Optimization Algorithms for Plane Selection in Interactive 3D Image Segmentation

Jonas van Marrewijk[1]
Supervisor: Nicolas Chaves de Plaza[1]**, Klaus Hildebrandt**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Jonas van Marrewijk
Final project course: CSE3000 Research Project
Thesis committee: Klaus Hildebrandt, Nicolas Chaves de Plaza, Thomas Abeel

Abstract. Segmentation of 3D medical images is useful for various medical tasks. However, fully automated segmentation lacks the accuracy required for medical purposes while manual segmentation is too time-consuming. Therefore, an active learning method can be used to generate an accurate segmentation using less user input. The active learning pipeline consists of automatic 3D segmentation, generation of a 3D uncertainty field and an optimization algorithm finding the optimal plane in the uncertainty field. This plane can then be shown to a user to be labeled so that the segmentation can be improved. This process is repeated until the user is satisfied with the output. If the plane is chosen so that it contains more errors, then less user input will be needed. This paper focusses on evaluating different optimization algorithms in the context of this pipeline. The three algorithms that are evaluated are particle swarm optimization, gradient descent and L-BFGS-B. The results of the evaluation show that particle swarm optimization converges the quickest, but to lower values than gradient descent. Gradient descent converges slowly, but to high values. L-BFGS-B converges quickly to values that are as high as those from gradient descent. Therefore, using L-BFGS-B in the pipeline instead of gradient descent will decrease the runtime of the pipeline. Using particle swarm optimization will decrease the runtime even further, but at the cost of requiring more user input to obtain a segmentation of acceptable quality.

1 Introduction

Segmentation of medical 3D images is useful for various tasks, such as computer assisted diagnosis and visual augmentation. Segmentation of medical 3D images means labeling voxels within a volume, indicating whether they are or are not part of an organ. However, manually annotating 3D images is a time-consuming task and is often deemed infeasible for many purposes. [3] In addition, fully automated systems are usually not deemed accurate enough for medical purposes.

A way to segment the images interactively is described in [5]. An image is first segmented automatically. After that an uncertainty field is generated, which estimates which parts of the segmentation are likely to contain an error. Then, the plane which is most likely to contain errors is shown to the user, so that they can label it manually. Using this information, the 3D image is then segmented automatically again. This process repeats until the user is satisfied with the segmentation.

This paper focuses on finding the plane that is most likely to contain errors. This is done by running an optimization algorithm to find a plane in the uncertainty field so that the sum of the uncertainty values is maximized. The idea behind this is that because of this, slices that contain more errors will be presented to the user, resulting in a decrease in required amount of user input. While [5] finds that using gradient descent results in a significant improvement when compared to random slice selection, other optimization algorithms are not considered.

The goal of this paper is to find out which optimization algorithm is best suited for finding the optimal slice to show to the user. To do this, different optimization algorithms are implemented and compared to each other in terms of iterations needed. These algorithms are particle swarm optimization, gradient descent as described in [5] and L-

BFGS-B. The three algorithms use zeroth order, first order and second order derivatives respectively, with zeroth order derivatives meaning it only uses the function value.

2 Problem Description

The problem this paper tries to solve is finding an optimal plane in an uncertainty field. Section 2.1 explains the context behind this problem and what an uncertainty field is, then section 2.2 explains the problem itself.

2.1 General Pipeline

The pipeline consists of four parts: segmentation, uncertainty field calculation, slice selection and user input. This is based on the pipeline described in [5] and is illustrated in figure 1.

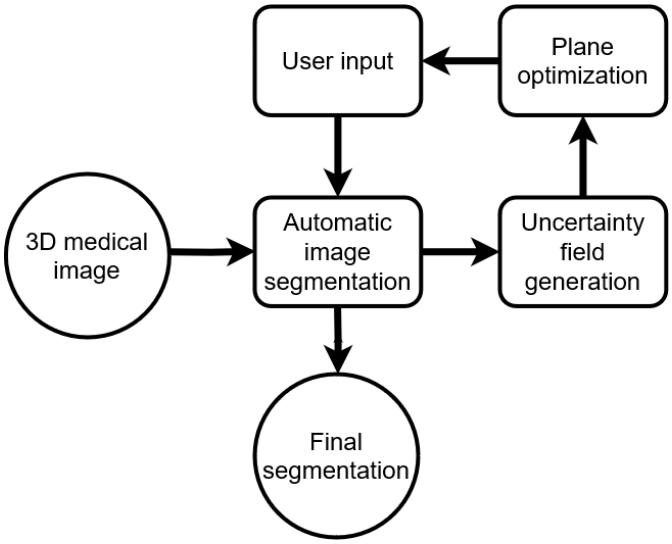

Fig. 1. Diagram showing the active learning pipeline, with its four steps as well as its input and output.

The first part of the pipeline is the automatic image segmentation. The segmentation takes as input a three-dimensional array representing an image, such as a CT scan. In addition, it can take seed points as input, which are points where the user has manually labeled the image. These seed points are represented as a three-dimensional binary array of the same dimensions as the input image. A segmentation is then generated, which is 1 at points that are part of the organ and 0 at points which are not part of the organ.

The uncertainty field is a three-dimensional array with the same dimensions as the input image and values between 0 and 1. The higher the uncertainty value at a certain

point, the less certain the pipeline is about the segmentation at that point. The uncertainty field is calculated as a weighted sum of different terms.

After that, the plane most likely to contain errors is found in the plane optimization step. This plane is shown to the user to be labelled manually. The data from the manual labeling is then used to run the automatic image segmentation again. By running this pipeline multiple times, the resulting segmentation is improved iteratively. This process repeats until the user is satisfied with the segmentation.

2.2 Plane Optimization

The goal of the plane optimization is to find the plane of highest uncertainty. That is, the plane whose points contain the highest uncertainty when summed up is found. This plane can be positioned anywhere in the uncertainty field and be rotated in any way.

The plane is parameterized by one three-dimensional point within the uncertainty field and a normal vector. It is then defined as

$$f_p(u, v) = p_P + ua + vb, \tag{1}$$

where p_P is the reference point, u and v are scalars, and a and b are orthonormal vectors perpendicular to n_P, the normal vector of the plane.

The total uncertainty U_P in a plane is then defined as

$$U_P = \int_\infty \int_\infty U(f_P(u, v))dudv, \tag{2}$$

where $U(f_P(u, v))$ is the uncertainty at a point in the plane.

3 Method

Three optimization algorithms are considered. First of all, particle swarm optimization uses only the uncertainty values itself, without using any derivatives. Then, gradient descent is considered, which uses the first derivative. Finally, L-BFGS-B is used, which approximates the second derivative.

3.1 Particle Swarm Optimization

Particle swarm optimization starts with a number of particles scattered randomly around the solution space. Every iteration, each particle's velocity is calculated using the following formula:

$$v_i^{t+1} = \omega v_i^t + c_1 r_1 (p_{best_i}^t - x_i^t) + c_2 r_2 (g_{best}^t - x_i^t) \tag{3}$$

and its position is then updated using:

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{4}$$

In these equations, r_1 and r_2 are random variables between 0 and 1, while ω, c_1 and c_2 are constants. p_{best}^t is the best position the particle itself has been in, while g_{best}^t is the

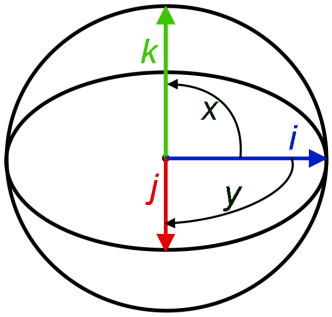

Fig. 2. Diagram showing rotation of unit vector in 3D space using xy Euler angles. Here $i = (1, 0, 0)$, $j = (0, 1, 0)$ and $k = (0, 0, 1)$ are the three standard basis vectors, while x and y are the rotation values.

global best, which is the best position any particle has been in. Both of these values are updated after each iteration. [2]

For finding an optimal plane, we consider x_i to be a five-dimensional vector. Its first three elements are the point in 3D space. The last two elements are the normal's xy Euler angle rotations in degrees. This rotation is applied to the vector $(1, 0, 0)$ to calculate the 3D normal, which allows it to cover any point on the unit sphere, as shown in 2. Since it is a 3D normal, equation 2 can be used to calculate the uncertainty.

3.2 Gradient Descent

The implementation of gradient descent is based on [5], which describes the gradients of the reference point and the normal of the plane in the following two formulas. Here, a plane is defined as $f_p(u, v) = p_P + ua + vb$, as described in equation 1. The gradient of the reference point is:

$$\nabla_{p_P} U_P = \int_\infty \int_\infty \nabla_x U(f_P(u, v)) du dv \tag{5}$$

and the gradient of the plane's normal is

$$\nabla_{n_P} U_P = \int_\infty \int_\infty (u J^T_{a,n_P} + v J^T_{b,n_P}) \nabla_x U(f_P(u, v)) du dv \tag{6}$$

where J_{a,n_P} is used to denote the Jacobian of a with respect to n_P:

$$J^T_{a,n_P} = \begin{bmatrix} -n_x & -n_y & -n_z \\ 0 & 0 & 0 \end{bmatrix} \tag{7}$$

and

$$J_{b,n_P}^T = \begin{bmatrix} 0 & 0 & 0 \\ -n_x & -n_y & -n_z \end{bmatrix}, \tag{8}$$

where n_x, n_y and n_z are the x, y and z components of the normal vector, respectively.

After each iteration, the normal vector is normalized to make sure it has a length of 1 after adding the gradient to it.

Since gradient descent is likely to converge to a local minimum, it is randomly initialized multiple times. This means gradient descent is started at a random point multiple times and the overall result is the maximum of the results of the initializations.

3.3 L-BFGS-B

L-BFGS-B uses an approximation of the inverted Hessian matrix to find roots of a function [1]. This approximation is improved every iteration. This helps it approximate Newton's Method, which iteratively improves a solution using the second derivative. Like gradient descent, this function tends to converge to local optima. Therefore, it is randomly initialized multiple times.

The search space is defined as being five-dimensional, identical to the search space of particle swarm optimization.

In this paper, the L-BFGS-B implementation from scipy's optimize module is used[1], which is based on [6].

4 Experimental Setup and Results

We verify the effectiveness of all three optimization algorithms experimentally and compare their performance. Section 4.1 describes the implementation details of the algorithms, then section 4.2 evaluates the algorithms qualitatively and section 4.3 evaluates the algorithms quantitatively.

4.1 Implementation Details

The step sizes of gradient descent and L-BFGS-B, as well as the ω, c_1 and c_2 of the particle swarm optimization were chosen empirically by running the algorithms for different values and comparing the results. The chosen step sizes for gradient descent are 0.05 for the point gradient and 0.00001 for the normal gradient. The values chosen for particle swarm optimization are $\omega = 0.8$, $c_1 = 0.2$ and $c_2 = 0.2$.

4.2 Qualitative Analysis

To evaluate the different optimization algorithms qualitatively, a simple uncertainty field is generated artificially. The three different optimization algorithms are then run on this artificial uncertainty fields, allowing us to confirm whether the algorithms tend to converge to a maximum.

[1] docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_l_bfgs_b.html

Experimental Setup The artificial uncertainty field is generated by placing three-dimensional Gaussians at three different points. These points can be chosen arbitrarily. The final uncertainty field is the sum of these three Gaussians.

The optimal plane in this uncertainty field is the plane that intersects all three centers of the Gaussians. To verify that the plane tends to converge to the optimum, all three optimization algorithms are run on the artificial uncertainty field, and the resulting planes are visually compared with the optimum.

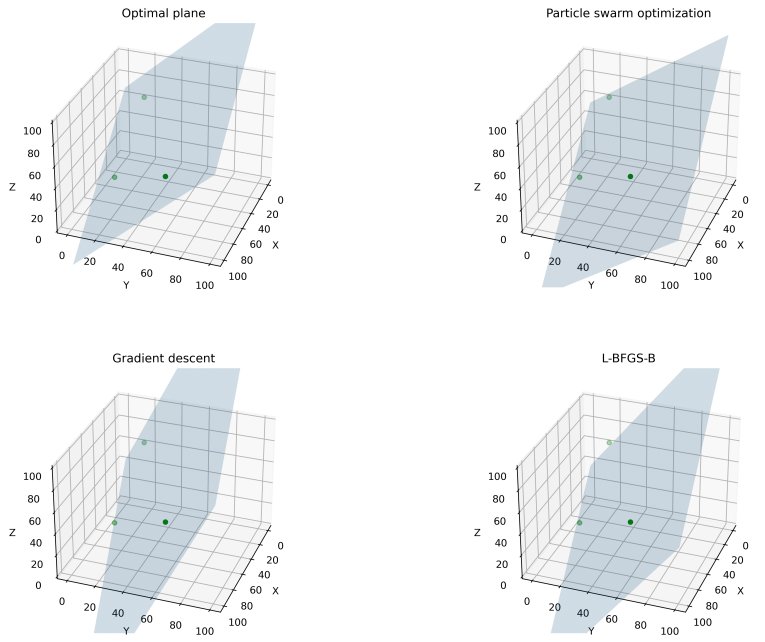

Fig. 3. Planes generated by running different optimization algorithms on an artificial uncertainty field. The top left figure shows the exact optimum, which passes through the three centers of the Gaussians that make up the uncertainty field. These three points are also visible in the other three graphs.

Results Figure 3 shows examples of resulting planes from this experiment, together with the centers of the three Gaussians. The planes usually tend towards the same normal, but can sometimes diverge. In addition, it is possible to see that the planes are not exactly at the same position as the optimal plane.

This shows that while the solutions tend to converge to maxima, none of the algorithms are guaranteed to find the global optimum. It also shows that the solutions are not exact, meaning they tend to vary.

4.3 Quantitative Analysis

To evaluate the different optimization algorithms quantitatively, the uncertainty found by the algorithms is measured against the iteration number. This allows us to compare their uncertainty values, as well as observe how quickly and where the algorithms converge.

Experimental Setup All three algorithms have a parameter that increases the amount of function calls linearly. In the case of particle swarm optimization, this is the number of particles N. In the case of gradient descent and L-BFGS-B, this is the number of random instantiations. Since both the runtime and the quality of the solutions is highly dependent on these parameters, we run the experiment for three different values, namely $N = 5$, $N = 12$ and $N = 30$.

For our test data, we use uncertainty fields generated after running the pipeline for one iteration, meaning without selecting any plane beforehand. This involves running the random walker algorithm on only initial seed points, without additional user input. The input images to the pipeline come from the data of the Head and Neck Auto Segmentation MICCAI Challenge (2015)[4]. This is an open dataset containing CT images from 48 different patients with labels for different organs. In our evaluation, we use 24 different images for evaluation. The rest were used for initial testing, setting parameter values or remain unused. We only use the labels for the mandible to generate initial seed points for the segmentation.

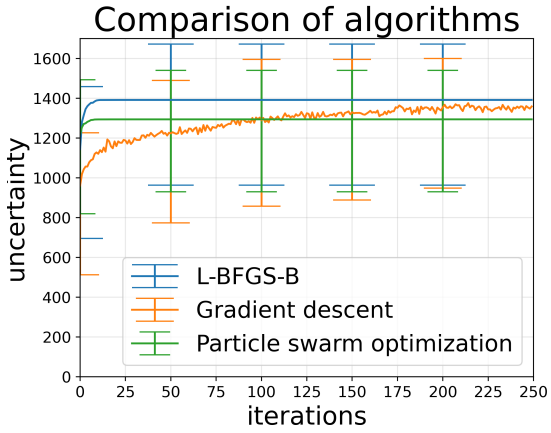

Fig. 4. The average uncertainty score (which is the cost function value) from running the three optimization algorithms, with the mean being calculated over the 24 generated uncertainty fields. These uncertainty fields were generated by running the pipeline for one iteration. The error bars show the top and bottom 25% amongst the uncertainty fields. For all three algorithms $N = 30$ was used.

In our results, we show the mean uncertainty value the algorithm found over these 24 different images. We also show error bars showing the top and bottom 25%.

Results Figures 4 and 5 show the results of the evaluation. Figure 4 compares the different algorithms for one value of N, while figure 5 compares different values of N per algorithm. Next to that, table 1 shows the average runtime of the algorithms for $N = 30$. There are a few things to note about these results.

First of all, figure 4 shows that gradient descent takes a lot more iterations to converge to an optimum than particle swarm optimization and L-BFGS-B. While increasing the step size might seem like a solution, this causes the position to diverge.

In addition, the results in figure 5 imply that for $N = 30$ and a large number of iterations, gradient descent and L-BFGS-B perform similarly. This is different from particle swarm optimization, which tends to converge to lower values than the other two algorithms.

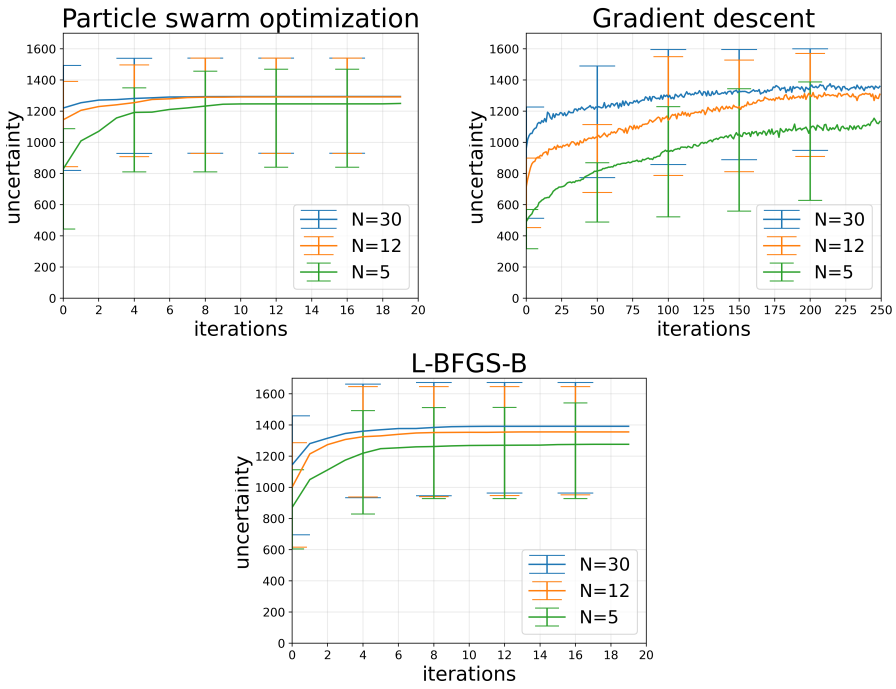

Fig. 5. The average uncertainty score (which is the cost function value) with different values of N for the different optimization algorithms, with the mean being calculated over the 24 generated uncertainty fields. These uncertainty fields were generated by running the pipeline for one iteration. The error bars show the top and bottom 25% amongst the uncertainty fields. Note that the horizontal axis only shows 20 iterations for particle swarm optimization and L-BFGS-B, but 250 iterations are shown for gradient descent.

Moreover, figure 5 shows increasing the value of N improves the result for all algorithms, both for large and small numbers of iterations. This shows that gradient descent and L-BFGS-B tend to converge to local optima. It also shows that particle swarm optimization experiences premature convergence. In addition, it shows that increasing N is a possible way to alleviate these problems, at the cost of a significant increase in runtime.

Figure 4 shows that gradient descent takes more iterations to converge than particle swarm optimization and L-BFGS-B, while table 1 shows that this difference in number of iterations also means a practical difference in runtime. While the runtime of gradient descent per iteration is lower than that of L-BFGS-B, the fact that it takes more iterations to converge means that it takes a longer time to converge. Particle swarm optimization is the fastest algorithm out of the three, both per iteration and in total.

	Average runtime	Number of iterations	Average runtime per iteration
Particle swarm optimization	7.5 s	20	0.38 s
Gradient descent	432 s	250	1.73 s
L-BFGS-B	70 s	20	3.5 s

Table 1. Average runtimes of the optimization algorithms for N=30. It should be noted that these runtimes are dependant on the actual implementations of the algorithms.

5 Responsible Research

During the research, significant emphasis is placed on its reproducibility. This is done in a number of ways. First of all, we make sure to mention implementation details like parameter values. The method of evaluation is also explained in depth. In addition, a publicly available dataset was used. Finally, the code is also available in a public Github repository[2].

6 Discussion

The main metric used to evaluate the different algorithms is the amount of iterations. However, the runtime per iteration is different for different algorithms. Theoretically speaking, particle swarm optimization should take the least amount of runtime, followed by gradient descent, then L-BFGS-B taking the most runtime. This is due to the fact that particle swarm optimization only has to evaluate the cost function itself, while gradient descent has to calculate the point and normal gradients and L-BFGS-B has to approximate the inverse Hessian.

The reason runtime was not plotted instead of number of iterations is because runtime is very dependant on the actual implementation. However, the measured runtimes

[2] github.com/avilakathara/Medical-Image-Processing

show that there is a very significant difference in runtimes caused by the number of iterations gradient has to run to converge. Therefore, it is safe to say that there is a practical difference in runtime.

The evaluation was only done on a limited dataset. It contains only CT images of the neck and head and the uncertainty images were only generated for one organ after one initialization. This is not representative of all possible uncertainty fields generated from 3D medical images. However, the results still make a strong case for L-BFGS-B as the best optimization algorithm out of the three.

The evaluation only measures the found uncertainty value, but the way this value influences the quality of the automatic image segmentation is not measured. This paper assumes that planes with more uncertainty are more important to label than planes with lower uncertainty values, because this assumption is also made in [5]. However, this paper does not evaluate this assumption.

7 Conclusions

The goal of this paper is to find out which optimization algorithm is best suited for finding the optimal plane in an uncertainty field. Particle swarm optimization, gradient descent and L-BFGS-B are evaluated and their convergence behaviour is observed.

Particle swarm optimization converges the fastest out of the three, but at lower uncertainty values than gradient descent. Gradient descent converges to high values, but does so over a very large number of iterations. L-BFGS-B generally converges to similar values as gradient descent and does so in a much smaller number of iterations. Therefore, L-BFGS-B seems most suited for solving this problem.

Compared to gradient descent, using L-BFGS-B in a full pipeline will decrease the amount of runtime needed. This means a user will have to wait for a shorter amount of time between annotating slices and the system will be quicker to use. Combined with finding high uncertainty values, the results imply that L-BFGS-B is the best algorithm to use in many situations. Particle swarm optimization decreases the runtime even further, but does so at the cost of reducing the found uncertainty. Therefore, more user input will be needed when using particle swarm optimization. However, the decreased runtime might still be useful if low runtime is of very high importance, such as in real-time systems.

Future work includes quantitatively evaluating how higher found uncertainty impacts the quality of the segmentation in a complete pipeline and evaluating more optimization algorithms than the three evaluated in this paper.

References

1. Byrd, R.H., Lu, P., Nocedal, J., Zhu, C.: A limited memory algorithm for bound constrained optimization. SIAM Journal on Scientific Computing **16**(5), 1190–1208 (1995)
2. Gad, A.: Particle swarm optimization algorithm and its applications: A systematic review. Archives of Computational Methods in Engineering **29**(5), 2531–2561 (2022)
3. Milletari, F., Navab, N., Ahmadi, S.A.: V-net: Fully convolutional neural networks for volumetric medical image segmentation. 2016 fourth international conference on 3D vision (3DV) pp. 565–571 (2016)

4. Raudaschl, P.F., Zaffino, P., Sharp, G.C., et al.: Evaluation of segmentation methods on head and neck ct: Auto-segmentation challenge 2015. Medical physics **44**(5), 2020–2036 (2017)
5. Top, A., Hamarneh, G., Abugharbieh, R.: Active learning for interactive 3d image segmentation. International Conference on Medical Image Computing and Computer-Assisted Intervention pp. 603–610 (2011)
6. Zhu, C., Byrd, R.H., Lu, P., Nocedal, J.: Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. ACM Transactions on Mathematical Software **23**(4), 550–560 (1997)