Final Project Report: Learning optimal parameters of Graph-Based Image Segmentation

Stefan Zickler
szickler@cs.cmu.edu

Abstract

The performance of many modern image segmentation algorithms depends greatly on the choice of their various parameters. Traditionally, in the vision community these parameters have often been manually tuned which can lead to non-optimal performance. In this project we attempt to automatically find the optimal set of a graph-based image segmentation algorithm's parameters using an evolutionary algorithm. We introduce an error metric that can be used to evaluate graph-based segmentation results by comparing them to a set of human-labeled segmentations. We then introduce an evolutionary algorithm which attempts to learn the optimal set of parameters for the graph-based segmentation algorithm. The performance of the results achieved by the evolutionary algorithm is analyzed and compared to the standard set of manually tweaked parameters.

1 Introduction

The performance of many modern image segmentation algorithms depends greatly on the choice of their various parameters. Traditionally, in the vision community these parameters have often been manually tuned which can lead to non-optimal performance. In this project we attempt to automatically find the optimal set of a graph-based image segmentation algorithm's parameters. In particular, we will investigate how evolutionary algorithms can be used to solve this problem.

2 Related Work

Graph based image-segmentation is a fast and efficient method of generating a set of superpixels, also known as segments, from an image. They supercede old edge-based approaches as they not only consider local pixel-based features, but instead look at global similarities within the image [3]. In the published literature, the graph-based algorithm has been run with a fixed, hand-tweaked set of parameters without any systematic optimization. Unfortunately, there exists no direct scientific metric of what makes a "good" image segmentation, other than the general idea that regions which are visually (or even contextually) similar should be grouped together to become part of the same segment. A frequent goal of image-segmentation is to determine the boundaries of each contextual object within the image. Thus, one of the best available methods of evaluating the quality of an image segmentation algorithm is by comparing it to some sort of ground truth, such as images which

have been manually segmented by humans. We will follow this approach in this project by comparing the results of a graph-based segmentation algorithm to the corresponding human segmentations.

Genetic algorithms (also known as evolutionary algorithms) have been a commonly used technique to perform parameter optimization [5][1][2]. They are particularly interesting because they can perform an efficient search without hard assumptions about the linearity of the function that is being optimized. Evolutionary algorithms follow the idea that the set of parameters can be improved by mutation, evaluation, and "survival of the fittest", similar to evolutionary processes in biology. Evolutionary algorithms are suited for our scenario, because we are not required to build a large dataset in advance (such as needed for e.g. regression-based methods), but rather generate required samples on the go.

3 Methodology

We would like to learn the set of parameters X which will minimize the mean segmentation error of our algorithm for some training dataset T which contains N images.

$$\arg\min_X \left(\frac{1}{N} \sum_{i=1}^{N} error(graphseg(t_i, X)) \right)$$

In our case the image segmentation algorithm has three free parameters: a smoothing factor σ, a scalar k which effectively determines the size of the created superclusters, and min, the minimum component size after post-processing. Therefore $X = [\sigma, k, min]$.

3.1 Segmentation Performance Metric

Before we can concentrate on the actual parameter optimization problem, we need to find a way of quantitatively comparing a computer-generated segmentation to the ground truth. In particular, we want to determine how well the edges of regions in our generated segmentation will match the edges of regions in the hand-labeled images. We do so by introducing an error metric where an error of zero will represent a perfect segmentation. Our error metric has two source components: oversegmentation error, and undersegmentation error. Oversegmentation error is caused by our algorithm finding region-edges where there should not be any. Undersegmentation is naturally its dual, namely when our algorithm does not generate a region-edge when in fact there should be one. We define the total segmentation error to be the maximum of its two components, namely:

$$error = \max(error_{\text{under}}, error_{\text{over}})$$

Our segmentation results are represented as binary edge-images, where each pixel represents a border between two segments. We call these images S_{algo} and S_{human}. A sample of these two edge-images can be seen in figure 1(b) and (c). The naive approach to compute our two error components would be to overlay our two edge-images and simply count all non-overlapping pixels vs all expected pixels.

$$error_{\text{under}} = \frac{|S_{\text{human}} - S_{\text{algo}}|}{|S_{\text{human}}|} error_{\text{over}} = \frac{|S_{\text{algo}} - S_{\text{human}}|}{|S_{\text{algo}}|}$$

This metric however is bad for two reasons. For one thing, it is assuming that the human segmentations are pixel-perfect which they are clearly not. Another problem of this metric is that it is an all or nothing approach. That is, a region-edge which is only one pixel away from the correct location will be penalized as much as a region-edge which is 100 pixels away. To overcome both of these problems, we introduce a metric which has a more probabilistic nature. We do so by constructing the distance transform of our edge images,

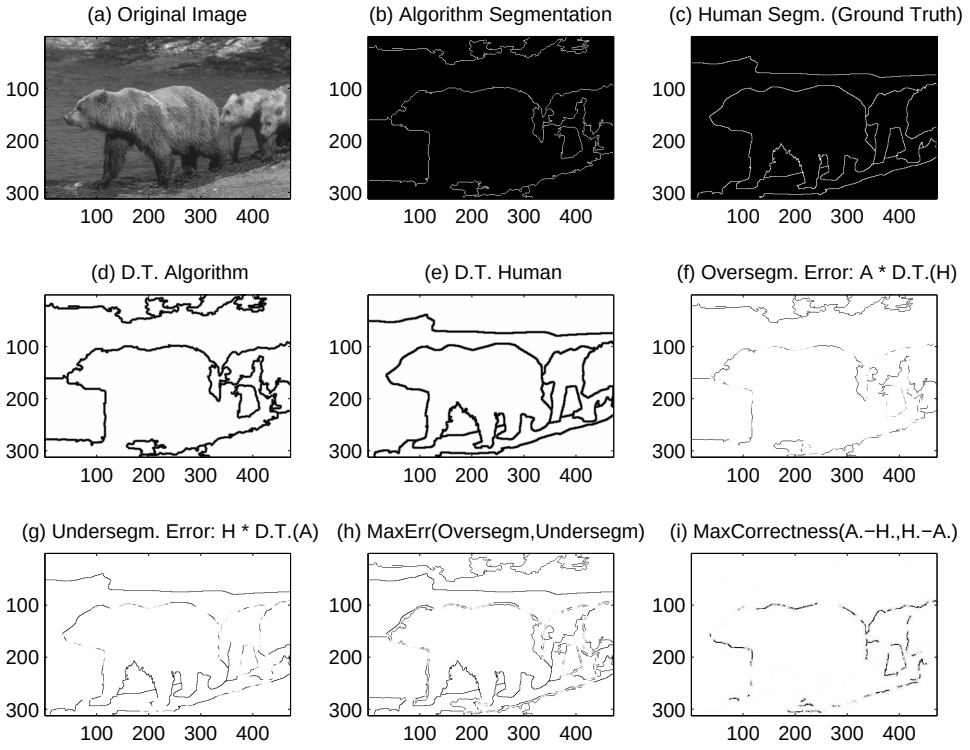

Figure 1: A sample image from the Berkeley dataset

and then applying the logistic function to it with parameters which will give us a smoothed grayscale widening of the edges in our original edge-images.

$$error_{\text{under}} = \frac{|S_{\text{human}} * \text{DT}(S_{\text{algo}})|}{|S_{\text{human}}|} error_{\text{over}} = \frac{|S_{\text{algo}} * \text{DT}(S_{\text{human}})|}{|S_{\text{algo}}|}$$

where DT is our modified distance transform. Note that this metric gives us a smooth transition from perfectly matching pixels to pixels which are considered too far away to be considered a match at all. The modified distance transforms and the error components can be seen in figure 1(d)(e)(f)(g) respectively.

3.2 Optimization Algorithm

As mentioned earlier, we will be using an evolutionary algorithm to find the optimal set of parameters. The basic structure of the algorithm is depicted in figure 2. In our case, the fitness of an individual is clearly the inverse of its $error$ as described in the previous section. A population is a set of parameter vectors such that $Population = \{X_1, X_2, ..., X_M\}$ where M is the population size and X is the parameter vector of our segmentation algorithm ($X = [\sigma, k, min]$). The CROSSOVER function will select a random element of the parameter vector X and compute the mean from both of its parents. The MUTATION function will mutate the offspring of a single parent by adding a random integer drawn from a gaussian distribution to a randomly chosen element of the parameter vector X. The MERGEANDCROP function will merge the children and parents into the same population and then shrink this population back to size M by removing the entries with the lowest fitness.

```
F ⟵ Fitness Function
F_t ⟵ Termination Fitness
G ⟵ Number of Generations
M ⟵ Size of Population
p_c ⟵ Crossover Frequency
p_m ⟵ Mutation Frequency
Population ⟵ RANDOMPOPULATION(M)
CALCFITNESSOFEACHINDIVIDUAL(Population)
while F(BESTINDIVIDUAL(Population)) < F_t and GenerationNum < G
{
  Parents ⟵ Population
  Children ⟵ CROSSOVER(Parents, p_c)
  Children ⟵ MUTATION(Parents, p_m)
  CALCFITNESSOFEACHINDIVIDUAL(Children)
  Population ⟵ MERGEANDCROP(Parents, Children)
  GenerationNum ⟵ GenerationNum + 1
}
return(BESTINDIVIDUAL(Population))
```

Figure 2: Pseudocode of GA

4 Experiment

The experiment is performed on the manually segmented Berkeley image dataset `http://www.cs.berkeley.edu/projects/vision/grouping/segbench/` [4]. This dataset contains 300 images in which objects' boundaries have been annotated by human subjects. We use a subset of these images as training data, while we reserve 100 images as independent testing data. We run the genetic algorithm described in the previous section on the training data to find the optimal set of parameters. As segmentation algorithm, we use the publicly available standard implementation of the graph-based segmentation algorithm [3]. We furthermore compute the mean testing error by using the parameters which were found by the genetic algorithm on the testing set.

It should be noted that the Berkeley dataset actually offers multiple human segmentations per image, thus offering multiple "ground truths" for each image. We make use of this fact by computing the error against each of these segmentations and taking the minimum. Thus if the algorithm perfectly matches any one of the human segmentations of a given image, it will be considered to have an error of zero on that particular image. In our experiments we use a population size of 8 and crossover and mutation frequencies of 0.8 each. This means that at each iteration, we will generate 6 children and 6 mutants, growing the population to size 20, which will then be shrunk to only contain the best 8 individuals again. Certainly these values are trade-offs. Generating more children or mutants increases the chance of creating stronger individuals, but it also increases computational complexity per generation. Since the fitness computation for a single individual will already take a fairly long amount of time (approximately one minute for a training set of 40 images, on a modern Pentium 4 machine), we kept the population size and mutation/crossover rates fairly low.

The implementation of the entire genetic algorithm, of the error metric computation, and the experimental automation was done in Matlab.

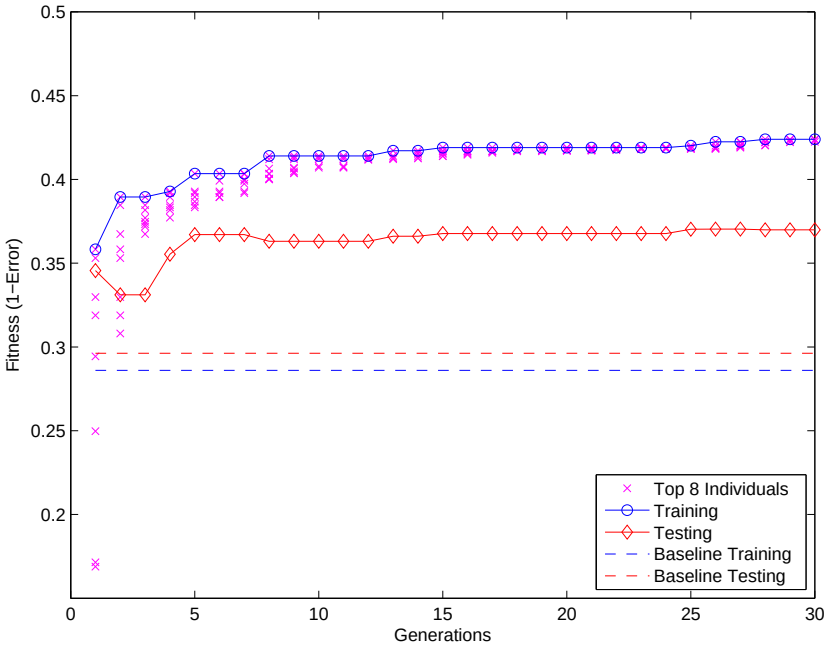

Figure 3: Fitness curve (training set size 20).

5 Results

In figure 3 we can see the performance of the algorithm over multiple iterations. The graph was generated with a training set size of 20 images. The y-axis in this graph represents the fitness-value which is defined as $1 - error$. The solid blue line represents the training fitness of the best individual of each generation. Furthermore we show the fitnesses of the top 8 individuals for each generation. The numeric values and the best individual for each generation can also be seen in table 5. Just as we would expect it from the algorithmic definition, the training fitness increases constantly. As it is typical for evolutionary processes, there are certain periods of plateaus until finding a lucky mutation which increases the fitness significantly. Eventually we see the training fitness converge. We can already see that our optimization algorithm easily outperforms the best-guessed values which were suggested by the segmentation algorithm's author (called "baseline" and represented as dashed lines in the graph). The parameters used for this baseline were: $\sigma = 0.8$, $k = 300$, $min = 50$. If all we wanted is to tweak the segmentation algorithm for a particular dataset then our goal could already be considered achieved.

However, we would like to make sure that our new set of parameters is generally applicable and will also perform well when used on other, non-trained images. For this purpose we run the segmentation algorithm on the independent testing data with the parameters obtained during training iterations. The solid red line represents the fitness-values of the testing set. We can see the general trend of the testing fitness increasing with more generations. Again, we can see a fairly quick convergence. More importantly, the optimized results are all very well above the manually tweaked parameter baseline (dashed line). In fact, this happens already within the first iteration of the algorithm.

Iteration	x_1 (σ)	x_2 (k)	x_3 (min)	Fitness (training)
1	0.4623	944.9364	734.3545	0.3583
2	1.1998	791.0263	652.4015	0.3895
3	1.1998	791.0263	652.4015	0.3895
4	1.0939	953.7888	364.9563	0.3928
5	1.1056	969.3711	176.7327	0.4035
6	1.1056	969.3711	176.7327	0.4035
7	1.1056	969.3711	176.7327	0.4035
8	1.1699	969.3711	176.7327	0.4140
9	1.1699	969.3711	176.7327	0.4140
10	1.1699	969.3711	176.7327	0.4140
11	1.1699	969.3711	176.7327	0.4140
12	1.1699	969.3711	176.7327	0.4140
13	1.1699	969.3711	147.5138	0.4171
14	1.1699	969.3711	147.5138	0.4171
15	1.1647	969.3711	103.4557	0.4190
16	1.1647	969.3711	103.4557	0.4190
17	1.1647	969.3711	103.4557	0.4190
18	1.1647	969.3711	103.4557	0.4190
19	1.1647	969.3711	103.4557	0.4190
20	1.1647	969.3711	103.4557	0.4190
21	1.1647	969.3711	103.4557	0.4190
22	1.1647	969.3711	103.4557	0.4190
23	1.1647	969.3711	103.4557	0.4190
24	1.1647	969.3711	103.4557	0.4190
25	1.1647	953.8638	105.7130	0.4201
26	1.1647	953.8638	103.7142	0.4225
27	1.1647	953.8638	103.6152	0.4225
28	1.1647	935.1816	103.6152	0.4240
29	1.1647	935.1816	103.6152	0.4240
30	1.1647	935.1816	103.6152	0.4240

Figure 4: Best individuals and their fitness-value for each generation (training set size 20).

In order to investigate what effect a larger training set size might have, we re-ran the experiment with a training set size of 40 which is twice as large as the previous run. The graph of the increased training set can be found in figure 5. While the overall result looks fairly similar, we can notice that the difference between training and testing is not as drastic. However, we can also notice that the testing set fitness actually decreases slightly at about iteration 20. We suspect that this is because the algorithm is overfitting its parameters onto the training set.

6 Conclusion

We have introduced a flexible error metric that can be used to compare computer-generated image segmentations to human-labeled ones. We have furthermore introduced a simple evolutionary algorithm which makes use of this error metric to find an optimal set of parameters for a graph-based image segmentation algorithm. We have analyzed the performance of the introduced approach by running it on independent training and testing sets. We have furthermore compared its performance to a standard set of manual parameters which are commonly used for this particular graph-based image segmentation algorithm. While this paper put the focus on a graph-based segmentation algorithm, it can be assumed that this

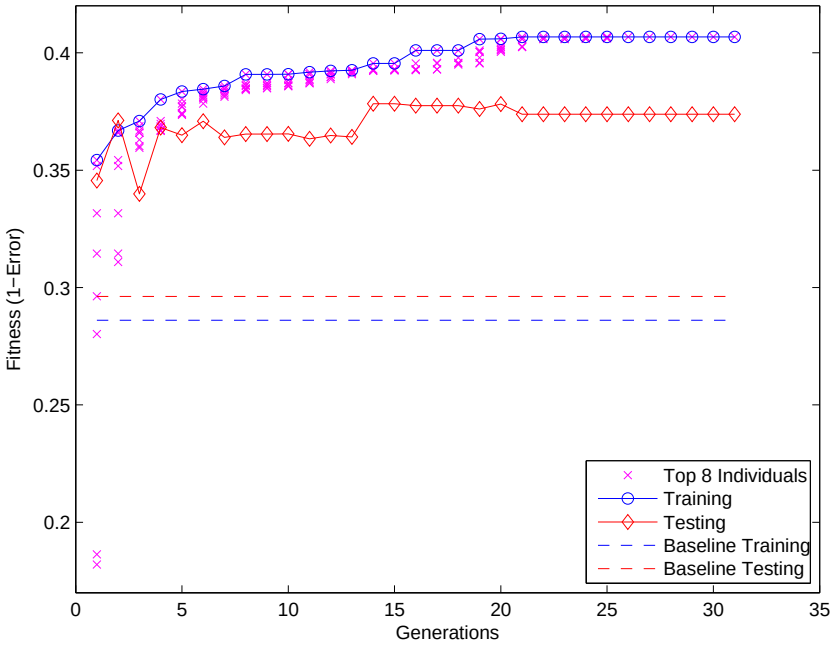

Figure 5: Fitness curve (training set size 40).

approach is easily extendable to any other segmentation algorithm that would otherwise require manual parameter tweaking.

References

[1] T. Bäck and H.P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[2] S. Chernova and M. Veloso. An evolutionary approach to gait learning for four-legged robots. *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, 3.

[3] P.F. Felzenszwalb and D.P. Huttenlocher. Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[4] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.

[5] A.H. Wright. Genetic algorithms for real parameter optimization. *Foundations of Genetic Algorithms*, 1:205–218, 1991.