between regions or groups of regions. It can also be grouping of low level features. These high level features are often represented with symbols. It is important to choose which features that are appropriate to which object.

2.4 Object recognition

Object recognition/classification is identifying groups of regions based on the high level features. This is usually done by comparing high lave features to a classification of objects. This can be done by taking the Euclidian distance [15].

2.5 Image interpretation

Image interpretation is about the final “solution” for a system or “the little bit extra”. This is done by evaluating the relation between the objects. For example understanding more about the situation the objects are in. This is most often done by AI-related reasoning. The interpretation is based on a knowledge models. Two possible systems types here are logic based systems and case based reasoning [16].

2.6 Domain knowledge

This chapter is a short introduction to domain knowledge in image understanding. This is a quite important part when making an algorithm that is supposed to work with different domains.

To group, recognise, and understanding an image in an automatic image understanding process, is often an “ill-posed” problem [17]. There are lot of difficulties that the system has to be able to deal with. It can be everything from lighting to shadows, distortions in the lens, object rotation and so on. Domain knowledge is knowledge that can help telling the algorithm something about the picture and thus helping the image processing and understanding process. The domain knowledge can be all kinds of information. It can be everything from possible objects in the image to time of day. The more domain knowledge there is about an image, the smaller the search space will be for the system.

2.6.1 Knowledge representation

Domain knowledge is knowledge and expertise from a special field that a computer program is fed, given or taught [18][19]. The knowledge may be procedural or declarative [9]. Procedural is how to do a thing. This is hard coded knowledge. While declarative knowledge is about a thing, often symbolic and not hard coded.

2.6.2 Different kinds of domain knowledge

The domain knowledge in image processing and understanding can often be divided into two kinds of knowledge. These are non-pictorial knowledge and image knowledge. Non-pictorial knowledge is information about the image [20] such as:

- Type of sensor
- Where the image was taken
- Time of day
- Kind of weather
- Time of Year

The image knowledge is the information about the properties in the image. Like pixel ranges and information about the spectrum and so forth. One key element about domain knowledge is what knowledge to store. A good choice here can save a lot of time and really improve the search.

2.7 Case Based Reasoning

The reason for looking at Case Based Reasoning (CBR) [21][22][23], is that this is a group of systems where the system like the one proposed in this report can be used. Not directly for processing but for making the first case base, or processing cases that are not already in the case base. In case based reasoning the experience in the system is stored as cases in a case database. Each case contains a description of the problem, a solution and often also some results. To solve a new problem, it is compared to the other problems already stored in the database. Similar problems are retrieved as a probable solution. Then the solution is run, and the results checked. Finally the new problem is stored in the database. One problem with this approach is to make a start database with cases, and when the new problems are far from the cases already stored. This is solved by either manually solving the problem or solving it by using a system like the one proposed in this paper. An example of a CBR cycle is shown in [Figure 2-2][23].

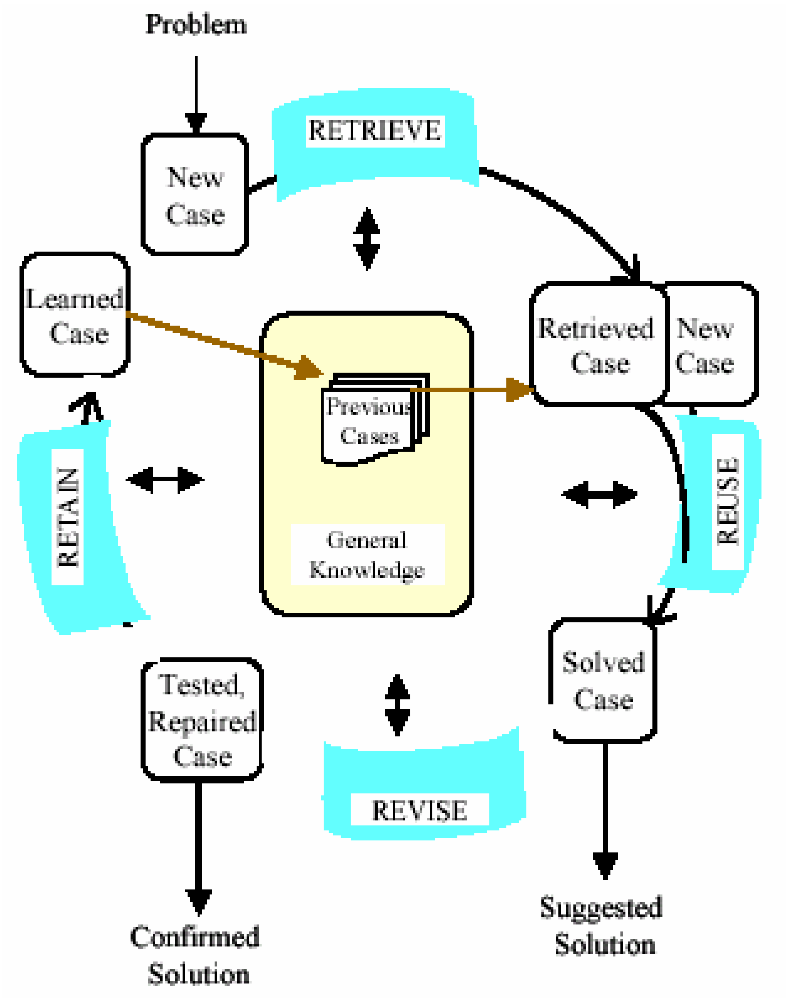


Figure 2-2 CBR cycle

The papers [21] and [24] by Petra Perner are about CBR used in image processing and understanding systems. In [24] CBR is used to optimize low level segmentation, and in [21] and [9] CBR is used throughout the Image understanding process.

3 Genetic algorithms

Genetic algorithms are based on nature's evolution, and are used in problems that are "ill-posed". Especially when certain criteria are fulfilled [1] [16] [25]:

- The search space is big complex and poorly understood
- Domain knowledge is scarce or the expert knowledge is hard to use for ordinary searches
- No adequate mathematical analysis
- Traditional search methods fail

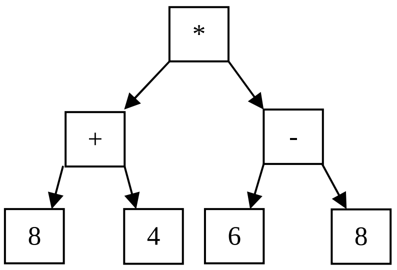
In nature, natural selection is used to find which chromosomes that gets to pass on its genes [9][26]. The genes that are passed on to the next generation are done so through mating. In this way nature will always adapt itself to the situation it is in [27]. This quality is the basic principle that is tried to recreated in genetic algorithms [16][25]. In genetic algorithms chromosomes is often called chromosomes. Genetic algorithms are about shuffle, change and combining genes. Some will be passed on to the next generation through mating and mutation, and some will not. The following three criteria have to be fulfilled to make a genetic algorithm work. The criteria are:

- The chromosomes have to be designed.
- A fitness function for evaluating the chromosomes (gives fitness values)
- A method to choose chromosomes based on fitness must be constructed.

Throughout the whole process a gene pool is used. A gene pool is a collection of chromosomes that contains different genes. The gene pool is developed in steps from generation to generation, just like in the nature. In contrast to nature, genetic algorithms are often designed to find one "best" chromosome and not a whole pool of chromosomes. There can be some exceptions when algorithms need more width in the final result.

3.1 Generation of chromosomes

All the chromosomes in a genetic algorithm must have a structure. How the structure should be, depends on the problem it is designed for. The information the chromosomes are going to contain comes from the problem, directly or indirectly. The problem may also give directions to how the information should be organized. The information can be structured in a multitude of ways. The structure can for example be simple binary values in a array with fixed length, or numbers with a big variation in arrays of different lengths. It does not have to be arrays either. The chromosomes can also be represented by other structures like trees [Figure 3-1] [28], or more specialized structures for more special purposes [29].



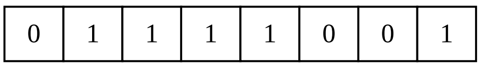


Figure 3-1 Tree structure (left) Array structure (right)

If the chromosomes are represented in a very complex way, it can lead to problems with pairing/mating (crossing) and mutation. Possible problems might be solved by creating new functions for mutating and pairing. If the representation have little flexibility, it may not be possible to do all the kinds of crossing. In that way the representation put down limitations of ways the population can develop. The cause of more complex representations can be that the problem contains certain structures like trees, strings or connection of values that do not correspond to real numbers or floating numbers. Problems can also arise because of rigid length or ordering of the numbers.

If you take the other approach and try to make very free and easy structures, another problem may arise. If the chromosomes are not made complex enough, it is not certain that the problem can be represented in a satisfactory way. One example of this is not being able to exploiting the properties of the problem in the search. It is a difficult compromise to find a good structure, because if it is too rigid, the system might not be able to search the whole search space.

There is much to consider when making the structure of the chromosome. The chromosomes can often be broken down into smaller groups or single values. If there are some properties in the chromosome that are more closely tied than other, it may be beneficial to group these parts together. The more the structure is divided in parts, the more rigid will the structure be. The values contained in the structure will also command the different ways the chromosomes can be paired and mutated. Values in the genes that are uniform in range and type may be interchanged, but there may be problems with values that are not uniform. With uniform values a part in the beginning of one string can be put in an arbitrary part of another string and still produce a valid structure. This is easy if the gene strings consists of one kind of values. If there are different types of values, there have to be constructed special functions to get the same interchangeability. Even then the effect will not be the same as with uniform values. The properties of the values also command how you can mutate the values in a sensible way [30].

All the chromosomes in the pool must be constructed in the first generation. There are many ways to generate a start generation. The easiest can be to make all the chromosomes with random values. This will create a new start point every time. It is also possible to devise a method that from certain criteria prefers some values in favor of others. In this way it is possible to get some control on how the start pool looks [31]. The start pool can effectively shorten the search time for the optimal chromosome, but might lead to neglect of other solutions if there are other possibilities that are acceptable. It might also be desirable to make some new chromosomes in later generation to prevent that the genes are so unidirectional that the algorithm get stuck in local maxima/minima.

3.2 Pairing and mutation

There are many ways to change chromosomes in the development of new generations. The most common ways of doing this is divided into two groups. These groups are paring and mutation. Inside these groups, there are many different possibilities, and it is also possible to do both on the same chromosome.

3.2.1 Pairing

Pairing is briefly told, to choose two chromosomes and mix parts of the chromosomes with each other [1][16][25]. In pairing the structure plays an important part. In a tree structure it will be natural to do pairing in a different way than with strings of genes. The rigidity of the length of the chromosomes is also important for the possibilities in pairing of chromosomes. Chromosomes with a rigid length will often have reduced capability possibilities when it comes to pairing.

There are different ways of pairing chromosomes. One of the simplest ones is the one point crossover [Figure 3-2]. This method is about finding a random point, and switching the last part of the chromosomes. The result is two chromosomes that might give better result than the two original chromosomes. It is also possible to make a two point crossover. This is the same as a one point crossover, only with two points. This effect can be achieved by making two one point crossovers with different points. Doing one point crossovers several times with different points, will result in a multipoint crossover [32]. If the chromosomes can have different lengths, it is possible to choose different crossover points in the two chromosomes that are getting paired [Figure 3-3 Non uniform crossover]. This will give different lengths on the result chromosomes.

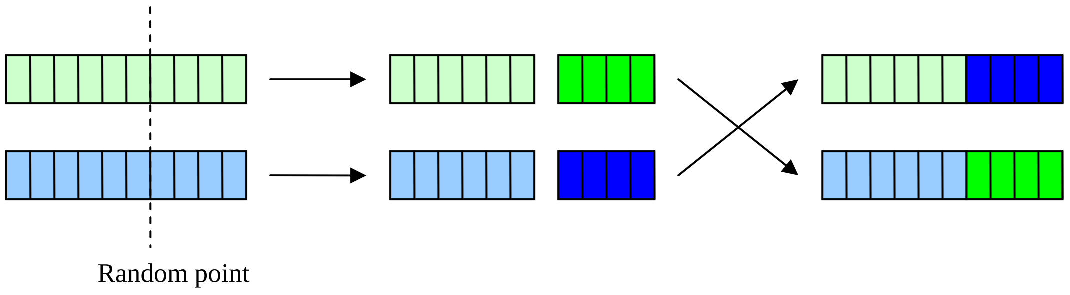


Figure 3-2 Uniform crossover

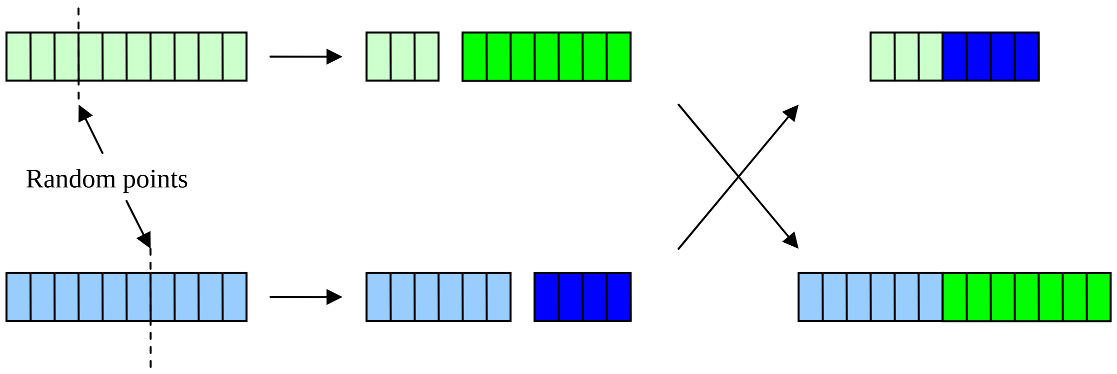


Figure 3-3 Non uniform crossover

Pairing can also be done on other structures. If doing a crossover with tree structures [Figure 3-4 Tree structure crossover], a different kind of dividing point must be found in the structure. Everything from that point and below should be changed [28]. Except from this it works like with strings. All the different structures will have different ways to be crossed.

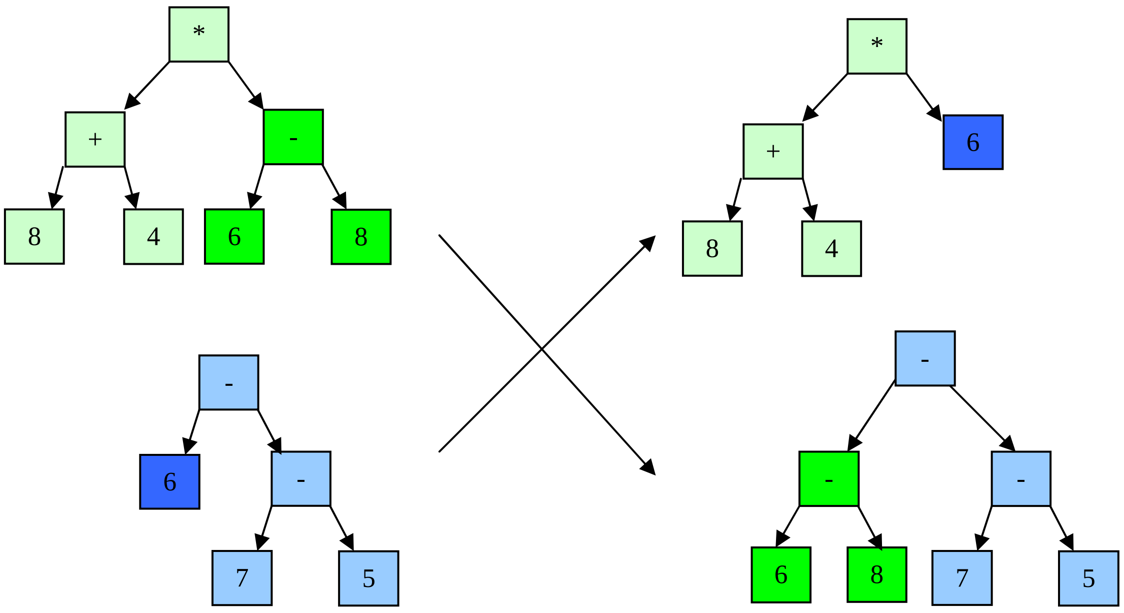


Figure 3-4 Tree structure crossover

3.2.2 Mutation

In the development from generation to generation it is also possible to mutate a chromosome which is getting to the next generation. Both chromosomes that are already paired and chromosomes that are not paired can be mutated. Mutation is simply taking a random part of the chromosome and changing it [Figure 3-5]. The part that is changed is often only one value, but if there are some connections between several values there might be wise to change all the parts that belong together. The change can either be random, or done according to some kind of distribution.

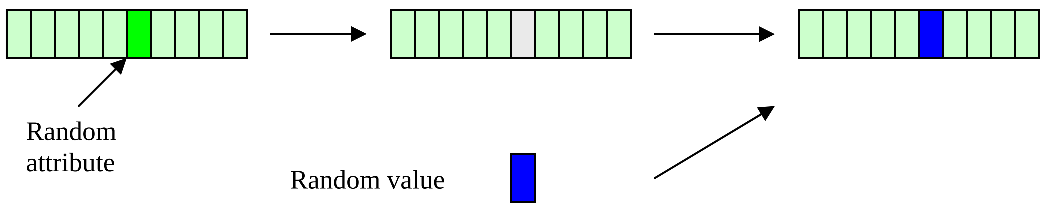


Figure 3-5 Mutation

The different distributions that can be used depend on the structure and type of value. If it is a bit, it will be either inversion or random setting. If it is a real number, it is possible to use all kinds of distributions. Gauss distribution or uniform distribution is often used for mutation. The gauss distribution can simply be control by setting different variances. It can for example be possible to use this for simulated annealing.

3.2.3 Evaluation

All the chromosomes have to be evaluated to find which chromosomes that is supposed to be paired and mutated. The function that tests the chromosome have to be based on the problem that is to be solved [1][16][25]. The purpose is to make a function that gives better and better value the closer the attempt is the perfect answer. The function that does this is called a fitness function and the chromosome gets a fitness value.

The evaluation function should not take too much processing time in comparison to the time used by each chromosome. This because then the algorithm will be very slow, and the whole genetic algorithm will be slower. The hard part is to find a good compromise. The result has to be a function that gives good enough results in reasonable time. A fitness function that gives a perfect distance to the optimal answer would be perfect [Figure 3-6]. The next best is a function that gives values that is strongly rising toward the global maximum, or strongly sinking towards the global minimum [Figure 3-7]. Unfortunately these fitness functions are often very difficult to make, if not possible. That means the function have one or more local minima/maxima [Figure 3-8]. It might also be that the function has some parts with flat eras or areas that are sloping toward local maxima/minima. These areas can easily lead to bad results, and a bigger number of chromosomes or more luck is needed to hit areas that slope to acceptable values. Sometimes a local maximum/minimum can be good enough, but often it is not desirable.

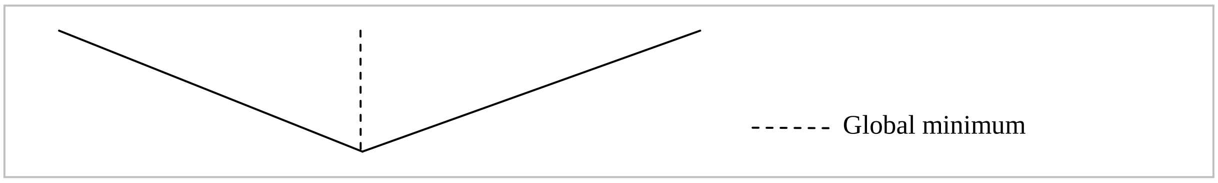


Figure 3-6 Perfect fitness function

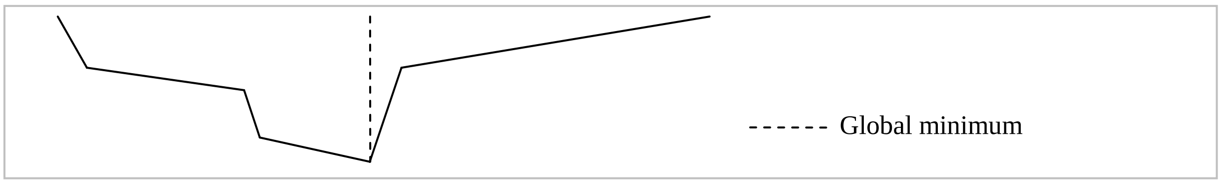


Figure 3-7 Strongly increasing fitness function

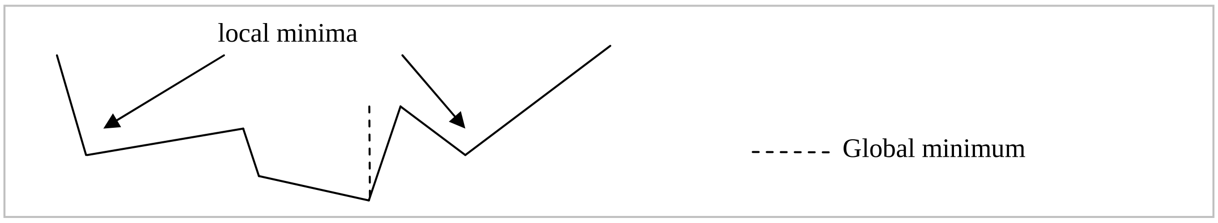


Figure 3-8 Fitness function with local minima

3.3 Selection

Which chromosomes that are used further and what actions made on which chromosomes selected are based on the fitness value [Figure 3-9] [1] [16] [25]. There are many ways to choose between the chromosomes. If the fitness function always gives a correct distance from perfect fitness it is easy to choose chromosomes based on the fitness value. For example a double fitness value can give double the possibility of a certain chromosome is chosen. If it is a strongly sinking or raising it is also possible to choose the closest chromosomes. The better the fitness function is, the narrower the selection can be. That means if a fitness function might have some local minima/maxima, it might be good to make a broader selection of chromosomes. As the pool develops it might be a good idea to change the level of favoring fitness. This can be done by simulated annealing. The different levels of favoring are for maximizing the speed, but the same time getting a broad enough search.

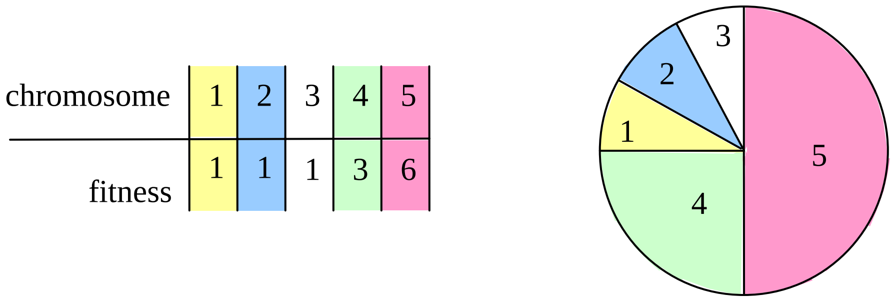


Figure 3-9 Fitness selection wheel

3.4 Evolution

Genetic algorithms are not guaranteed to come up with an optimal solution in reasonable time, but are an attempt to get an adequate solution within reasonable time. In genetic algorithms there is developed a gene pool generation for generation. Every generation the chromosomes go through an evaluation, selection and possible pairing and mutation. The result of this gives the next generation. The genetic algorithm can be written like this in pseudo code [1] [16]:

```
Start genetic algorithm
    Time = 0
    Generate first generation
    Condition (While not finished)
        Evaluate chromosomes
        Select chromosomes
        Do pairing and mutations
        Keep chromosomes that are going to next generation
        Set Time = Time +1
        Go to condition
    End condition
End algorithm
```

3.4.1 Size of gene pool

The size of the gene pool should be determined by several different factors. Two important factors are the quality of the fitness function and the size of the search space. If the fitness function has few locale minima/maxima, fewer chromosomes will be needed because it will be more areas that lead to satisfactory results. A bigger search space can give a reason to increase the number of chromosomes. The number of chromosomes does not have to be constant. It can vary from generation to generation, but there should be a thought-through policy on how to increase or shrink the size of the gene pool. If getting too many chromosomes there will be a lot of processing per generation, and too few too few will cause a narrower search.

3.4.2 Search intensity

To avoid problems with locale minima/maxima it is a good idea to start with a wide spread of chromosomes in the search space. As the search progress the search might be narrow around the best results. There are different ways to change the search width. It is possible to increase the favoring of good fitness values. Also intensifying mutation or restrict the reach of the mutation will narrow the search. If a gauss function is used, it is easy to lower the variance to achieve a narrower search. There are also possibilities to change the way the pairing is done, but this is often specific to how the chromosomes are structured and what is represented in the chromosomes.

One problem by changing the selection and mutation during the search is when to intensify the search. It is very difficult to know in advance how fast the algorithm will converge towards a solution. How fast the algorithm is converging is often different each time it is run.

There are two way to deal with this problem. One approach is to set a fixed number of generations for changing the intensity. Another is to set some criteria regarding fitness values in the gene pool. It might be very difficult setting a good parameter for how many generations the algorithm needs to get a good enough answer, because it may differ each time you run it.

3.4.3 Stop criteria

When deciding criteria for when to stop the algorithm, it is important to guess the quality of the chromosome and how far from the best chromosome the system is [1][16][25]. It is very seldom that these things are known. It may just be a local minimum/maximum that is found. The choice of stop criterion should be tightly tied to how dense the search is. It is good to have a very dense search around the best results just before stopping the algorithm.

It is possible to set a number of generations the algorithm will do. This is a very easy solution that can be combined with simulated annealing. The number of generation will then be the criterion for how dense the search is. This approach can be adjusted from search to search to better suit the problem. Often a more automatic way to do this is wanted.

There are many possibilities when making a stop criterion. It is also possible to have more than one stop criterion. The fitness is the most common factor in the criterion beside the number of generation it is working on. The criterion may be something as easy as a threshold telling that if a better fitness than the threshold is found, the algorithm is stopped. If using a threshold, it may give some problems. If it is too easy to achieve, it will be stopping the algorithm to early, and the other way around. This is often solved by using more than one criterion.

3.5 Image processing with genetic algorithms

There are many way to use genetic algorithms in image understanding. This is because genetic algorithms can be used in fields where the search space is very big. It is also possible to evaluate how good an attempt is in image understanding thus making it possible to make some sort of fitness function.

In many cases the objects that can appear in the image are already known. This makes the problem, not knowing what kind of distortions the object might be exposed to. From another angle the system is given a distorted image and is supposed to find what in the image that is interesting. Finally you often have to classify the important parts of the image [33][34][35], and this connects to the objects that might be in the image.

Fitness is often based on some ideal version when genetic algorithms are used more directly in image understanding. Or more correctly the distance to this perfect version. If the object can be anywhere in the image, it might be far too many possibilities, and to difficult to find good ways to compare pixel with pixel. The challenge is to find out what kind of representation that is desirable. This is of course in addition to make methods that can estimate the difference between the ideal representation and the representation found in the image.

Segmentation with different algorithms without genetic algorithm
In [36] there are used different algorithms to segment an image. Four algorithms are used, but are not put in any kind of sequence. The scheme is to select the best algorithm based on properties in the image. The selection procedure has to be taught interactively, but is automatic after fully trained. The algorithm with the highest rank based on the properties form the image, is regarded as optimal when used automatically. Most of the paper is about the selection algorithm and the training of it. A test system is trained and tested to see how good the system actually can be. The trained selection algorithm was tested on 9000 images and the result was a 97.5 percent correct selection of segmentation algorithm.

Narrowing search space
Genetic algorithms can also be used to narrow the search space in image understanding. This is done in [37]. The goal here is to diagnose the image of a retina. The genetic algorithm is not used directly, but is used to find out which parts/objects that are of most interest. Reducing the number of parts that are searched for will make the algorithm faster. When using the most significant parts the accuracy will not be compromised too much. The processing cost will be less with no regard to which algorithm you use afterwards.

Help for neural net that segments images
Genetic algorithms can also be used to help other algorithms to process images. In [38] genetic algorithms are use to help and train a neural net to better classify images. The neural net is trained by sending pixels through a training network. The chromosomes have two parts, one contains how much change the training is supposed to cause, and the other how big influence previous training should have. It is checked if the training is better or worse by taking the Euclidian distance from the right classification. With the help of genetic algorithms they evolve better and better training, so that they can train the network at the right speed. It is shown that all the genes that are left have more or less the same values. The result is a good estimate on the best parameters for training the neural net.

3.6 Segmentation with genetic algorithms

There are many ways to use genetic algorithms in segmentation [8][39][40]. Here two articles using genetic algorithms for segmenting are summarized

Tuning range segmentation
In [41] is a genetic algorithm used in segmentation. The system is not segmenting regular color or gray scale image, but Range Images. Range Images are pictures that show how far away an item/object is. Every pixel in the image shows the distance to the point it hits, instead of showing the color of that point. This representation gives somewhat special qualities when trying to segment. The system has to find surfaces/planes that correspond to the values in the image, in stead of finding areas with similar colors. The surfaces may be curved, plane or any other shape that may be captured in this way. Another difference from ordinary pictures is the edges. In range images the edges are not big differences in value. The edges in the images are just changes in plane normal. It is the surfaces converging there that dictate if it is an edge or not.

It is not used a lot of time on the segmentation algorithm in [41]. They already have two algorithm to segment Range Images. The genetic algorithm is used to optimize the already existing algorithms. The problem to optimize parameters in the algorithm is caused by an uneven function with a lot of local maxima/minima. They have chosen to use a genetic algorithm because it is a widely tested and utilized type of method. The special part here is the way they use the gene pool in the search. This pool is increased and reduced in an uneven fashion. The method that adds new chromosomes is partially random. The removal of the chromosome is also random, but also based some on fitness and age of the chromosome. To make a fitness value they have four different values they put together to one value.

- Correct segmentation
- Over-segmentation
- Under-segmentation
- Miss segmentation
- Noise segmentation

In the article there are two algorithms to adjust the values to. Both algorithms are segmentation algorithms. That means that the same evaluation function is used for both algorithms. The genes have to be built for each of the two algorithms because of different parameters in the different algorithms. The rest is reusable without much modification. The results from using the genetic algorithm to find the parameters work excellent on one of the algorithms. On the other it worked only ok, but in that algorithm there were no data on what the different parameters meant. That implies it was impossible to put relations between the different parameters.

Segmentation with long term gene pool
In article [42] normal distortions of the images found in the real world are handled. It might be the different time of day, time of year and type of weather. The genetic algorithms are used for searching through the parameter combination for segmentation. Before segmenting the picture, a quite extensive statistical analysis is done on the picture. This reveals different qualities in the picture. This pre information is used to choose chromosomes in the start-pool for the genetic algorithm. The genetic algorithm is then run until a satisfactory result.

The most special and interesting part in this article, is the way chromosomes are put in the start pool [42]. They keep the fitness values from the pre analysis, parameters from the genetic algorithm and fitness value. These values are kept in a long term gene pool. This long term gene pool is modified every time the genetic algorithm produces a new result. This gene pool is helping to choose what parameters should be used. This technique seems to achieve good results.

Transformation search
There are possibilities for using genetic algorithms also after the segmentation is finished. The result pool can have different results that are parts of the perfect answer. In [43] a different angle than usual segmentation is used. Here matching partially visible object in an image is attempted. The purpose is to find the least amount of transforms to get to the reference image. It can be many reasons that the object is only partially visible. One possibility is that the object is partially covered by other objects. Another is that parts of the object is in the shadow such a way that it is difficult to segment as one object.

A genetic algorithm is set up to search the different transformations. As an evaluation the transformed picture is compared to a solution image. The fitness function is just checking if the images have the same pixels on the same place. The chromosomes are sets of transformations that are used before the result transformation is tested. This way a set of transformations giving the best possible hit compared with the solution picture is found. This example is just a little experiment that shows good opportunities, but demands some more work.

4 Theory foundation

There are many aspects when making a genetic algorithm with a pool of segmentation algorithms. This chapter is about what parts are needed and the problems that can arise when trying to build such a system. The first chapter is a short summary of what is preferred for a complete commercial system. The second chapter is some more detailed thoughts around each part and its problems and solutions. The system is examined part by part, because of the complexity of the system.

To test all the aspects that arise around such a system a lot more time is needed. The goal is to make a test system and see how the results are, and what might be done further. The results will show if a genetic algorithm with a pool of segmentation algorithms is feasible and will work.

4.1 Concept

The problem can be divided into several parts. Here is a short summary of what each part should contain.

4.1.1 Segmentation algorithms

The first part is to get a large pool with segmentation algorithms, preprocessing algorithms and maybe other algorithms that might be useful. All the algorithms should take several parameters that control how they function. It might be possible to make more than one pool if there are many algorithms.

4.1.2 Genetic algorithm

The genetic algorithm has to be able to choose algorithms from an algorithm pool or pools. It also has to be able to sequence them, so that more than one algorithm may be in a chromosome.

There are several functions that have to be possible to do with the chromosomes. They are:

- Re-sequencing algorithms inside the chromosomes
- Pairing of two chromosomes
- Mutate an chromosome

Some functions have to be possible to run on the algorithms. These are:

- Pairing of two algorithms
- Mutate an algorithm

A way to combine the possibilities of the chromosomes and the algorithms has to be devised. This solution has to be flexible, because it is the center of the system and has to accommodate many different algorithms. It also should have stop criteria, so that it may be stopped when further processing is not needed.

4.1.3 Fitness function

This is a part of the genetic algorithm, but a very important one. A fitness function has to be made. It has to give a clear fitness even though there are several different segmentation algorithms involved. The fitness function can either be generic and used by many domains, or it has to be possible to change the function. This means either one general fitness function or several fitness functions that are used for different domains.

4.1.4 Result presentation

It is desirable to be able to view the different results and the progress of the system. The results have to be stored in a way that it is possible to analyze the progress as well as the results. The storing of the results should be done in such a way that the same results are not repeated, but also easy to view.

4.2 Thorough study

There are a lot of different problems preparing for such a system like this. Exactly how of the different parts that will affect each others, is very difficult to say without really doing thorough calculation. It is not even certain that such calculations are possible at all. The reason for this is because of the complexity of the system and the genetic algorithm part of the system is based on chance. The complexity is why the system is taken part by part when examined. Both problems and some solutions are looked at. It may not be all the problems and solutions but the more general ones.

4.2.1 Segmentation algorithms

The first part is finding a lot of different segmentation algorithms. There are several possibilities for getting segmentation algorithms. They are:

- Implement algorithms from scratch
- Implement algorithms from a pseudo code
- Translate algorithms from other programming languages
- Finding already implemented algorithms

The first three options take much work especially the first one, but will also give control on how the different parameters are used in the code. Finding an already implemented algorithm does not give control like this without getting the source code. If the source code is available it is possible to make some changes, but this will take time. There are also some more things to consider when choosing segmentation algorithms. It is not just to take a lot of segmentation algorithms. They should be based on different principals and/or function in different ways.

This will give different solutions to the segmentation. There are also some other properties that has to be considered when selecting algorithms. These properties are:

- Quality of the segmentation
- Processing time
- Number of attributes
- Search space

The quality of the segmentation is obvious why is important. The others are important especially since they are used in a genetic algorithm.

The speed of an algorithm has to be taken into the account when deciding which algorithms to take into the pool. If an algorithm is very slow, it will take a lot of time to test the chromosomes with this algorithm. That means the genetic algorithm will be very slow, and will use a lot of time to get a satisfactory result. Even if just one algorithm is slow, it will have considerable consequences for the sped of the genetic algorithm. Also more algorithms will always take more time. This is because the search space will be larger and needs more chromosomes [Figure 4-1]. The time issue will also apply if adding preprocessing algorithms. It may be possible that in some cases preprocessing algorithms can speed up some of the segmentation algorithms. This is because of more uniform image to segment thus less processing, but it might also shrink the size of the search space if not optional. Also if an algorithm has more “permutations” of attributes there will be a larger search space.

If there are different sizes of the search space for the different algorithms, there may become problems with the favoring of algorithms. Algorithms with small search spaces may get better before the other algorithms have come far enough in their search.

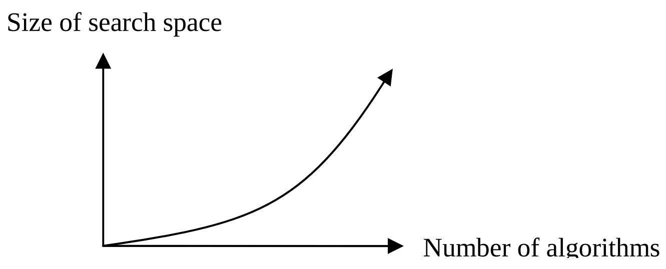


Figure 4-1 Search space and number of algorithms

4.2.2 Genetic algorithm

The genetic algorithm is the core of the system. This is where all the parts come together, and it is very important to get it as robust yet flexible as possible.

4.2.2.1 Chromosome design

The first thing is how long the chromosomes should be. The possible length of the chromosome will have big influence on the size of the search space. Since it has influence on the search space it also influences time the genetic algorithm will need. The possibility of longer chromosomes will not only slow the genetic algorithm down, it will give the opportunity to search a larger search space. The possible length of chromosomes should depend on the algorithms in the pool. Here are four factors quite important.

- Processing time
- Search space
- If prone to under or over segmentation
- If it converts the image

Processing time and search space will have impact on how long it will take for the genetic algorithm to find a good result. This might make the algorithm very slow if combined with long chromosomes. If prone to under or over segmentation and if it converts the picture will tell more about the possibility of improving the results by having longer sequences of algorithms.

The length of the chromosomes will put restrictions on the structure of the chromosomes. If allowed to have only one algorithm per chromosome, it will be easier to design the chromosomes than if it has to be variable length. The chromosomes that allow flexible length are probably the best way for this kind of system, because of the larger search space and flexibility.

The next thing to consider when designing the structure is if there are some divisions of the values in the chromosomes. The top level of values is algorithms. The difference in how the preprocessing algorithms and segmentation algorithms function can make it possible to divide the different kinds of algorithms [Figure 4-2]. An example is one pool for pre processing algorithms, one pool for over-segmenting algorithms and one for under-segmenting algorithms. All the different pools can be set to a suitable size. The search space is effectively decreased, if dividing the different kinds of algorithms. This will cause some algorithms to always come before others. Preprocessing algorithms are often a little bit different, when it comes to having many in a row. They often work best with many after each other, because they only change the information in the picture a little bit. For example blurring and median filter. The will both have a good effect even when the other algorithm have run.

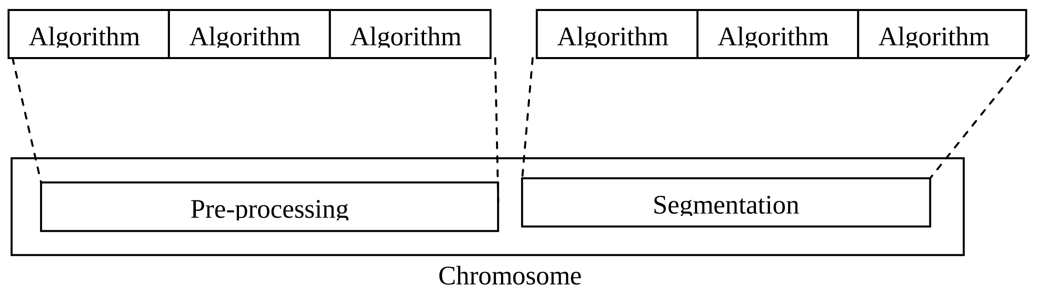


Figure 4-2 Divition of pre-processing and segmentation algorithms

The chromosomes consist of different algorithms, in different order. When it is possible have different algorithms on different places in a chromosome, it can be important knowing which algorithm is where. It is important If only similar algorithms are allowed to be paired. The algorithm type can be represented in many different ways. It can be a number, name or some other form of identification. It is not a problem when only using one and one algorithm, but it might be important when having more than one.

4.2.2.2 Mutation

When it comes to mutation there are some problems. The different attributes to the different algorithms do not necessarily have the same accuracy and range. A parameter might be an integer ranging from one to ten, and another might be a float between zero and one. There are to ways to deal with this problem. One is to try to standardize the algorithm parameters converting the numbers from a pre specified accuracy and range. Then all the values can be mutated quite easy, and then converted back to the specified accuracy and range. There are some problems with this approach, not just the rewriting of the algorithms. It might also cause problems if using some gauss function when mutating numbers. One example here can be if a parameter can only have two values, one and zero. When the parameter is changed in another scale and range, is might be that the parameter is almost never changed to the other value. Another possibility is to add extra information about the numbers. The information could be accuracy and range of the parameters. This will make more work and more information to keep track of.

The chromosomes contain algorithms, and the algorithms all have different attributes. This gives several possible ways to mutate a chromosome. For example:

- Mutating an random part even across algorithm boundaries
- Mutating an attribute to an algorithm
- Mutation an algorithm with attributes

All three types of mutation have advantages and problems. The best solution is probably to use more than one of these alternatives when making the system.

If trying to mutate across algorithm boundaries [Figure 4-3] there are some problems. If the change improves one algorithm, it might be bad for the next. Also the values that are mutated do not necessarily have a connection. This mean that connected attributes are not mutated together, at least only by coincidence. This may cause the genetic algorithm to use more time.

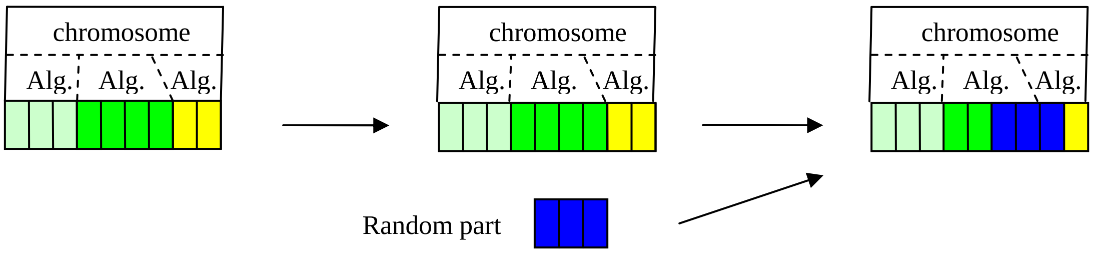


Figure 4-3 Mutation across boundaries

Only mutating one attribute each mutation [Figure 4-4], can make the genetic algorithm very slow. It also might be very difficult to get a wide enough spread in the search space without having a very large start pool.

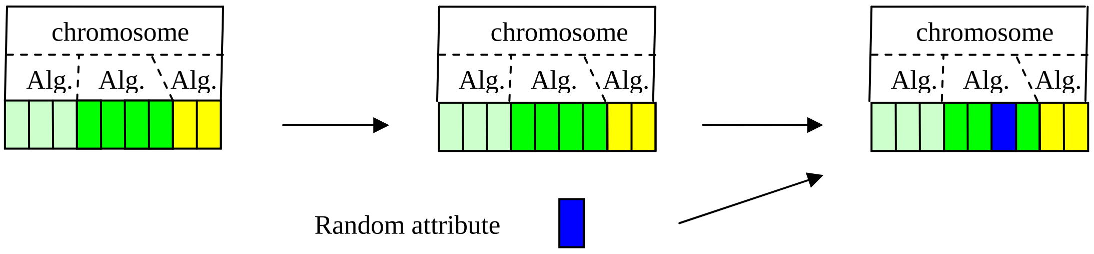


Figure 4-4 Single attribute mutation

Mutating all the attributes in an algorithm is the third option. When all the attributes in an algorithm is simultaneously changed, no parts of the existing and maybe successful algorithm is kept. This means that there is just as probable to get a good algorithm the first time the algorithm is generated, as when it is mutated. When all the attributes are replaced it is possible to change the kind of algorithm also [Figure 4-5]. This can help to change sequences of algorithms in the chromosomes. If the algorithm is changed, it might be a problem that the new algorithm might have a different number of attributes.

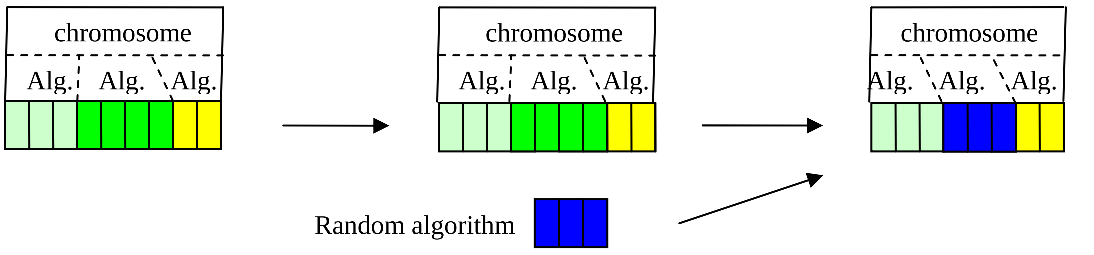


Figure 4-5 Mutation of algorithm

4.2.2.3 Pairing

There are several ways to pair chromosomes or algorithms. The one pairing method that is the most common is crossover and is the one considered here. Some of the same problems as with morphing are with pairing too. Making a crossover between two chromosomes of different kind can cause problems. One problem is if the parameters that are crossed do not have a common range and accuracy. It is best to have uniform values when pairing parameters. If the values are not uniform in pairing the values can get corrupted. Even if the values are uniform, there may still be problems with how many parameters the different algorithms have. This will also affect the rest of the chromosomes. When pairing two algorithms, something has to be done with the extra parameters on the largest algorithm or chromosome. It is possible to ignore the extra attributes or put in some extra values in the shortest algorithm or chromosome. Putting in extra attributes can make some values in good chromosomes or algorithms prevail even though they are not contributing to the result. If ignoring the last parameters in the longest algorithm or chromosome, it will cause those parts not to improve. This will cause the genetic algorithm to miss some parts of the search space.

Pairing of similar algorithms is possible [Figure 4-6]. It is far fetched to try to find chromosomes with the same sequence of algorithms, but it is possible to find equal algorithms to pair. The problem is how to find the equal algorithms. One way is to store knowledge about which algorithms are where. It might also be useful to have a register over which algorithms are where. With a register like this it is easy to find to algorithms that are similar. There are many possible schemes to choose a kind of algorithm, for example how many algorithms there are of each type. Fitness based selection is also possible. One way to do this easy is to store the fitness value along with the algorithm in the register.

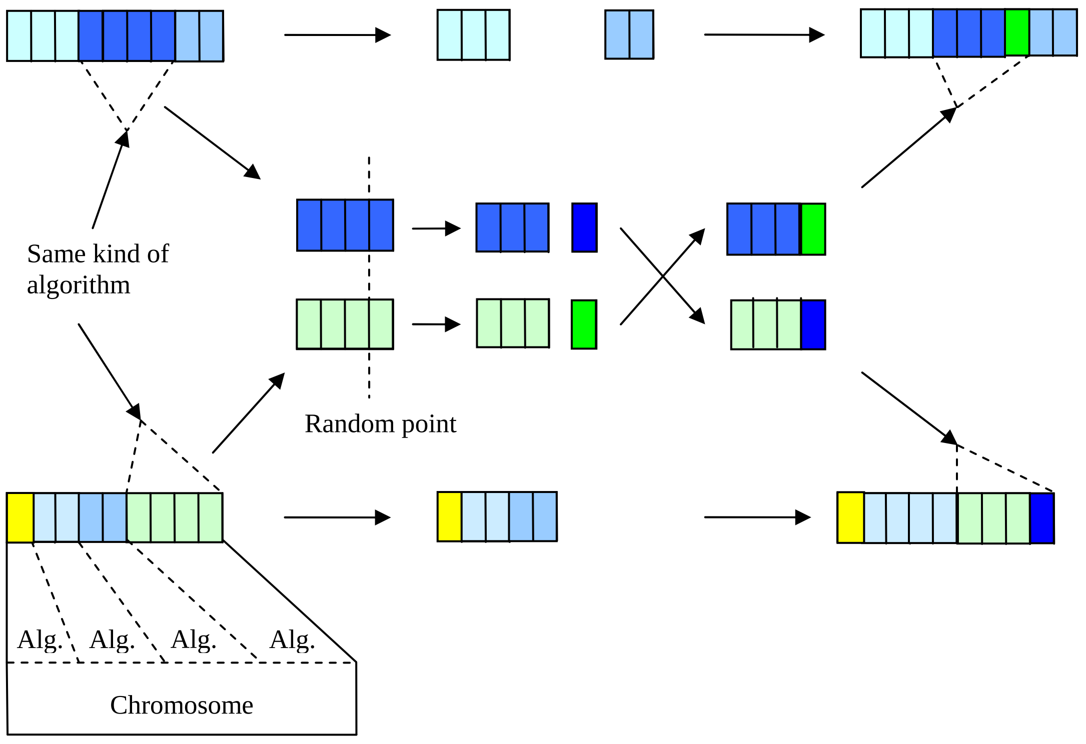


Figure 4-6 Pairing of similar algorithms in chromosomes

It is easy to find two chromosomes of the same type, id you have a register. Without a register there are two different methods. Both methods will use some extra processing time to find the algorithms. One of the methods is to choose a chromosome based on fitness value. When an algorithm is chosen a search through the other chromosomes for similar algorithms is made. Then when all the similar algorithms are found, the second algorithm is chosen based on the chromosomes' fitness. The second algorithm chosen, is chosen based both on type of algorithm and of fitness of the chromosome. The other method is to choose two chromosomes. They can be chosen either by random or by fitness. If the two chromosomes do not have the same algorithm, two new chromosomes are chosen. This method not only favors fitness but also algorithms that occur often in the gene pool. How much extra work is generated, because of chromosomes not having the similar algorithms, depends on how many algorithms there are in each chromosome. If the gene pool contains the same number of algorithms of all the different algorithms, then the probability of pairing is the same.

It might be preferable to try to change the ordering of the algorithms. This can be done by pairing chromosomes on algorithm level [Figure 4-7]. Here you get some of the same problems like with mutation of whole algorithms. The problem lies in knowing which algorithms are where, the number of parameters and choice of structure. It is important if you have a constant or variable length on the chromosomes. A constant length will only give room for uniform crossover, while a variable length will give room for a non uniform crossover.

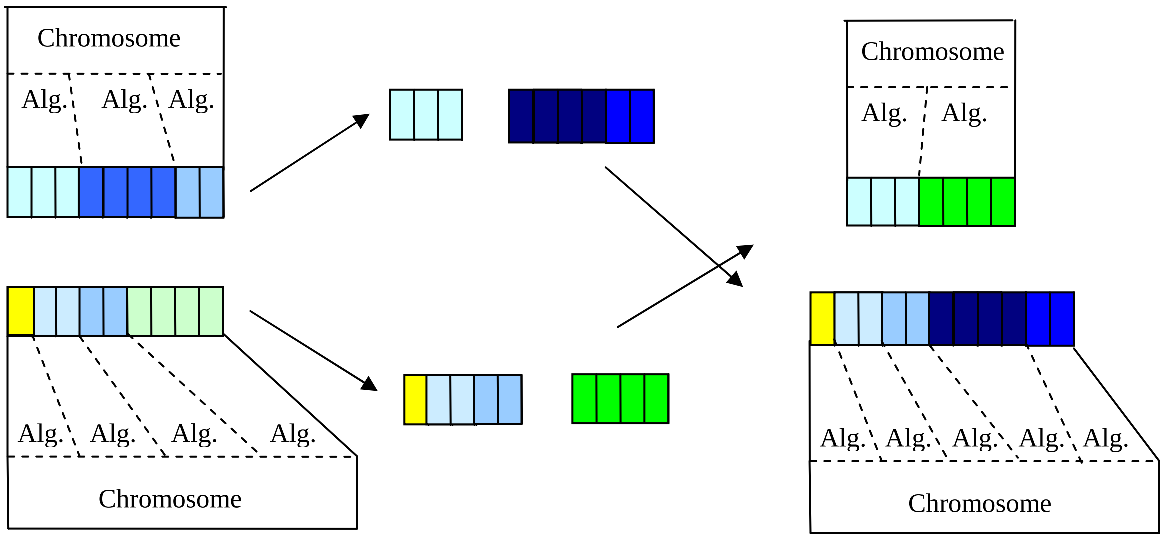


Figure 4-7 Pairing of chromosomes

4.2.2.4 Search space

The number of chromosomes you need to search the whole search space, depends on how many algorithms and how many parameters the different algorithms have. It might be possible to reduce the search space in different ways. One possibility is to use a kind of pre evaluation that say something of the properties in the image. This may give some sort of guidance on what the start gene pool should contain. Another is by combining other gene pools that already exist [Figure 4-8]. How to make these gene pools is under experience in the next section.

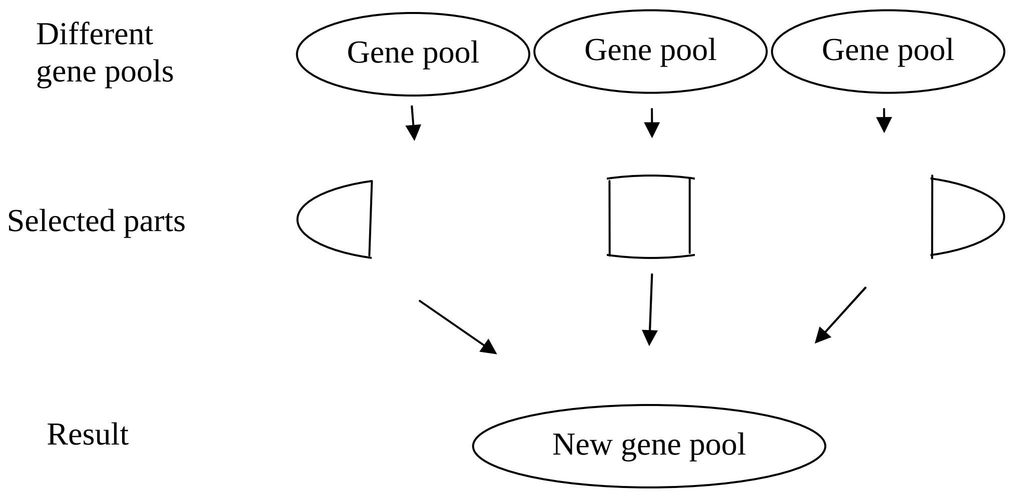


Figure 4-8 Combining gene pools

One problem that is always with genetic algorithm is time. It is a very big search space and many algorithms that have to be run to test the different chromosomes. This amounts to a lot of processing time. There are some ways to better the situation. The main areas are choice of language, choice of segmentation algorithms and increasing the efficiency of the algorithms. It might also be possible to do some parallel processing and/or make a distributed processing system. This will save time, but not processing. The time issue is not given much attention in this report, because of the uncertainty without proper testing and experiments.

4.2.2.5 Evolution

Evolution is the transformation of the gene pool. The best way of doing this depends on the search space. There are different ways of manipulating the search of the genetic algorithms. One common scheme that will work well on this system is simulated annealing. There are many places in the algorithm where simulated annealing can be used. How simulated annealing will perform in the different places in the genetic algorithm can be difficult to predict without thorough testing. The most common places to do simulated annealing are:

- the selection process
- the mutation process
- the rate of mutation and pairing

In the selection process it is not necessarily only in the weighting of the fitness simulated annealing is used. It can also be used for selecting type of function to use. In the mutation process it may be used to change the variance in selection of new parameters. It should be used with different distribution for the different attributes with different range and accuracy. The frequency of mutation and pairing can also be changed when reaching the best fitness. It might be desirable to change the frequency of attribute pairing and chromosome pairing. These suggestions can give a very good control in the development of the algorithm when the chromosomes get better and better.

4.2.2.6 Experience

It might be wise with a more long term storing of the gene pools to keep the accumulated experience. One gene pool for each domain can avoid conflicting domains deteriorating the pool. This also makes it possible to start images form the same domain with a pre compiled gene pool. Each new image will train the pool a little more. This training can also continue when the program is used when the “training period” is over. In this way the system will be in constant training and constantly improving. How much of the gene pool that is kept as a long term pool is difficult to say. A smaller gene pool will give less processing each time you get a new image unless it is padded with new fresh chromosomes. It will also give less possibility to improve. How much of the pool that is kept should be based on what it is going to be used for.

The gene pool’s usage is not necessarily limited to its own domain. If there are domains with similarities, it might be desirable to take some genes from more than one long term gene pool and mix as a start pool. This might make the start gene pool better than just taking from one long term pool. It is really difficult to find out if these schemes help significantly or helps at all. It will depend on the properties of the different domains, and how the rest of the genetic algorithm is functioning.

A finished gene pool can be stored for use by other programs than genetic algorithms. It might be for some program that segment images inside one domain, or if the program can automatically choose between domains based on previous stored information. An example of this is Case based reasoning (CBR). A genetic algorithm like this could make the case base in the CBR-program.

4.2.3 Fitness function

A fitness function for such a system has to be based on some mode. There are many ways to describe images and segments. It can be everything from manually segmented models to statistical or descriptive models. The problem is to compare the models of what you are trying to find and the test segmentation. Preferably the result tells how far from the perfect goal the test segmentation is. To make a function that does this across many different domains is very difficult. One solution might be to just take a pixel for pixel comparison. There are a number of problems with this:

- Every image has to be manually segmented
- The object can not be placed other places in the image
- A function calculating fitness based on hits and misses has to be devised

A pixel by pixel comparison will not give credit for round shapes or anything like that. It will also be a quite big job doing the manual segmentation, and of course it will not be automatic for new images. It might still be useful for making a case base for CBR.

There might be some sort of representation that may be generic enough that it will be sufficient for several domains. Mostly it will depend on the different domains. If divided into different domains, there are some choices to make on a domain to domain basis. The choices made should reflect the segmentation goal in that domain. It is a big difference segmenting all the patches in a patchwork quilt and segmenting a ball on a desk, and this should be reflected in the fitness function for the domain.

It is also possible to do some more high level evaluation of the images, for determining the fitness. It can be relations between segments and so on. This will take quite some time for each chromosome. The method for calculation the fitness value should be compared to the time it takes to run each chromosome. If each chromosome take a lot of time, then it can be good to have a quite large fitness function. This will make a good fitness function needs fewer chromosomes to find an acceptable value, because of fewer local minima/maxima. Fewer chromosomes will lead to a faster genetic algorithm. The problem is to make fitness function as accurate as possible within reasonable time.

4.2.4 Result representation

The results of the genetic algorithm are more than the gene pool. It is also the result from the segmentations and how many pairings and mutation that is made. All these results have to be available for analysis and improvement. The information that might be interesting to save is:

- The result images
- The chromosomes' fitness
- The chromosomes in each generation
- The pairings
- The mutations

There are several ways to store information like this All this information has to be linked in some way or another to be effective for analysis, and the representation form is important for easing the analysis. One way is to put some of the information as name for the different images. This will make it easier to link it to other information that is textual.

5 Construction

This chapter is about the implementation and the choices made when making the system and some of the reasons for those choices. The first segment is about the choices for the system. The second segment is how the choices are implemented and realized.

5.1 Choices

The system that is made is a genetic algorithm with an algorithm pool with segmentation and preprocessing algorithms. It is only a test system, and not a full system with many different features. The idea is make a system to test the feasibility and possible advantages. Some parts are made for this paper and some are improvements and adjusted parts from "fordypningsprosjekt" [1]. In addition some external libraries are used.

5.1.1 Segmentation algorithms

The algorithms that are translated or come from external libraries in the previous system are kept as they were. These algorithms are:

- Mean Shift algorithm
- Maximum Likelihood algorithm
- Split Merge algorithm

The algorithms Maximum Likelihood and Split Merge have approximately the same search space. Mean Shift have a much larger search space.

A threshold algorithm from "fordypningsprosjekt" is expanded to a multi threshold algorithm. This gives the threshold algorithm more possibilities and makes the search space much bigger. This makes the search space for the different algorithms more even. In addition two new algorithms are implemented. These are:

- EdgeBased algorithm
- Watershed algorithm

The EdgeBased algorithm has a Sobel filter for enhancing the edges [7]. The watershed uses the general algorithm idea explained earlier. Both algorithms are made partially from scratch and have approximately the same search space, but smaller than the other algorithms. This is because these algorithms are more process driven and not that attribute driven. The upside is that they make more diversity in the algorithm pool.

There are no single preprocessing algorithms in the pool. The closest to preprocessing in this system are smoothing part tied into a couple of algorithms. This is for speeding up the algorithms and for limiting the memory usage.

5.1.2 Genetic algorithm

The system uses the same basic genetic algorithm as in “fordypningsprosjekt”, to save time implementing. It is based on an external library that has been modified and fixed. When making this system, some parts have been changed to make it fit the purpose of this system better.

The first thing about the genetic algorithm is the structuring of the genes. It is a tree structure with two layers [Figure 5-1]. The first layer contains which algorithms are where and in which order. The second layer contains which parameters the algorithms are using. There are several reasons for making two layers in the chromosomes, but the biggest reason it that this structure makes it easy to change algorithms that do not have the same number of attributes. Also it makes no difference which algorithm is where, and it is easy to add and remove segmentation algorithms from the algorithm pool.

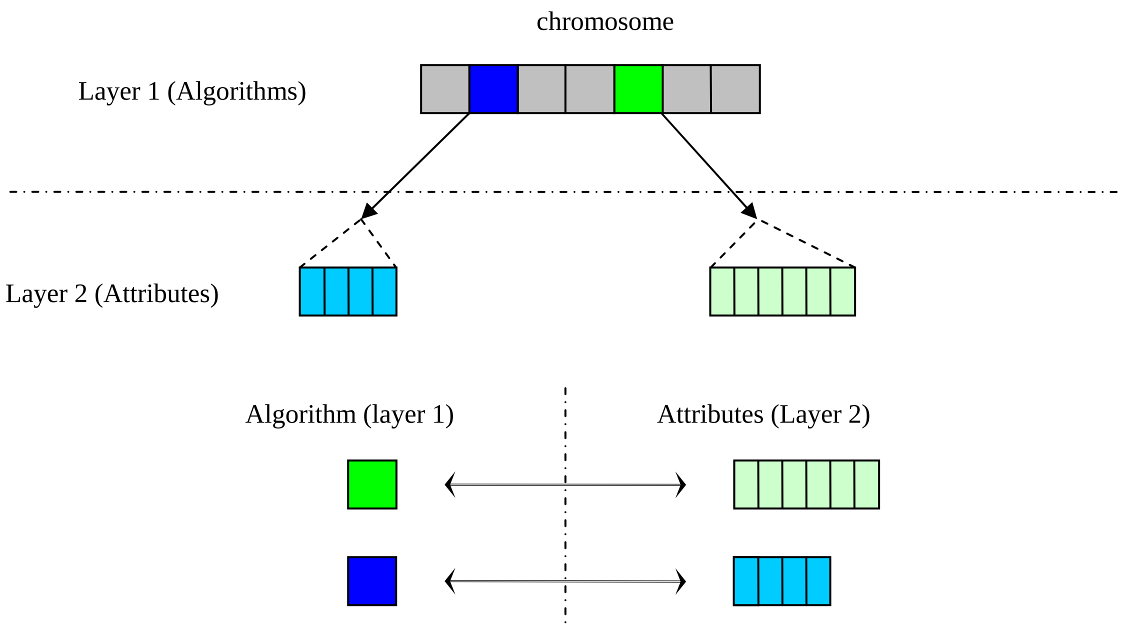


Figure 5-1 Chromosome structure

The structure makes every algorithm equal to the chromosome, in that way that there is no difference in what you can do with it. The only thing that separate the different algorithms, is that only similar algorithms can pair on an attribute level. To mutate and pair algorithms in a functional way, it is up to the algorithms themselves to do this when told to. The order to do this comes from the genetic algorithm.

The attributes to the segmentation algorithms have not been forced to change range. All the attributes are stored as float numbers. Each algorithm has a set of attributes, and specifies the range of the values. Float numbers are chosen because it can accommodate both integers and floats. Integers are made with a quick truncation. This method let the different algorithms have different ranges and accuracies.

5.1.2.1 Mutation

Both algorithm and chromosomes can be mutated in this system. The algorithm mutation is only a one attribute mutation [Figure 5-1], and that is why it is important to be able to mutate whole algorithms too. The mutation of chromosomes is really mutation of whole algorithms and gives the possibility to change the type of algorithm [Figure 5-2]. This gives the possibility to search more different combinations of algorithms. The mutation of attributes takes care of the smaller mutation and small adjustments. The chromosome mutation takes care of the bigger changes. This is done by will making new random algorithm from the pool of algorithms. All the algorithms can take over for each other, or take over for another version of itself. The algorithms created will not just be the algorithm, but also randomly selected attributes. This is also needed when making a new start pool. There is not made an attempt to make relations between the different algorithms when mutating whole algorithms.

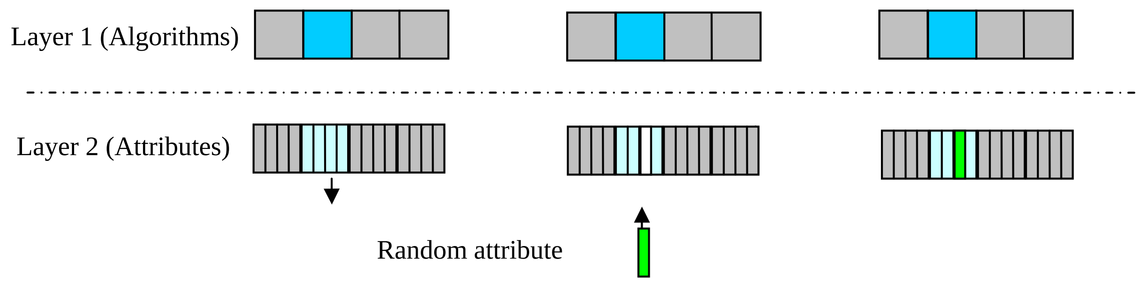


Figure 5-1 Algorithm mutation

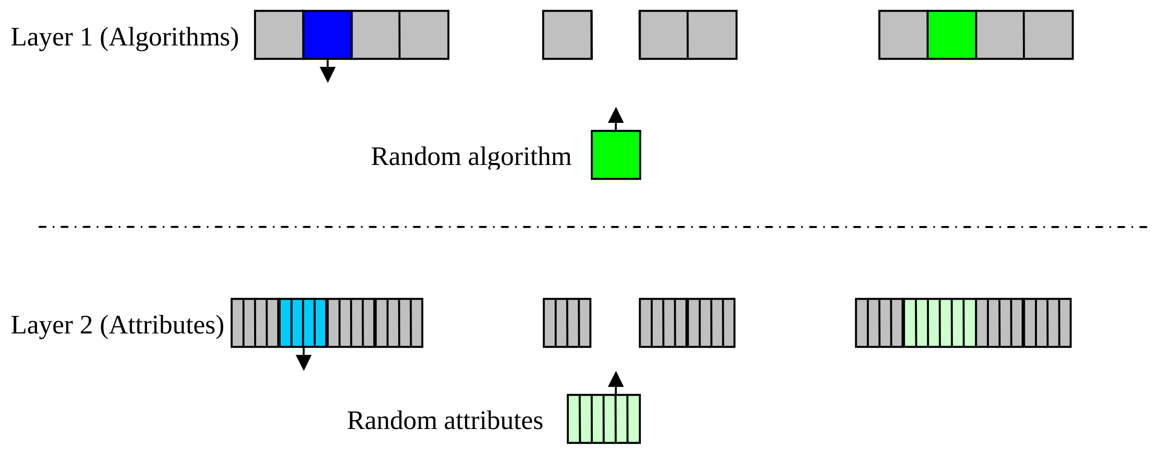


Figure 5-2 Chromosome mutation

5.1.2.2 Pairing

Both algorithms and chromosomes can be paired in this system. Both parings are done as a single point crossover. The pairing of chromosomes is just taking two chromosomes and taking a single point non uniform crossover with algorithms as the smallest entity [Figure 5-3]. It has to be non uniform because there may be different lengths of the different chromosomes.

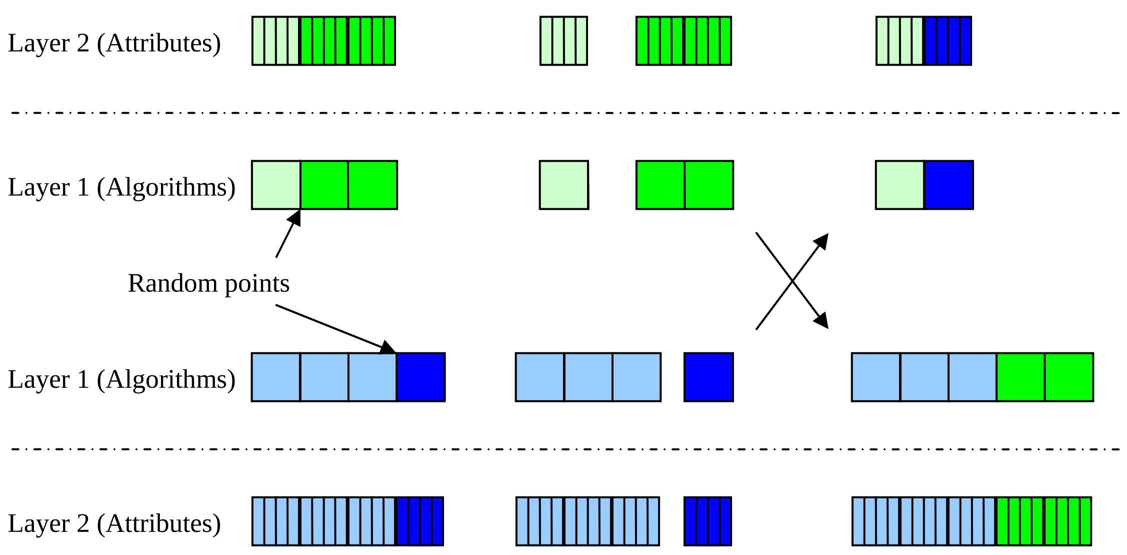


Figure 5-3 Chromosome crossover

The pairing of two algorithms has to be with two algorithms of the same kind. This is to avoid any problems with the number of attributes. The pairing of the two algorithms is a uniform one point crossover. It has to be a uniform crossover because of the equal length both before and after pairing. The problem when trying to pair two algorithms is that there is no register over where the different algorithms are. This is because of trying to keep the genetic algorithm as simple as possible, but it makes it difficult to find two algorithms that are the same. To do this, all the chromosomes have to be searched through to find out which algorithms are where.

5.1.2.3 Selection

Both for mutation and pairing it is needed to choose chromosomes, algorithms and parameters from the gene pool. The selection of the chromosomes is happening by ranking the chromosomes based on fitness. How big the difference in fitness value does not matter when using a ranking system. When algorithms or attributes are wanted, they are just picked at random from the selected chromosome. This is because there are no intermediate fitness values, only one fitness value for the whole chromosome. It helps to keep the genetic algorithm as simple as possible.

5.1.2.4 Evolution

The best chromosome is kept without mutation or pairing. This is to ensure that the best gene is kept, even though it is just pure luck. There are more chromosomes that are not mutated or paired, but this is random. Only the worst chromosome is always forced to either mutate or be paired to get to the next generation.

Simulated annealing is not implemented, and there is not made any special way to stop the genetic algorithm. This is because the library used for genetic algorithm does not support it. To stop the genetic algorithms, only the number of generation is used. That means that the number of generations has to be set based on the problem at hand. It might cause some extra problems to get the right number of generation. Often it might be wise to have too many generations than too few.

5.1.3 Fitness function

This system has a simple pixel by pixel fitness function. This means that every image has to be manually segmented. This is not a big problem because it is only a test system. The pixel by pixel comparison gives only hits and misses. The challenge is to use these numbers as good as possible. Several different functions were tried. Some testing was done until the choice fell on a very simple function of taking hit minus miss, but not allowing negative numbers.

The function tries to match each segment to each reference segment. The segment that has the highest hit minus miss value is chosen. The fitness value is the sum of these numbers for the different reference segments. This function is very simple, but is converging nicely towards the best image. The only problem is that the function does not converge satisfactory when miss is more that hit. The reason for not allowing negative scores is that one segment could then ruin the fitness even though the other segments are good. Also it would be difficult to choose between a small hit with big misses and no hits but few misses [Figure 5-].

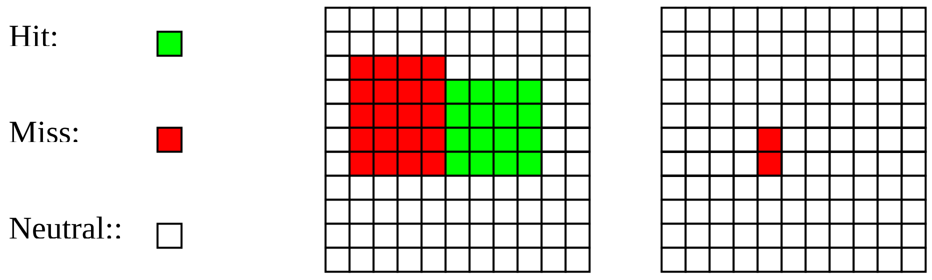


Figure 5-5 big hit and big miss (left) no hits but small miss (right)

5.1.4 Result representation

When it come to the long term gene pool, the gene pool is only stored as text. All the chromosomes in the generation with matching fitness are stored as text. The images made by the genes are also stored. There is not made a method for loading chromosomes in this system. This gives only insight in the time it takes from scratch and not the time from a carefully prepared gene pool. Loading chromosomes is not implemented because it will take a lot of time to calibrate the inclusion of chromosomes from stored gene pools.

5.1.5 Dataflow

Below is a data flow diagram for the system [Figure 5-4]. The path to the left follows the image and the comparison. The path to the left follows the chromosomes in the genetic algorithm.

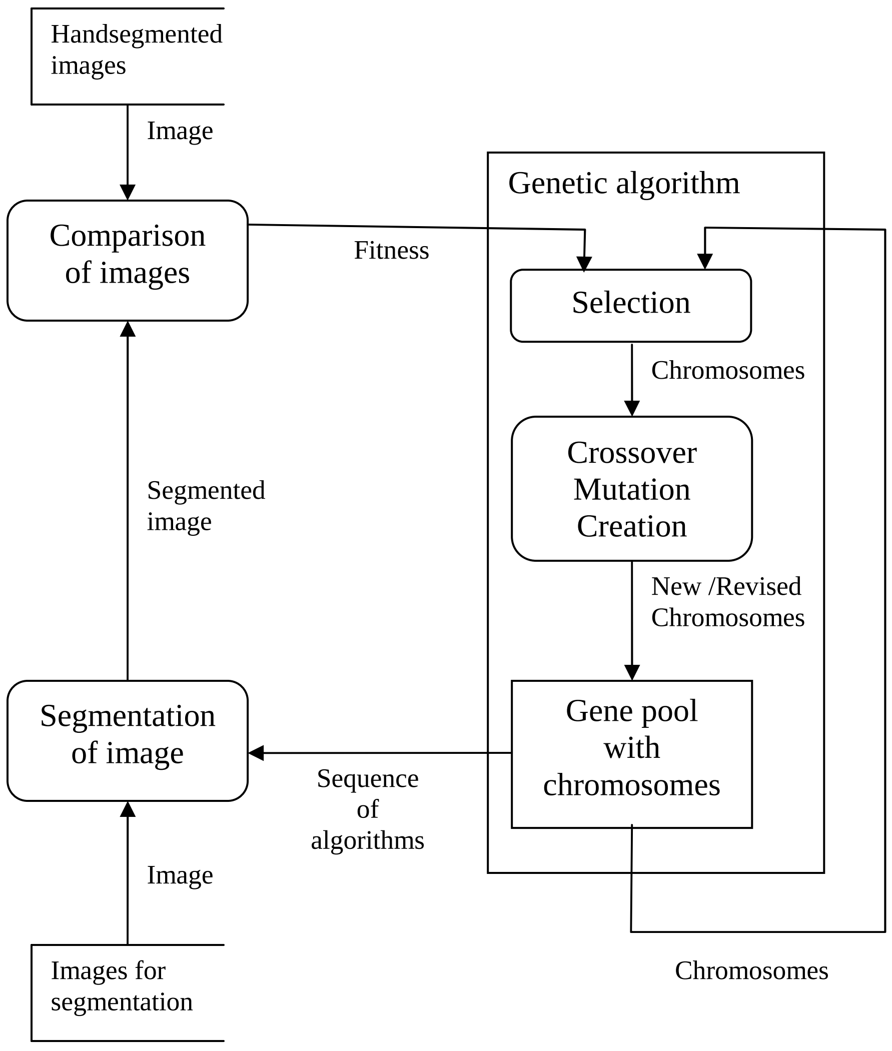


Figure 5-4 Dataflow diagram

5.2 Implementation

The source code, a running version and JavaDoc is put on the CD that comes with this paper.

The programming language used in this system is Java. The main problem with Java is the speed, but this it is a test program and need not to be very fast. The parts of the system from "fordypningsprosjekt" were also made in Java. The development platform is Eclipse [44]. The system consists of:

- external libraries
- code from "fordypningsprosjekt" that is improved or changed,
- code made for this project

5.2.1 Functionality

This chapter explains which inputs are required and which outputs that are given. The system is a command-line based. It is not made a graphic user interface on the system, because the system is many processing images and very little interaction.

5.2.1.1 Input

There are inputs of two images and some attributes.

The two images are the original image that is to be segmented and the manually segmented image. Both images have to be of the same size and preferable be .bmp images with a colour depth of 24 bit. The path to the manually segmented image is one of the attributes, the images to segment is located in the tilSegmentering folder.

The attributes are for setting parameters for the genetic algorithm. In correct order, the parameters are the following:

- The path to the manually segmented image
- The folder for the image to segment
- The destination file for textual output
- The probability for paring to occur
- Maximum number of algorithms when generating new chromosomes
- The size of the gene pool
- Probability of crossover when pairing
- Random selection probability without regard of fitness
- The number of generation to run
- The probability mutating or pairing between chromosomes not algorithms

There are used some different representation for this parameters. Some are intergers, and some are float or text. The reason for all the different kind of values is both because how they are used and also because of the external library.

5.2.1.2 Output

For each chromosome there are generated an image that is stored in the outImages folder. The filename is constructed by concatenating information about the picture, and is constructed like this:

img_"fitness"_"100000/fitness"_"generation"_" chromosome number".bmp

In addition to all the images that are stored the generation information is stored in a "results.txt" file located in the source folder. This file contains the following:

- The name of the image that is loaded and tried to segment
- The name of the different algorithms followed by their parameters
- The number for the generation and the fitness of the chromosome
- The crossovers and mutations made from generation to generation.
- If two algorithms are equal and an chromosome is replaced

5.2.2 External libraries

The system uses three different external libraries. These are:

The jEdison library[46]
The KUIM System[45]
Java Genetic Algorithm library[47]

Semgnetation
The jEdison[46] library contains a Mean Shift algorithm and is directly used in the system. This library previously translated from C++.

The KUIM System[46] library contains both a Split Merge algorithm and a Maximum Likelihood. It was in C so it was translated for the "fordypningsprosjekt".

Genetic algorithm
To avoid making the whole basic motor for the genetic algorithm, the GALib(Java Genetic Algorithm (JAR) library) [47] is used. The library had some shortcomings for its use in this system, so it is done several changes and bug fixes to the core of the library. One problem that is not fixed is that it does not have the possibility to set different stop criteria.

5.2.3 Program Structure

This chapter show the structure of the packages and classes, and which packages are using the other ones. Most of the packages in the no.ntnu.stud.imageUnderstanding package, are using the classes in that package. All the packages are shown here in the figure below [Figure 5-5]. A complete class diagram over the system and the different packages are in [Appendix A].

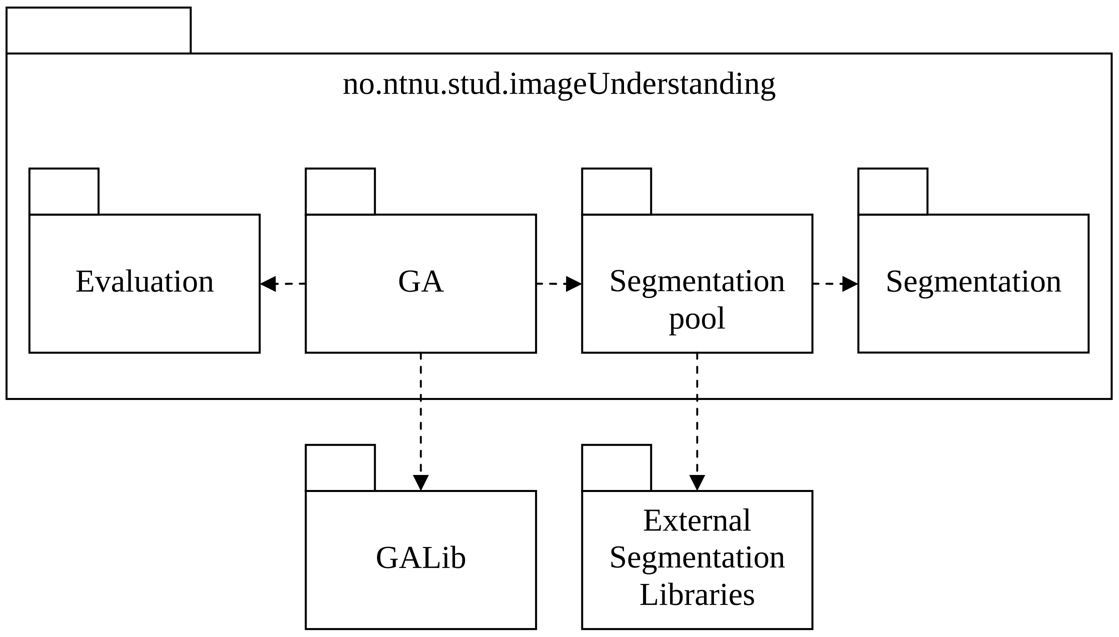

no.ntnu.stud.imageUnderstanding
Evaluation
GA
Segmentation pool
Segmentation
GALib
External Segmentation Libraries

Figure 5-5 Package diagram

5.2.3.1 imageUnderstanding

The name imageUnderstanding is kept from the system in "fordypningsprosjekt". It is the root package, and contains the whole system except the external libraries. The classes contained in this package are:

- App
- Algorithm
- ImageException
- NotColorImageException
- NotGreyImageException
- XImage

The App class is the start-up class for the system. This class contains the main method, but it is just a small class and contains little functionality.

The Algorithm class is a quite important class. This is the super-class for all the algorithm gene classes (the wrapper classes for the segmentation algorithms). This class makes it possible to put classes that inherits from this class into an chromosome. The Algorithm class has to be updated every time an algorithm is removed or added to the system. This is because of the random selection of algorithms. To make the random selection easy the algorithm class contains a list over the different algorithm gene classes. For a test program it is not a problem, but means that if algorithms are to be removed or added, some of the source code has to be changed. The actual selection is just a random number from 0 to the number of algorithms.

The NotColorImageException and NotGreyImageException classes are subclasses of the ImageException class. The purpose of these classes is to handle problems and avoid unexpected crashes. They are typically used when checking if the image is loaded before accessing the data, or when colour or grey images are needed.

The XImage class contains the image, and makes it easy to access the image data. It is a wrapper class over the classes BufferedImage and WritableRaster [48]. In addition to this it also has some other features. Some of these features are:

- Colour conversion from colour to grey scale or the other way around.
- Making clones of an image.
- Reading and writing of images to file.

The class also contain several different functions for accessing the image data.

5.2.3.2 imageUnderstanding.SegmentationPool

This package is for wrapping the segmentation algorithms. This makes a uniform appearance even when dealing with external libraries. There is one wrapper class for each algorithm. The wrapper classes used in the system are:

- EdgeBasedGene
- MaximumLikelihoodGene
- MeanShiftGene
- MultiThesholdGene
- SplitMergeGene
- WatershedGene

Besides wrapping the segmentation algorithms with almost the same names, they also contain the information about the attributes. For each segmentation algorithm attribute, these wrapper classes contain three numbers. The numbers represent:

- The present attribute value
- The minimum value for that attribute
- The maximum value for that attribute

The present attribute values are the only values that change during the running of the system. The others are set when making the wrapper classes. The classes also contain the name of the class for result purposes, and it takes care of any image conversions needed before running the algorithm.

5.2.3.3 imageUnderstanding.GA

The GA package is a wrapper class between the system and the external library GALib (Java Genetic Algorithm (JAR) library). The representation of the chromosomes is also here. The classes contained in this package are:

- ChromInterface
- ChromIU
- GAImageUnderstanding

The ChromInterface class is purely a wrapper class to encase the chromosomes in the GALib library.
The ChromIU class is the implementation of the top level of the structure of the chromosomes. It contains an array of algorithms and is responsible for running the functions in the Algorithm class. It also checks if it contains a certain algorithm if asked. This is for finding two similar algorithms for pairing purposes.

The GAImageUnderstanding class inherit the GALib main class, and is the core of the genetic algorithm. This class controls most of the genetic algorithm functions such as:

- Pairing
- Mutation
- Fitness evaluation
- Making new chromosomes
- Loading images

It only controls those tasks in the sense that is start the other classes responsible for those tasks. This class does not do the selection of chromosomes, or the keeping and sorting of chromosomes. This is done by the GALib main class.

5.2.3.4 imageUnderstanding.Evaluation

The Evaluation package is for calculating the fitness of the chromosomes. The classes in this package are:

- Evaluation
- SegmentContainer

The Evaluation class is the class where the fitness is calculated. The evaluation can be written like this in pseudo code:

```
Fitness = 0
For each reference segment do
    Calculate hit for each segment for evaluation
    Calculate miss for each segment for evaluation
    For each segment for evaluation find max value
        If hit > miss
            Value = hit - miss
        Else
            Value = 1
        End
    Fitness = Fitness + Max (Value)
End
```

This algorithm is quite simple but effective. It also leaves some decision up to the person manually segmenting the images on what should be valued. This is done by carefully selecting the segments in the image.

The SegmentContainer class is just a dummy class for keeping track of the segments when calculating hit and miss for each segment.

5.2.3.5 imageUnderstanding.Segmentation

This package contains the segmentation algorithms that are either translated or implemented for the system. Five of the segmentation algorithms are single class algorithms. They are:

- BlobColour
- EdgeBased
- MaximumLikelihoodSeg
- Multithreshold
- SplitMerge

BlobColour class is not actually a segmentation algorithm. It only finds connected regions with the same value and labels them. This is used for dividing the different segments before the fitness evaluation.

The EdgeBased class is a filter algorithm. This means that the actual processing of the image is done with filters. This algorithm uses two Sobel filters as shown in [Figure 5-8].

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

Figure 5-8 Sobel filter

The filters are run over the image and make two separate images. The values are then put together by using the function value $= \sqrt{value\_a^2 + value\_b^2}$.

The maximumLikelihoodSeg and SplitMerge classes are just translations from an external library.

The MiltiThresh class is an implementation that can take multiple thresholds, but it is still possible to take a normal threshold with this algorithm. There are two attributes to control the algorithm; seed and delta [Figure 5-9]. Seed is the value as in normal threshold algorithms. Delta controls how big the ranges of values to be joint. It will function as a normal threshold algorithm if the delta is large enough. A large enough delta will make the seed the only splitting point.

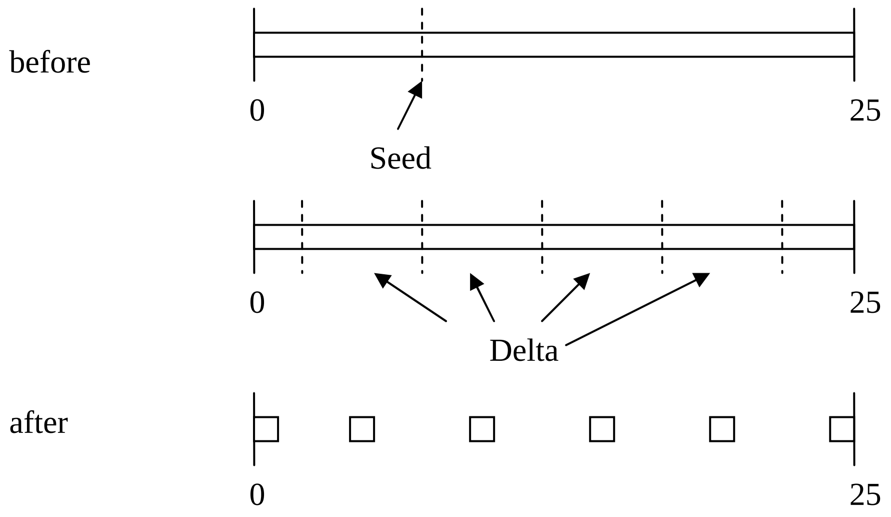


Figure 5-9 Multi threshold

The last algorithm in the segmentation package is the Watershed algorithm. The watershed algorithms consists of these four classes:

- Basin
- Link
- WatershedStructure
- Watershed

The Basin class contains the information about the basins in the Watershed algorithm. For example the centre coordinates that are used for deciding which basin a pixel belongs to.

The Link class is just a dummy class.

The WatershedStructure keep the preliminary information bout the pixels from the image.

The Watershed algorithm is the main class. First it does some smoothing and filling of the image. This is to reduce the amount of processing done by the algorithm. This also reduces the amount of memory needed for running the algorithm. The rest is closely based on the algorithm described in chapter two. The only part that is not discussed there is the way the algorithm decides which basin the pixel will be a part of. If there are two possible basins for a pixel, the distance from the pixel to the centres of the basins are calculated. The one with the shortest distance gets the pixel. This will favour small basin rather than large ones [Figure 5-10]. This will make the segment more equally large. There are several ways to determine where the pixel will join. Two other examples are joining the pixel with the largest one, or just by random.

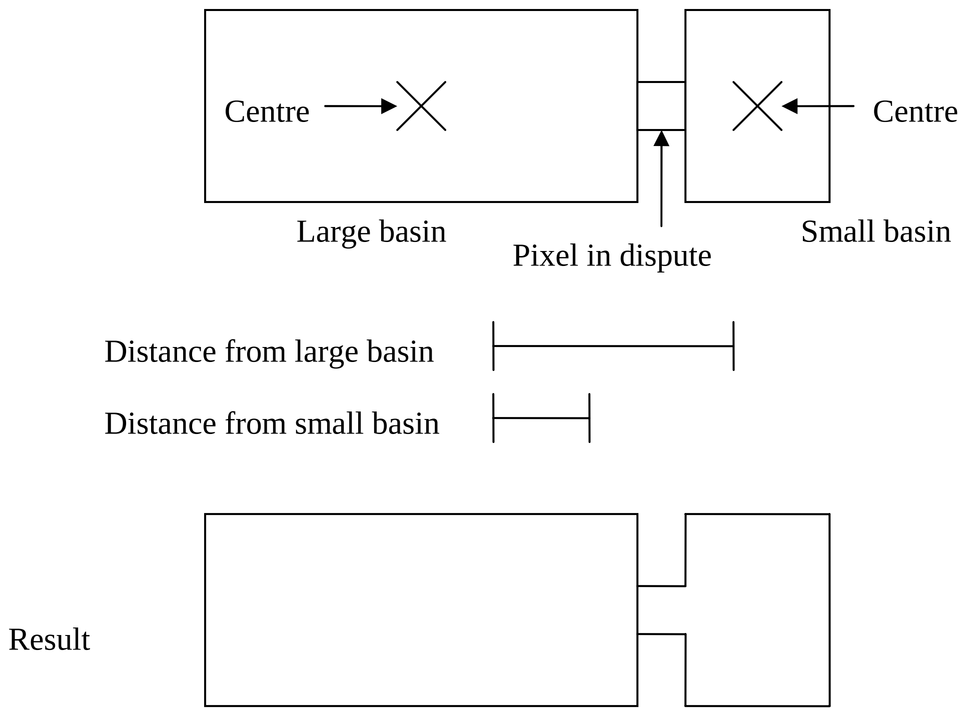


Figure 5-10 Dispute of pixel in watershed

6 Results

This chapter presents the input used and the results from the different tests of the program.

6.1 Input images

The program is run several times with different images. All the images are also manually segmented. The segmentation is chosen for the different images to challenge the program. The images used are:

- An ultrasound image
- An image of a lighthouse
- An image of a boy sitting by a river
- A Magnetic Resonance Image

Real size images of the original images are put in [Appendix B] and the real size segmented images in [Appendix C]. The ultrasound image has three different segments that are picked out [Figure 6-1]. It is 100 x 100 pixels large. The difficulty in this picture is the distortion.

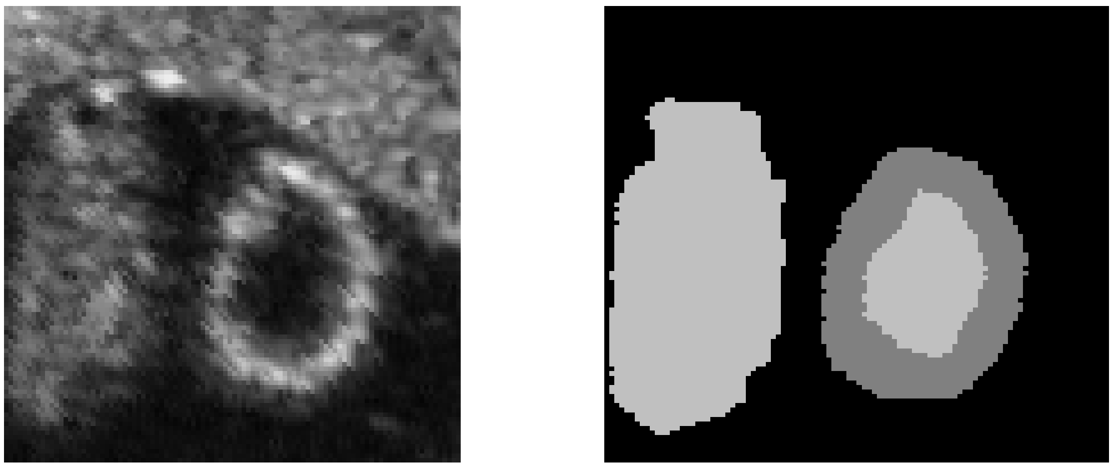


Figure 6-1 Ultrasound image

The image of the lighthouse has two segments that are picked out [Figure 6-1]. It is 240 x 168 pixels large. The difficulty with this picture is to make the whole lighthouse as one segment including the stairs.

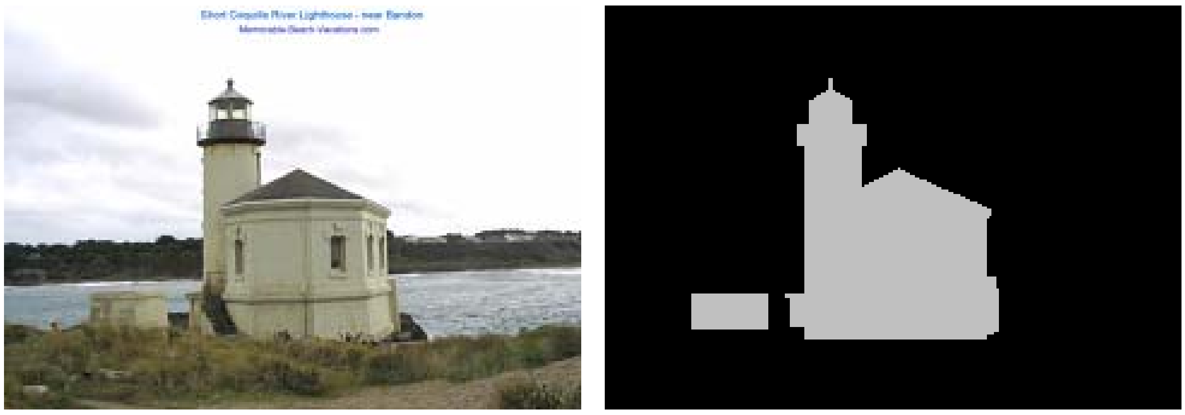


Figure 6-1 Lighthouse

The image of a boy sitting by a river has two segments that are picked out [Figure 6-2]. It is 442 x 338 pixels large. The difficulty with this picture is to get the whole boy, and at the same time segment out the stone he is sitting on. The boy is difficult because of his white collar. The stone is difficult because of the subtle edges to the surrounding areas. To segment out both segmentations at the same time is extremely difficult.

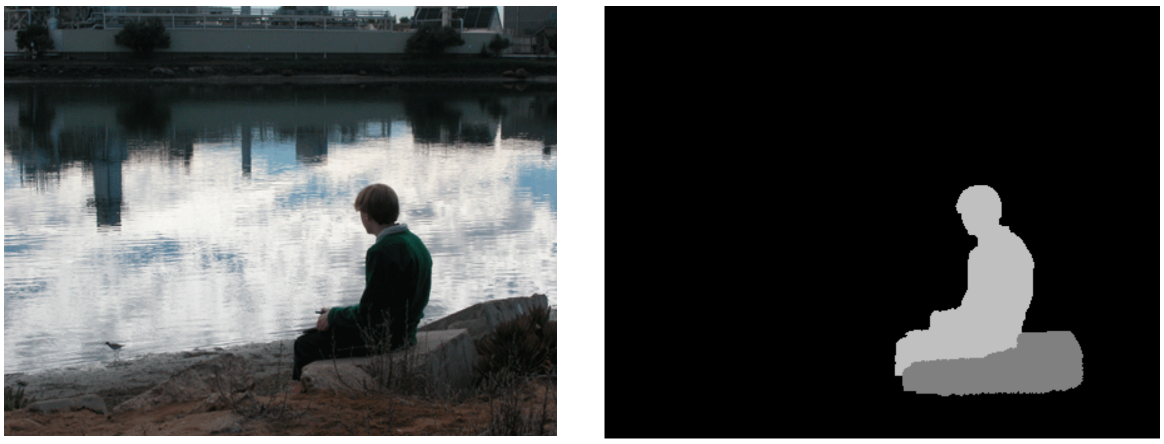


Figure 6-2 Boy sitting by the river

The Magnetic Resonance Image has two segments that are picked out [Figure 6-3]. It is 227 x 181 pixels large. The difficulty with this picture is the some what blurred edges around the segments. This makes it difficult not to merge with the surrounding areas.

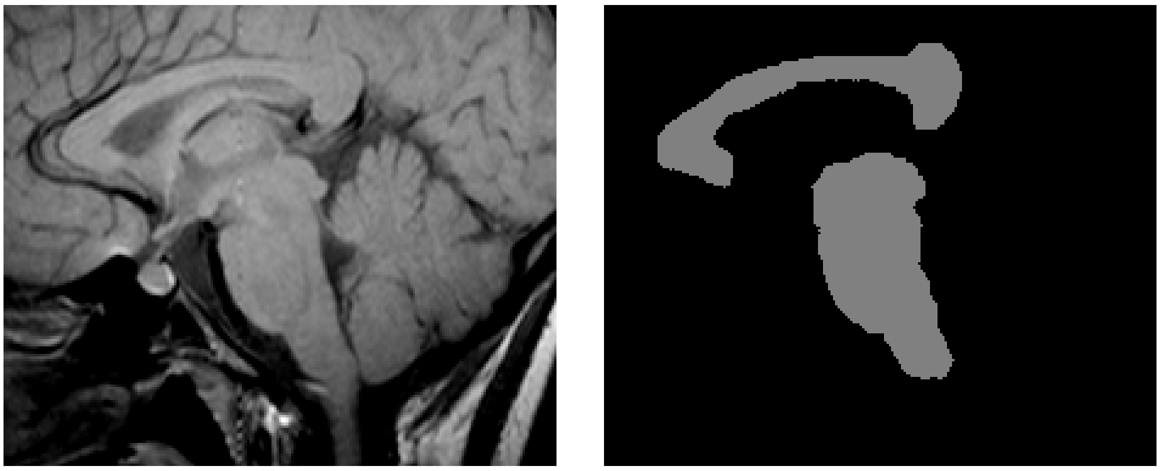


Figure 6-3 Magnetic Resonance Image

6.2 Genetic algorithm configuration

This chapter is about the setting of the parameters for the genetic algorithm.

6.2.1 Common parameters

To get some common criteria for the different tests, many of the parameters are the same during all the tests on the images. This is to minimise the bias in the results. The common parameters are:

- The probability for paring to occur (60 %)
- The probability of pairing (20 %)
- Probability of crossover when pairing (0.6 i.e. 60 %)
- Random selection chance for without regards for fitness (10 %)
- The probability mutating or pairing between chromosomes not algorithms (0.3 i.e. 30%)

These values may not be the best values, but a least they will give a more common starting point.

6.2.2 Variable parameters

Some parameters are changed to do different tests. These parameters are:

- Maximum number of algorithms when making new chromosomes
- Number of chromosomes in each generation
- The number of generation

If the maximum number of algorithms is set to one, there will only be algorithms that are one algorithm long. This is because after pairing chromosomes, both chromosomes have to have at least one algorithm. When it is set higher than one, the different chromosomes can grow larger than the initial maximum by pairing.

6.3 Tests

The tests are not large enough to say something definitive about the quality of the system, but can give some idea of it. Some samples of results are put in [Appendix D].

6.3.1 Test one

The first test of the systems is between single algorithms and sequences of algorithms. The parameters are set to:

- 20 generations
- 20 chromosomes per generation

For the single algorithms, each algorithm is run with a maximum chromosome length of 1. For the sequence with all the algorithms available, the maximum start length of the chromosomes is set to 3. The six first single algorithm searches are much more exhaustive because of the smaller search space. The results are in the table below [Table 6-1]:

Table 6-1 Results for test one

	Edge Based	Max. Like	Mean Shift	Multi thresh	Split Merge	Watershed	All
Mag. Res. Image	2814	2865	3228	2689	2412	1625	2846
Lighthouse	2120	3181	5814	3711	3967	3951	5805
Boy	5458	1296	6412	5164	5989	6042	7434
Ultrasound	1121	1543	2497	2230	1553	1264	2502
Sum	11513	8885	17951	13794	13921	12882	18587
Average	2878,25	2221,25	4487,75	3448,5	3480,25	3220,5	4646,75

The Mean Shift algorithm has a clear advantage on the other single algorithms. Only the one with all the algorithms and the possibility to make a sequence of algorithms has done better.

6.3.2 Test two

This is for finding the time used by the different algorithms. The system is run with each algorithm for each image. Every run consists of 20 segmentations including fitness evaluation. The results are in the table below [Table 6-2].

Table 6-2 Results of test two

	Edge Based	Max. Like	Mean Shift	Multi thresh	Split Merge	Watershed
Ultrasound image	00:20	00:21	02:33	00:19	00:32	00:42
Lighthouse	01:03	01:19	04:55	01:08	01:16	02:34
Boy by the river	03:07	03:34	18:30	02:58	03:04	10:52
M.R. Image	00:55	01:04	03:05	00:55	01:09	02:20

The slowest algorithm is the slowest for all the images, but also the best single algorithm. All the algorithms are responding to the images approximately the same way in used processing time. This is probably because of the size differences.

6.3.3 Test three

This is a running of the genetic algorithm with all the algorithms available, but with different number of chromosomes per generation. This time it is only for 10 generations, and on the ultrasound image. The results were [Table 6-3]:

Table 6-3 Results for test three

	10 chromosomes	30 chromosomes	50 chromosomes
Round 1	2319	2699	2662
Round 2	1539	1823	2673
Round 3	2325	2539	2274

The results show a tendency to get worse the fewer chromosomes there are as should be expected. Also the results are closer together for the “50 chromosome” tests than the others.

6.3.4 Sequences of algorithms

The different sequences from test one and three that gave significantly better results that the single algorithms are (attributes omitted):

• Lighthouse:	Mean Shift, Mean Shift, Watershed	Fitness = 7434
• Ultrasound Image:	Multi threshold, Split Merge, Mean Shift	Fitness = 2662
• Ultrasound Image:	Split Merge, Mean Shift	Fitness = 2673
• Ultrasound Image:	Mean Shift, Multi Threshold	Fitness = 2699

All the different sequences contains Mean Shift, but in different places. This means that there are many different locale maxima when dealing with such a genetic algorithm.

6.4 Discussion

When looking at the results, there are a couple of things with this system that should be done differently. The first is to do a tougher check on the segmentation methods. This is because Mean Shift algorithm was much better than the other segmentation algorithms. This may have been bad for the results, but even with this problem the sequencing algorithm were better than the single Mean Shift algorithm. Because of smaller search spaces, the single algorithms had an advantage over the sequence. This can be corrected if using a lot of time and chromosomes.

The other thing is the parameters for the genetic algorithm. More testing and tuning should be done before using the different segmentation algorithms. This should be done because it may give some better results for both the sequencing algorithm and for the other algorithms. It will also give some extra insight.

The rest of the system worked very well. The chromosomes functioned as expected and did their job well, and most of the design seems to be slow, but functional.

6.5 Further improvements

This is just a test program and has some missing features. The parts that have to be worked on to make this a solid program are the following:

- A new automatic fitness function
- Reduce processing time
- Consider parallel processing
- Re-evaluate segmentation algorithms

A way to automatically evaluate the fitness without manually segmenting each image is quite hard to make, but is a necessity if such a program was to segment images on its own. Reduction of the processing time can easily be achieved by changing the programming language, and also optimizing the different algorithms. The parallel processing shortens the time before an answer but not the processing time. Parallel processing can be done by spreading the chromosomes on the different computers. The re-evaluation of the segmentation algorithms will potentially give a more even algorithm base to work with, and maybe also pre-processing algorithms.

7 Conclusion

The fitness of the combined algorithm is good in comparison with the other algorithms. The improvements done by the multi algorithms scheme in comparison with the best single algorithm in the test are:

• Average improvement	3.54 %
• Maximum improvement	15.94 %

This show some improvement, but there is some “luck” in the picture when running genetic algorithms, so more testing is recommended.

The drawback with this genetic algorithm is the processing time. It takes a lot of time to run the algorithm, and it is not certain that it will come up with an answer that is optimal. Even with these problems it has potential to become a good tool in segmentation. Especially if there are put in adequate number of chromosomes.

The test system has shown that this kind of system is quite feasible and has the potential to give good results.

8 References

[1] Erlend Flaten and Lars Erik Hoel – " Bruk av kunnskap og erfaring i bildetolking" Prosjektoppgave i TDT4725 bildebehandling, fordypningsemne høsten 2004

[2] Matt Ottewill – "Colour & image modes"
http://www.planetoftunes.com/computer/colourmodes.html

[3] Luong Chi Mai - "Introduction to computer vision and image processing"
http://www.netnam.vn/unescocourse/computervision/computer.htm

[4] Michal Fano and Martin Polák - "The 3D Object Reconstruction from 2D Slices - Image Preprocessing"
http://www.cg.tuwien.ac.at/studentwork/CESCG/CESCG-2000/MFano/

[5] Ming Jiang - Digital Image Processing
http://ct.radiology.uiowa.edu/~jiangm/courses/dip/html/dip.html

[6] National Aeroneutics and space administration - Recursive Hierarchical Segmentation (RHSEG) Pre-processing Software
http://techtransfer.gsfc.nasa.gov/RHSEG/index.html#benefits

[7] Milan Sonka, Vaclav Hlavac and Roger Boyle - "Image Processing Analysis and Machine Vision" 2. Edition, PWS Publishing 1999,
ISBN 0-534-95393-X

[8] Ahinn-Ying and Kual-Zheng Lee – "Design and Analysis of an Efficient Evolutionary Image Segmetation Algorithm"

[9] Ahmed E. Ibrahim - "An Intelligent Framwork for Image Understanding"
http://www.engr.uconn.edu/~ibrahim/publications/image.html

[10] Jos B.T.M. Roerdink and Arnold Meijster - "The Watershed Transform: Definitions, Algorithms and Parallelization Strategies"
http://www.cs.rug.nl/~roe/publications/parwshed.pdf

[11] Tom McReynolds and David Blythe – "Advanced Graphics Programming Techniques Using OpenGL"
http://www.opengl.org/resources/tutorials/sig99/advanced99/notes/notes.html

[12] G Davidson, K Ouchi,G Saito, N Ishitsuka,K Mohri and S Uratsuka - "Performance Evaluation of Maximum Likelihood SAR Segmentation for Multi-Temporal Rice Crop Mapping"
http://www.radarworks.com/Papers/R2002RiceFinal.ppt

[13] Changjiang Yang, Ramani Duraiswami, Nail A. Gumerov and Larry Davis - "Improved Fast Gauss Transform and Efficient Kernel Density Estimation"
http://ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=1238383

[14] Richard E. Blake – Forelesningsnotater, TDT4265 Datasyn
http://rover.idi.ntnu.no/~cv/FEATURES1AAA.shtml

[15] Daniel Brännström,Daniel Bäck and Johnny Widerlund – "OCR - Optical Character Recognition"
http://www.dtek.chalmers.se/~d95danb/ocr/

[16] George F Luger – "Arificial Intelligence" 4. Edition, Pearson 2002,
ISBN 0-201-64866-0

[17] Pål Magnus Joyce og Sturla Johannessen - "Using knowledge and expierience in image understanding"

[18] Wikipedia, the free encyclopedia - "Empirical knowledge"
http://en.wikipedia.org/wiki/Empirical_knowledge

[19] WikiWikiWeb – "Domain Knowledge"
http://c2.com/cgi/wiki?DomainKnowledge

[20] Bir Bhanu, Sungkee and John Ming - "Adaptive Image Segmentation Using a Genetic Algorithm"
http://www.vislab.ucr.edu/PUBLICATIONS/JOURNALS/IEEE/ga-smc-95.pdf

[21] Petra Perner – "Why Case-Based Reasoning Is Attractive for Image Interpretation"
www.ibai-institut.de/document/iccbr01.pdf

[22] Morten Grimnes and Agnar Aamodt - "A two layer case-based reasoning architecture for medical image understanding"
www.idi.ntnu.no/~agnar/yarc/papers/ewcbr-96.pdf

[23] M. Colilla, C.J. Fernández and E. Ruiz-Hitzky - "Case Based Reasoning (CBR) for Multicomponent Analysis Using Sensor Arrays: Application to Water Quality Evaluation"
http://www.rsc.org/suppdata/AN/b2/b205590b/B205590B.pdf

[24] Petra Perner – "Ultra Sonic Image Segmentation and Interpretation based on CBR"
www.ibai-institut.de/document/eccbr00.pdf

[25] Tom M. Mitchell – "Machine Learning", McGraw-Hill 1997,
ISBN 0-07-115467-1

[26] Haym Hirsh – "Trends and Controversies Genetic programming"
http://www.cs.nott.ac.uk/~gxk/courses/g5baim/papers/gp-001.pdf

[27] Charles Darwin – "The Origin of Spieces"
http://www.talkorigins.org/faqs/origin.html

[28] Peter J. Angeline – "Genetic programming: a current snapshot"
http://www.cs.plu.edu/courses/csce330/arts/gprog1.pdf

[29] Jason Lohn, Greg Larchev and Ronald DeMara – "A Genetic Representation for Evolutionary Fault Recovery in Virtex FPGAs"
http://ic.arc.nasa.gov/projects/esg/publications/lohn_papers/lohn_ices03.pdf

[30] Laura Barbulescu, Jean-Paul Watson, and L. Darrell Whitley – "Dynamic Representations and Escaping Local Optima: Improving Genetic Algorithms and Local Search"
http://www.cs.colostate.edu/~genitor/2000/aaai00.pdf

[31] Sushil J. Louis – "Solving Similar Problems using Genetic Algorithms and Case-Based Memory"
http://www.cs.unr.edu/~sushil/papers/conference/papers/icga97/icga97_2/icga97_2.html

[32] Akihiko Uchibori and Noboru Endou – "Basic Properties of Genetic Algorithm"
http://markun.cs.shinshu-u.ac.jp/Mirror/mizar.org/JFM/pdf/genealg1.pdf

[33] Matthew Boutell, Christopher Brown and Jiebo Luo – "Review of the State of the Art in Semantic Scene Classification"
http://www.cs.rochester.edu/research/vision/pubs/2002/Boutell_Review_Semantic_Scene_Class_TR_2002/Boutell_Review_Semantic_Scene_Class_TR_2002.pdf

[34] Matthew R. Boutell, Jiebo Luo, Xipeng Shen, Christopher M. Brown – "Learning multi-label scene classification"
http://www.cs.rochester.edu/~boutell/ publications/boutell04PRmultilabel.pdf

[35] Matthew Boutell and Jiebo Luo - "Beyond Pixels: Exploiting Camera Metadata for Photo Classification"
http://www.cs.rochester.edu/~boutell/publications/boutellIUPR.pdf

[36] Xia Yong, Dagan Feng and Zhao Rongchun – "Optimal Selection of Image Segmentation Algorithms Based on Preformance Prediction"
http://crpit.com/confpapers/CRPITV36Yong.pdf

[37] Michael Goldbaum, Saied Moezzi, Adam Taylor, Shankar Chatterjee, Jeff Boyd, Edward Hunter, and Ramesh Jain – "Automated diagnosis and image understanding with object extraction, object classification, and inferencing in retinal images"
http://people.brandeis.edu/~altaylor/icip-1996-analysis.pdf

[38] Nilton Correia da Silva and Antônio Nuno de Castro Santa Rosa – "Estimative of SOM Learning Parameters Using Genetic Algorithms"
http://www.cic.unb.br/docentes/nuno/artigos/SOM_AG_2002.pdf

[39] Melanie Aurnhammer and Klaus D. Tönnies - "The application of genetic algorithms in structural seismic image interpretation"
http://isgwww.cs.uni-magdeburg.de/bv/pub/pdf/TR-ISGBV-02-02.pdf

[40] M. Aurnhammer, K. D. Tönnies and R. Mayoral. - "A genetic algorithm for constrained seismic horizon correlation."
http://wwwisg.cs.uni-magdeburg.de/bv/pub/pdf/cvprip_neu.pdf

[41] Gianluca Pignalberi, Rita Cucchiara, Luigi Cinque and Stefano Levialdi –" Tuning Range Image Segmentation by Genetic Algorithm"
http://imagelab.ing.unimo.it/pubblicazioni/pubblicazioni/gecsp130.pdf

[42] Bir Bhanu, Sungkee Lee and John Ming – "Adaptive Image Segmentation Using a Genetic Algorithm"
http://www.vislab.ucr.edu/PUBLICATIONS/JOURNALS/IEEE/ga-smc-95.pdf

[43] Kresimir Simunic and Sven Loncaric – "A Genetic Search-based Partial Image Matching"
http://ipg.zesoi.fer.hr/papers/icips98.pdf

[44] Eclipse Foundation – Eclipse
http://www.eclipse.org/

[45] The KUIM Image Processing System
http://www.ittc.ku.edu/~jgauch/research/kuim/html/

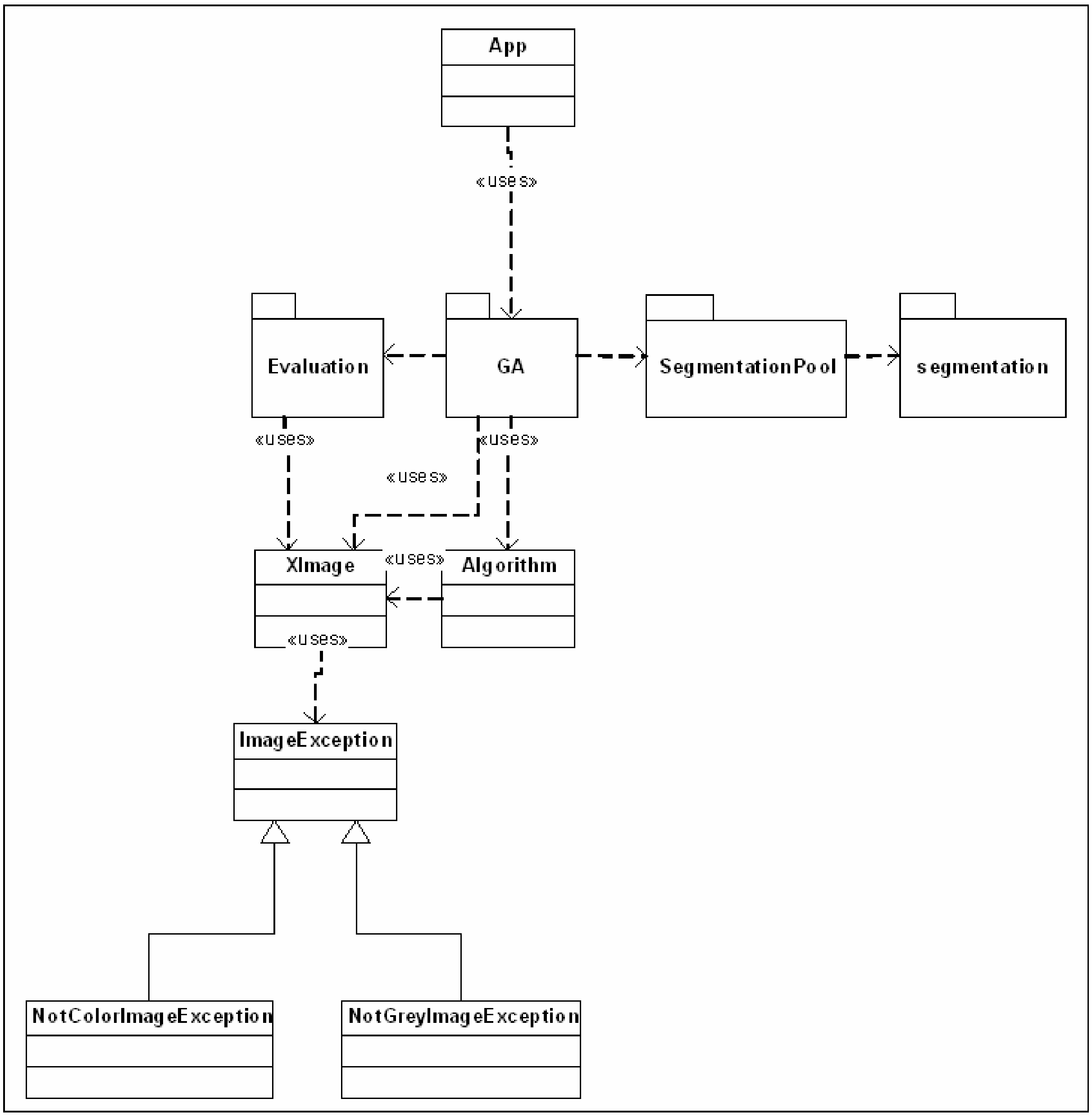
[46] jEDISON Java Package
http://sourceforge.net/projects/ebla/

[47] GALib, Java Genetic Algorithm (JAR) library
http://sourceforge.net/projects/java-galib/

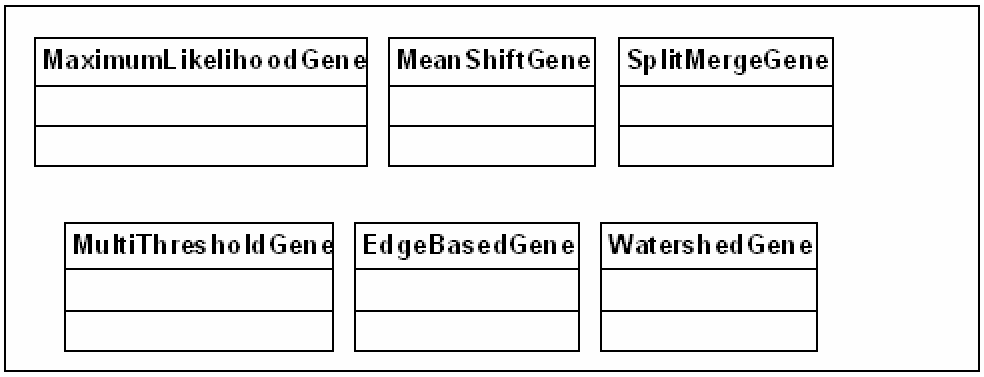
[48] JavaTM 2 Platform, Standard Edition, v 1.4.2 - API Specification
http://java.sun.com/j2se/1.4.2/docs/api/

Appendix A

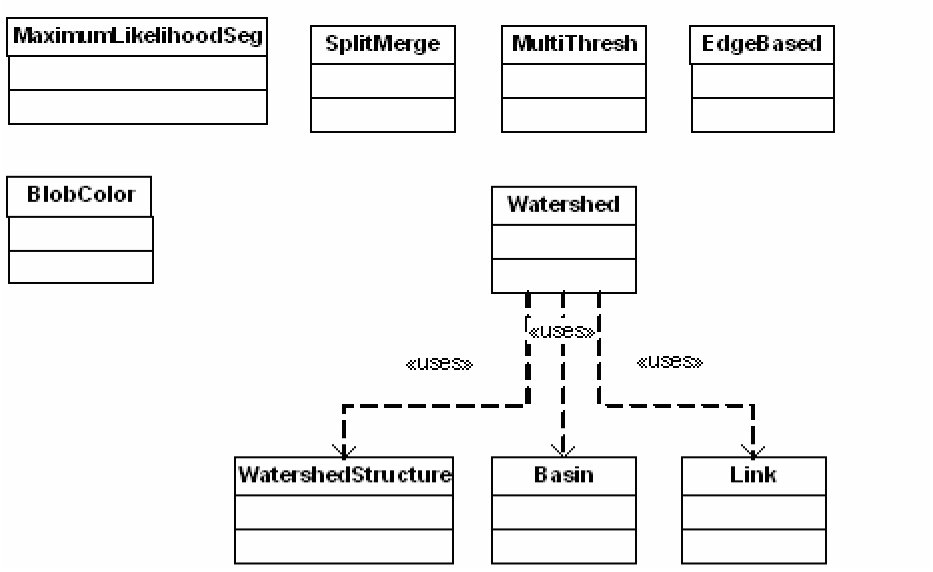
This appendix contain the class diagram for the different packages in the system [A - 1] [A - 2] [A - 3] [A – 4] [A - 5] and a complete class diagram for the whole system [A - 6].



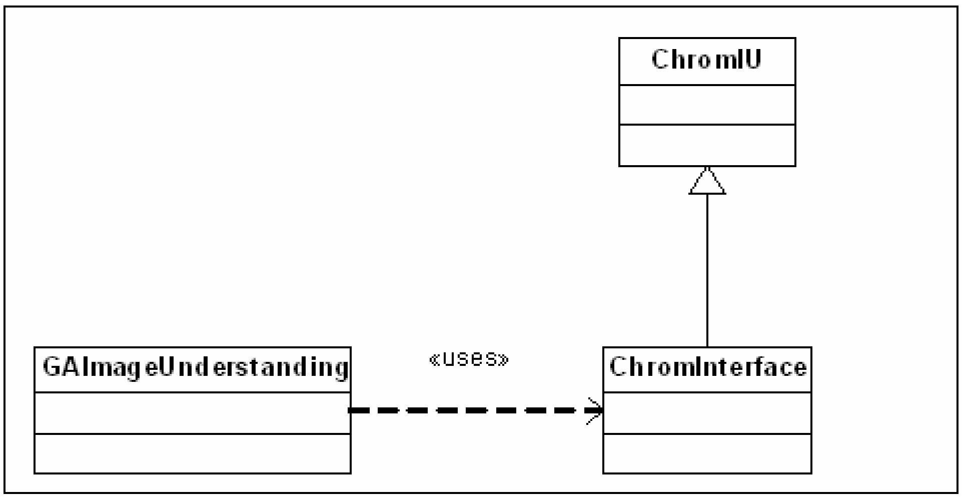
A - 1 ImageUnderstanding package



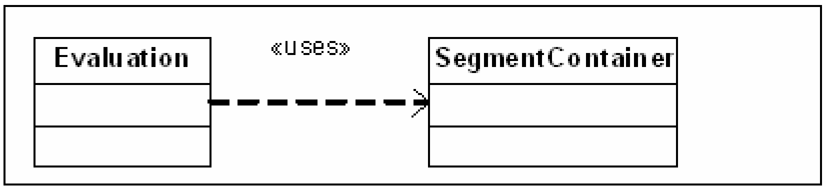
A - 2 SegmentationPool package



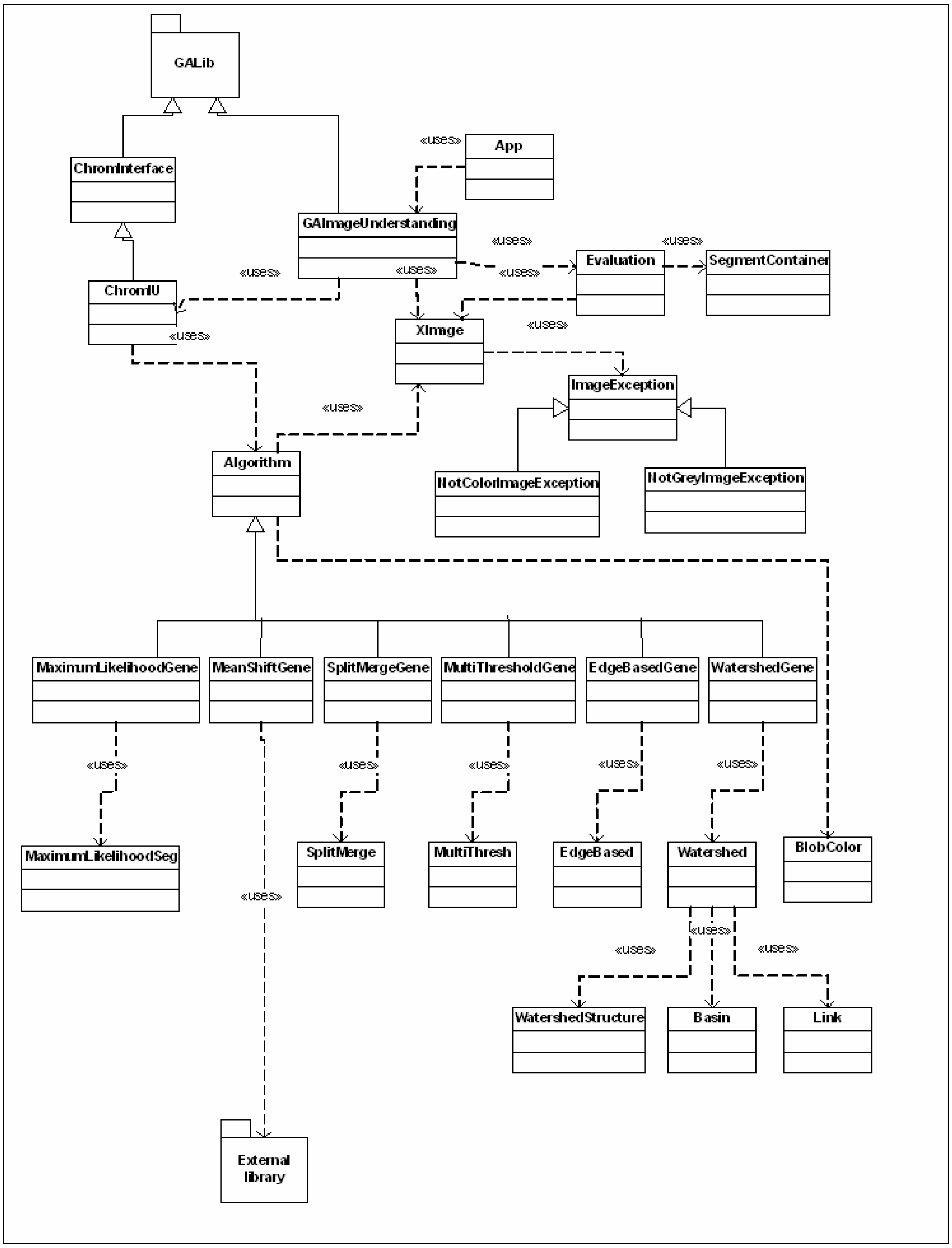
A - 3 Segmentation package



A - 4 GA package



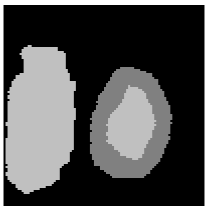
A - 5 Evaluation package

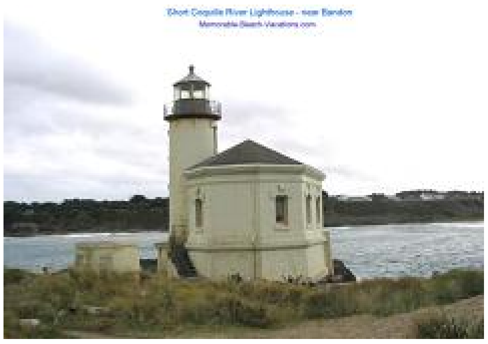


A - 6 complete class diagram

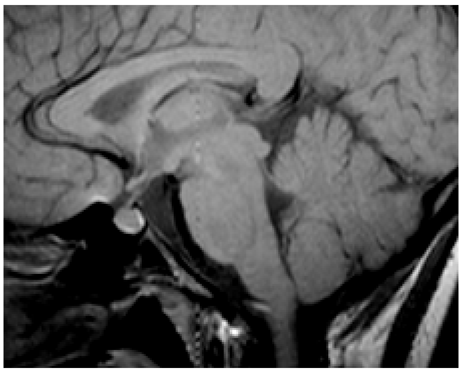
Appendix B

This appendix contains the not segmented images in the original sizes. These images are [B - 1] [B - 2] [B - 3] [B - 4].


B - 1 Ultrasound image

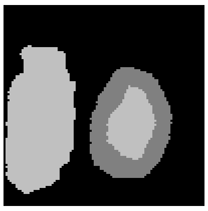

Short Coquille River Lighthouse - near Bandon
Memorable-Beach-Vacations.com
B - 2 Lighthouse

B - 3 Boy sitting by a river

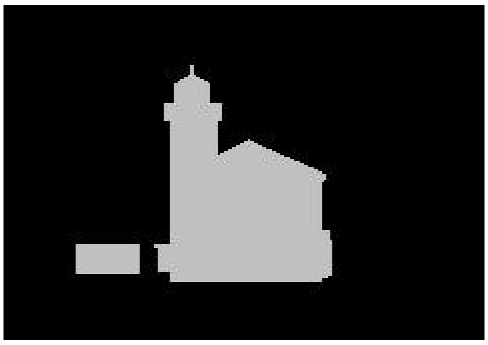

B - 4 Magnetic Resonance Image

Appendix C

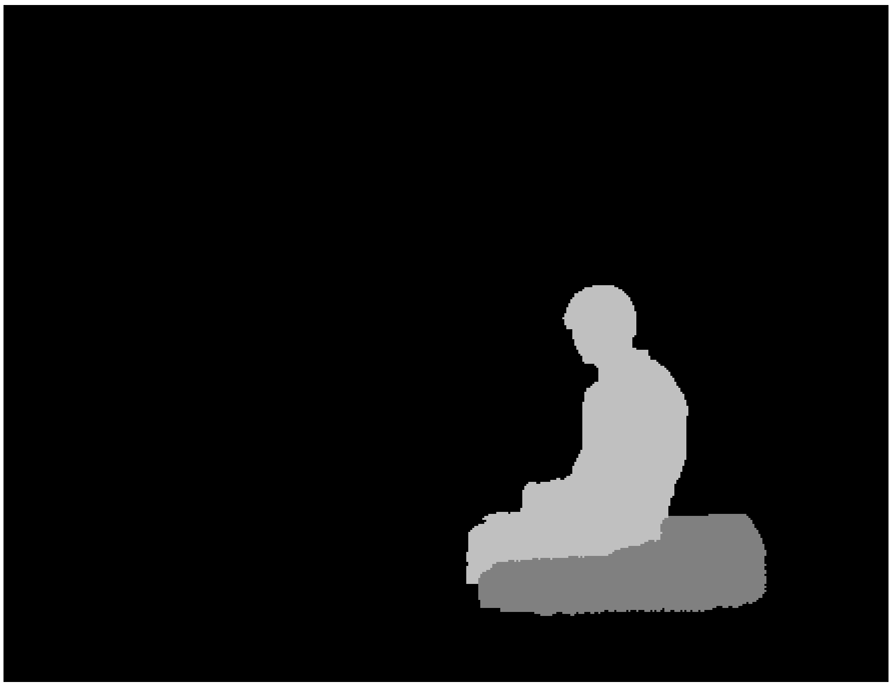
This appendix contains the manually segmented images in the original sizes. These images are [C - 1] [C - 2] [C - 3] [C - 4].



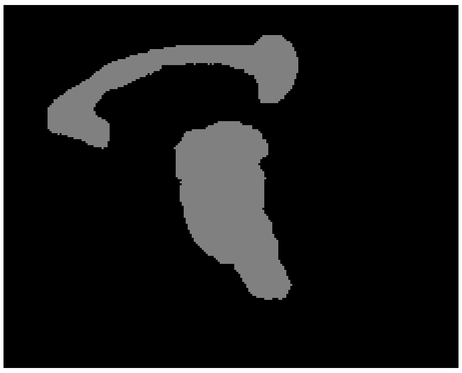
C - 1 Ultrasound image



C - 2 Lighthouse



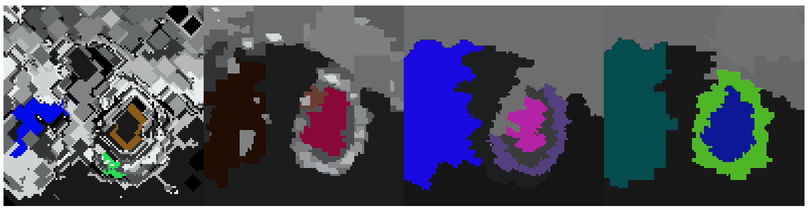
C - 3 Boy sitting by a river



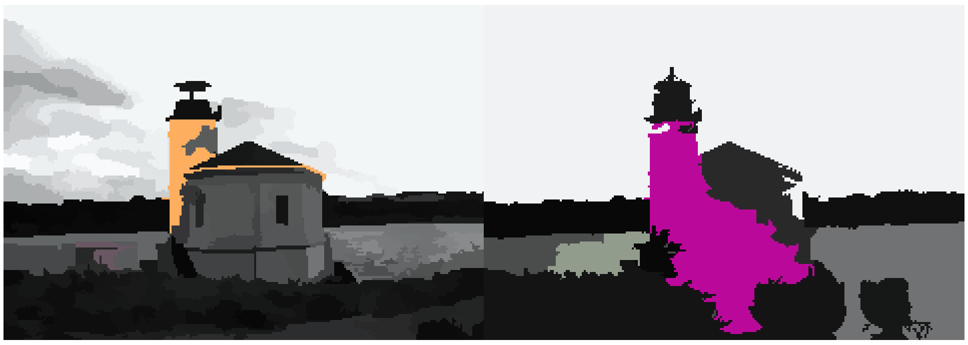
C - 4 Magnetic Resonance Image

Appendix D

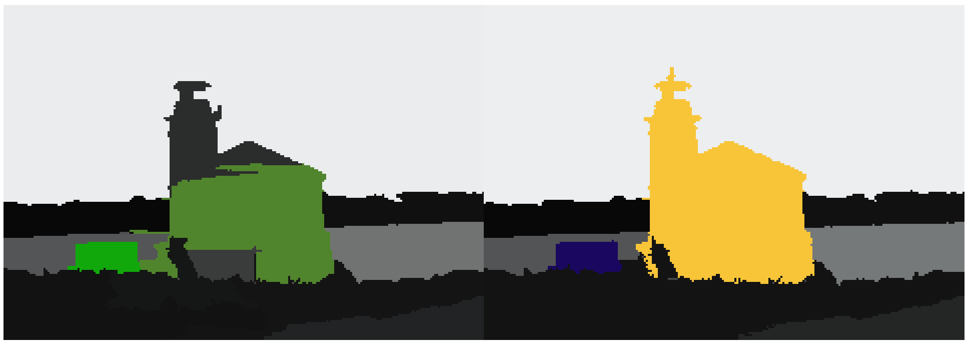
This appendix contains some examples of the different segmentations of the pictures. They are posted from bad to good for each image [D - 1] [D - 2] [D - 3] [D - 4] [D - 5] [D - 6] [D - 7] [D - 8] [D - 9].



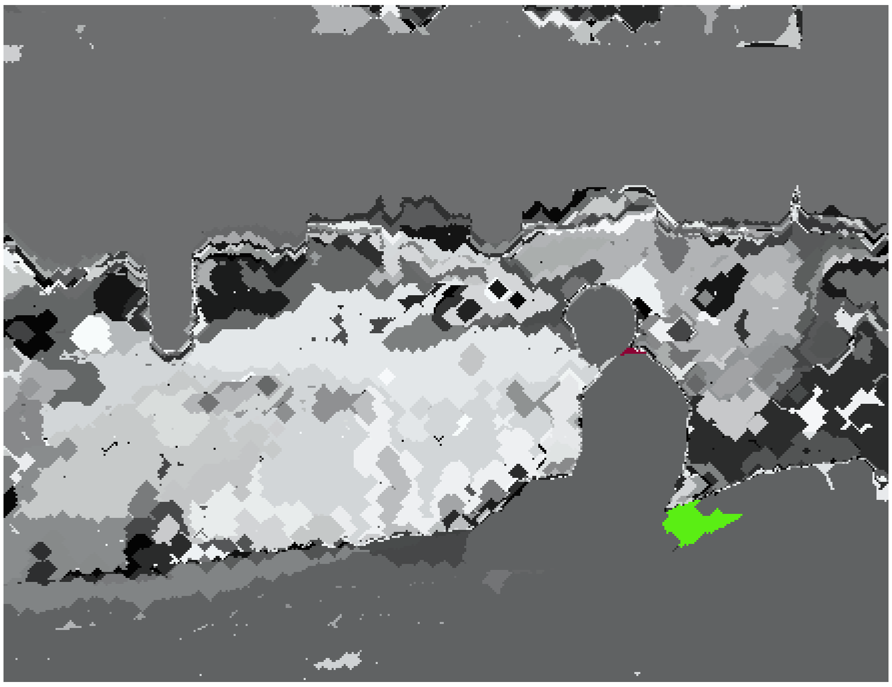
D - 1 Ultrasound image



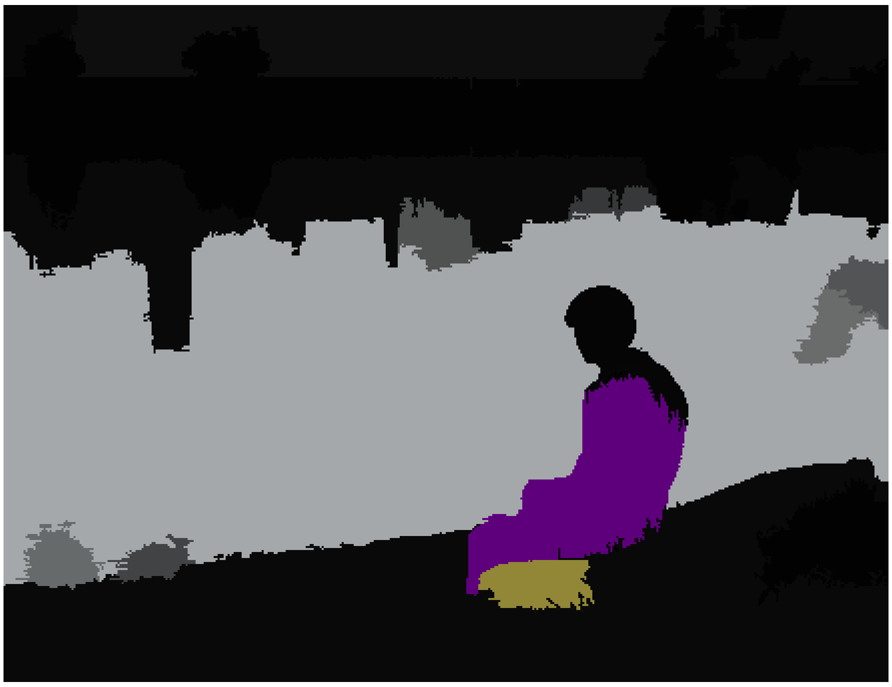
D - 2 Lighthouse1

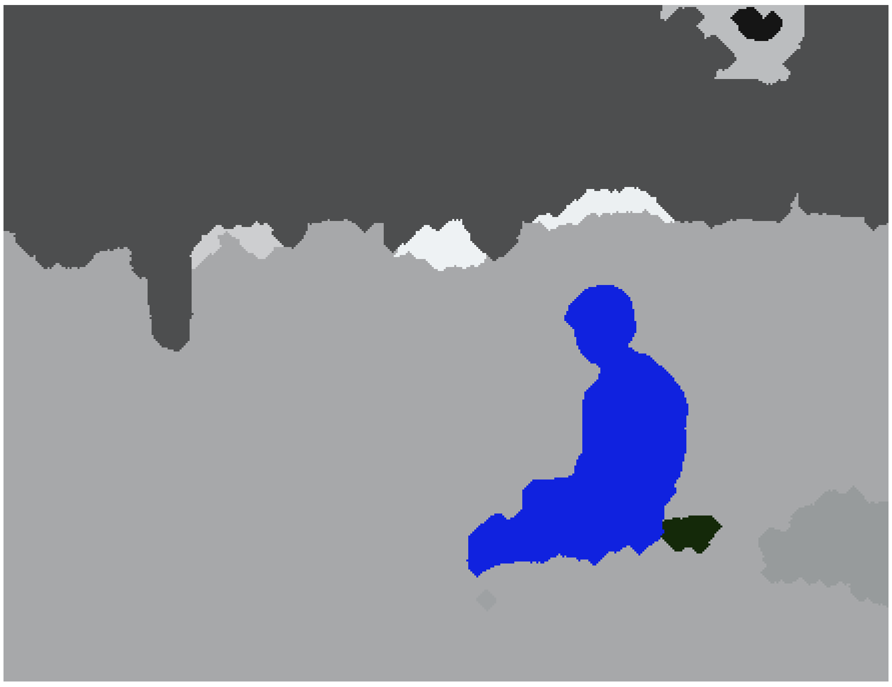


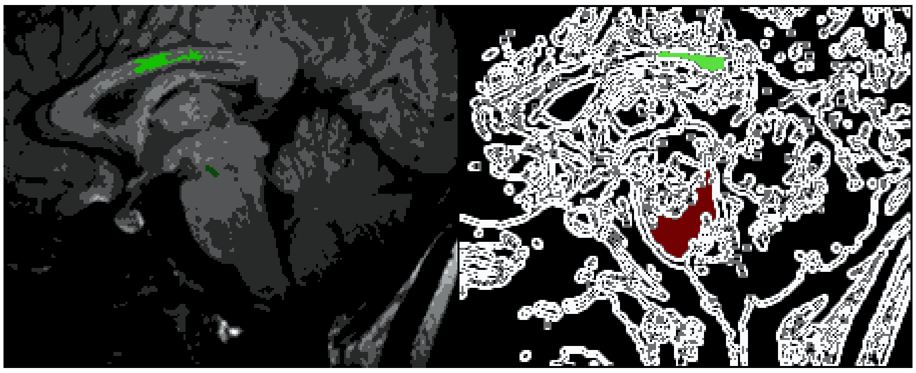
D - 3 Lighthouse2


D - 4 Boy sitting by a river1

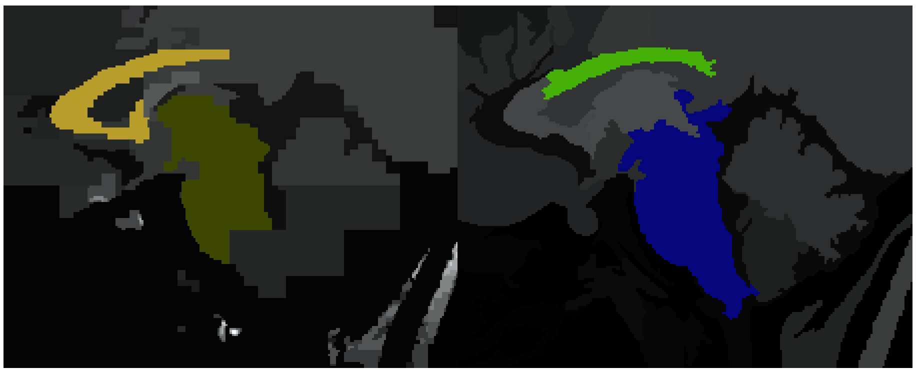
D - 5 Boy sitting by a river2


D - 6 Boy sitting by a river3


D - 7 Boy sitting by a river4



D - 8 Magnetic Resonance Image 1



D - 9 Magnetic Resonance Image 2