Improving Graph-based Image Segmentation Using Automatic Programming

Lars Vidar Magnusson[1] and Roland Olsson[1]

Østfold University College, IT Department, Halden, Norway

Abstract. This paper investigates how Felzenszwalb's and Huttenlocher's graph-based segmentation algorithm can be improved by automatic programming. We show that computers running Automatic Design of Algorithms Through Evolution (ADATE), our system for automatic programming, have induced a new graph-based algorithm that is 12 percent more accurate than the original without affecting the runtime efficiency. The result shows that ADATE is capable of improving an effective image segmentation algorithm and suggests that the system can be used to improve image analysis algorithms in general.

Keywords: image segmentation, graph algorithm, evolutionary computation, automatic programming

1 Introduction

Image segmentation involves partitioning an image into segments or components corresponding to objects in the image. Image segmentation has many applications and is typically used as an early step in a series of image processing techniques. As a result, accurate image segmentation is important; it is likely that it will affect the quality of all image processing that follows. Image segmentation algorithms employ a set of visual cues, such as intensity, texture or shape, to partition an image into its constituent objects. Combining two or more of these cues has been shown to improve the accuracy of an algorithm, but it also requires more processing time. There are also applications for image segmentation that require extremely fast processing, in which case, accuracy has to be sacrificed for runtime performance.

Here we show that it is possible to use evolutionary computation to automatically improve the core algorithm of a highly efficient graph-based segmentation technique [1] – without having to incorporate any additional visual cues, and without altering its overall computational efficiency. By using the ADATE automatic programming system, we have been able to automatically generate a new algorithm with 12 percent better segmentation accuracy on a popular image database.

The scientific contributions of this paper can be summarized as follows.

1. An automatically generated and significantly more accurate graph-based image segmentation algorithm that runs as fast as the algorithm on which it was based.

2. Further evidence to support the hypothesis that ADATE can generate new and improve image analysis algorithms in general.

2 Background

Two scientific areas provide the background for this paper. The first is the relatively new machine learning discipline of automatic programming, and the second is the well studied discipline of image segmentation. The following sections present the background considered directly relevant for this paper.

2.1 Automatic Design of Algorithms Through Evolution (ADATE)

ADATE [2] is a system for automatic programming which infers purely functional programs through incremental program transformations – guided by evolutionary principles. The system is capable of inventing auxiliary functions, generating general recursive patterns, and creating and optimizing numerical constants. It can be used either to create entirely new programs, or to improve existing ones.

All programs generated by the ADATE system are evaluated with a user-specified evaluation function – allowing anything from simple input/output pairs to complex simulations. This flexible evaluation system, along with the search-based approach of incremental transformations, make the system useful in many cases where other automatic programming systems would fall short [3–5].

The program transformations performed by the ADATE system are grouped into four categories. The first, and most fundamental, category is called *Replacement* (R). A replacement can either replace an entire expression with a synthesized expression, or it can reuse parts of it, as subexpressions. Replacements are separated into two groups to facilitate the evolution process. In one group are the synthesized expressions that change the semantics of the original program. These provide a mechanism for introducing improvements into the program. The replacements in the other group typically maintain the semantics of the original program. More precisely, they do not harm the performance of the original program. These are referred to as *replacements preserving equality* (REQ), and they are important in the evolution process, as they provide a mechanism for doing neutral walks in the search landscape.

The remaining program transformation categories are *Abstraction* (ABSTR), *Case-distribution* (CASE-DIST) and *Embedding* (EMB). These are responsible for creating new auxiliary functions, changing the scope of variables and function, and introducing new function parameters respectively. They maintain the semantics of the original program, and are far less combinatorially challenging than replacements.

The problems are provided to ADATE in *specifications* containing any code needed to run the algorithm, along with any predefined functions, the training instances and the evaluation function that will be used to evaluate the generated programs. ADATE operates using a bare-bone subset of SML [6] called ADATE ML. The language has been stripped of all syntactic sugar to simplify the evolution – essentially reducing the language down to functions and case-expressions.

2.2 Graph-based Image Segmentation

Graph-based image segmentation is a generic term covering image segmentation algorithms that use a graph-theoretic approach to partition an image into its constituent objects. In this respect, images are typically represented as a graph $G = (V, E)$, where each element in the set of vertices V represents a pixel in the image, and the set E contains edges that connect the vertices in the image according to a neighborhood relation. Each edge has an associated weight that represents some attribute derived from the vertices that it connects.

Wu and Leahy [7] proposed a graph-based data clustering algorithm based on a minimum cut – the set of edges with the smallest weights that partitions a graph into two disjoint subgraphs – and a graph compacting technique. The algorithm employs an efficient multi-terminal network flow algorithm to find the maximum flow between all the nodes in the image graph. This makes it possible to optimally divide the input graph into K regions by removing the edges belonging to the $K - 1$ minimum cuts. The weights in the graph represent the difference in intensity between neighboring pixels in the image being segmented. The runtime performance of the algorithm is polynomial in the number of nodes in the graph, but the algorithm is biased towards small regions.

This shortcoming was addressed by Shi and Malik [8] with the introduction of a *normalized* cut, which normalize the value of each cut using the sum of the weights of the edges between the nodes in a subgraph and the entire graph. This criterion removes the bias towards small regions, but it is computationally inefficient compared to the minimum cut – the decision variant is NP-complete. They show that an approximation can be found by solving a generalized eigenvalue system. This makes the problem tractable, but it still requires long runtimes due to the size of the matrix required to represent the images. Shi and Malik proposed a set of cues suitable for different applications, but only one can be used at any particular time. The framework was extended further by Malik et al. [9] by incorporating a combination of contour and texture cues. They proposed combining the two cues using a simple gating mechanism triggered by the *texturedness* of a region. After an initial over-segmentation, they recalculate the weights and combine regions until a normalized cut threshold is reached.

Felzenszwalb and Huttenlocher [1] introduced a graph-based algorithm for image segmentation that – while operating solely on local attributes – manages to satisfy certain desirable global properties, by producing segmentations that, as defined by the authors, are neither too fine nor too coarse. Unlike the algorithms presented above, this algorithm starts out with each pixel as a separate region and continues to merge regions in a bottom-up fashion. The proposed algorithm employs an adaptive segmentation strategy that keeps track of the similarity of the pixels within a segmented region and the dissimilarity between the different regions. The algorithm is $O(n \log n)$, where n is the number of pixels in the image – a significant improvement over the algorithms above – and most of the time is spent sorting the edges. The proposed algorithm is simple and efficient in its design, and it employs intensity cues only.

Alpert et al. [10] proposed a Bayesian probabilistic framework for combining visual cues. The framework was designed to work with any bottom-up merge based image segmentation algorithm, but it was demonstrated using a Segmentation by Weighted Aggregation (SWA) algorithm as proposed by Galun et al. [11] and Sharon et al. [12]. The algorithm starts with each pixel in the image being represented by a node in the graph. In each iteration, the graph is made coarser by merging seed nodes with their neighbors according to their similarity. A segmentation hierarchy is formed by relating nodes in the coarser graph with the nodes in the previous step. Edge weights are updated recursively by averaging the features, or cues, from earlier steps through weighted aggregation. The algorithm is linear in the number of pixels in the image, but the actual runtime is high due to large matrices and large runtime constants. The two original variants of the SWA algorithm use a wide set of visual cues, whereas the Bayesian framework proposed by Alpert et al. [10] was demonstrated using only intensity and texture cues. The reported runtime for the algorithm with the full feature-set [11] is between 5 and 10 seconds on a 400×400 image using a 1.6 Xeon GHz processor.

It is apparent from the research presented above that the algorithm proposed by Felzenszwalb and Huttenlocher [1], though relatively simple both in terms of the overall design and its use of a single intensity cue, is capable of competing with more complex algorithms [13]. As such, it is a good starting point for an attempt to improve a leading image segmentation algorithm using automatic programming. Preliminary work by Huyen and Olsson [14] indicated that the algorithm can be improved. However, this preliminary study had serious limitations. The images used were scaled down to reduce the memory required, and the specification lacked essential features, such as noise filtering, available in the original algorithm. As a consequence, the algorithm is less general than the original and practically useless on the full-sized images. All of these limitations have been addressed in our work to allow the evolution of an algorithm that can perform well even if the conditions change.

3 Experiments

This section describes the most important parts of converting the problem into the proper format for ADATE.

3.1 The Implementation of the Original Algorithm

The original algorithm had to be ported in its entirety to ADATE ML – the language in which ADATE evolves programs. The C++ code provided by Felzenszwalb and Huttenlocher, in addition to the actual segmentation algorithm, features a Gaussian noise reduction filter and a post-processing step that merges any neighboring components under a certain size. Both these features have been included in our ADATE ML implementation to ensure identical operational semantics for both implementations.

The most important parts of the code in the ADATE ML implementation are located in two functions *main* and *f*, where the definition of the latter contains the code to be modified and improved by ADATE. The *main* function is executed once per image, and is responsible for transforming the pre-processed image data into a graph by letting each pixel be represented by a node, which is connected to its eight immediate neighbors by weighted edges corresponding to the dissimilarity of the pixels. All of the edges are sorted in non-decreasing order, and the data structures used to represent the components during the segmentation process are initialized. Each node in the graph starts out as a component with a threshold corresponding to a constant C that controls how large the resulting segments will be. After these initial tasks, the function invokes the recursive f function to do the actual segmentation, the result of which is post-processed to merge components that are smaller than a certain size.

Every invocation of f selects the next candidate edge from the list of sorted edges. Its weight is compared to the thresholds of the two components that it connects – if they do not belong to the same component already. If the weight W is smaller than the thresholds of both components, the two connected components are merged into a new component, and the threshold of the component is set according to the following equation.

$$T_N = W + \frac{C}{|N|} \ . \tag{1}$$

Here N represents the new component, $|N|$ is the cardinality of N, and T_N is the threshold of N.

The ADATE ML implementation of the algorithm was tested both with a third-party ML compiler and with ADATE's internal compiler before the evolution was started, and it produced the exact same results as the original C++ implementation in both cases.

3.2 The Training and Test Images

There are several image databases available that provide natural images manually annotated by humans, but two of them distinguish themselves from the others in terms of quality, the Berkeley Segmentation Data Set (BSDS) [13] and the Weizmann Segmentation Evaluation Database (WSED) [10]. The way the BSDS evaluates segmentations arguably favors algorithms that are either based on or include some form of edge detection. This makes the dataset unsuitable for evaluating region growing algorithms like the algorithm by Felzenszwalb and Huttenlocher [1]. We therefor chose to use the WSED instead, even though it has fewer images and contains only images with one foreground object.

The dataset contains a total of 100 images with a single foreground object that have been annotated by three or more individuals. Of these 100, 50 were used for training, and 50 were used for testing.

3.3 Measuring the Accuracy of Generated Programs

We have used the same means of measuring the performance of the generated programs that is used in the WSED to determine the accuracy of a segmentation, the F-measure [15] as defined in (2). *Precision* (P) is the ratio of the number of true-positive pixels to the sum of the number of true-positive and the number of false-positive pixels, and *Recall* (R) is the ratio of the number of true-positive pixels to the sum of the number of true-positive and the number of false-negative pixels.

$$F = \frac{PR}{0.5(P+R)} \tag{2}$$

The Felzenszwalb and Huttenlocher algorithm produce segmentations that partition an image into its objects, rather than just foreground and background. To determine the quality of any segmentation the regions are all evaluated and the region with the highest score is selected.

3.4 Selecting The Constant Values

There are three constants in the original algorithm: the standard deviation of the Gaussian noise reduction filter, the threshold for merging components in post-processing, and the constant C that controls the tendency for components to merge.

The standard deviation for the noise filter was set to 0.5, which produced marginally better results on our dataset than the one used by Felzenszwalb and Huttenlocher [1]. The component size threshold for merging components in the post-processing step was not discussed in their article, but it is included in the C++ code provided. We therefore had no reference for choosing this value, and, due to the way the post-processor operates, it could not simply be chosen by optimizing its value. This might interfere with the evolution of an improved algorithm by forcing it to produce suboptimal segmentations to fit the post-processor. Based on this, we decided to use a relative small threshold of 20 to keep the interference with the evolution to a minimum, but at the same time ensure that drastic over-segmentations do not slow down the evaluation.

The third constant C – the only constant used directly in the algorithm – was chosen by running the algorithm on the entire image dataset with values ranging from 500 to 2500. Based on the results, we decided to run our experiments with a value of 1000 for the C constant.

4 Results

In this section the evolved algorithm is presented, along with an analysis of how it behaves and performs in terms of segmentation quality.

4.1 The Improved Algorithm

The program shown in Listing 1 was evolved over only ten generations – an incredibly low number when compared to other problems tackled by ADATE in the past. This shows that it was quite easy for ADATE to improve the Felzenszwalb and Huttenlocher algorithm [1]. We will discuss the changes separately to highlight the semantic difference between the two algorithms, then we will discuss how the changes in semantics collectively affect the behavior of the algorithm.

```
1  fun f( Universe, SortedEdges, Constant ) =
2    case SortedEdges of
3      enil => Universe
4    | econs( CurrentEdge as edge( A, B, W, X ), RestEdges ) =>
5    let
6      val ( ComponentA, ThresholdA ) = find( A, Universe )
7      val ( ComponentB, ThresholdB ) = find( B, Universe )
8    in
9      if differentComp( ComponentA, ComponentB ) then
10       if W < ThresholdA andalso W < ThresholdB then
11       let
12         val NewUniverse =
13           updateThresholdValue(
14             ComponentB,
15             W+Constant/
16               getComponentSize(
17                 if Constant < ThresholdA then
18                   ComponentB
19                 else
20                   ComponentA ),
21             union( Universe, ComponentA, ComponentB ) )
22       in
23         f( NewUniverse, RestEdges, Constant )
24       end
25       else if W > ThresholdA andalso W > ThresholdB then
26         f( Universe, RestEdges, getComponentSize( ComponentB ) )
27       else
28         f( Universe, RestEdges, Constant )
29     else
30       f( Universe, RestEdges, Constant )
31   end
```

Listing 1. The improved algorithm – written in Standard ML to simplify the syntax.

The two algorithms are identical in terms of semantics until the if-expression on lines 10 through 24. The first case of this if-expression – when the edge weight is smaller than both the thresholds of the components that the edge connects –

cause the components to be joined and the threshold to be updated. But, instead of setting the threshold on the new component like in the original algorithm, it is set on the second connected component. The threshold is also calculated differently by no longer using the size of the joined component to divide the constant as in (1), but instead uses the size of one of the connected components – depending on a test to see if the constant is less than the threshold of the first connected component. The new algorithm has also introduced a new test on line 25 that checks whether the edge weight is bigger than both the thresholds of the components that the edge connects. In these situations the value of the constant is changed to the size of the second connected component. This, technically, makes the value a variable, but, for the sake of convenience, we will continue to refer to it as the constant.

The deceptively simple change that sets the new threshold on the second connected component, rather than the joined component, has the obvious effect that only about half the threshold updates will make a difference. Due to the way the components are represented, the second connected component has to be the component with the highest *rank* – a disjoint-set heuristic used to keep the trees shallow [16]. In most cases this translates into to the second connected component being the largest of the two. This is exploited on line 17 where a comparison of the current constant and the threshold of the first connected component determines which size of the components to use to calculate the new threshold. If the constant is equal to or greater than the threshold, the size of the first component is used, and in 98 percent of the cases, the size of this component is equal to or less than the size of the second component.

The new test on line 25 culminates into a recursive call that changes the constant value to the size of the second connected component when the current edge weight is larger than both the thresholds of the connected components. This normally occurs only after the components have been growing for some time, and as a result the new value is normally much larger than the original value. This changes how the algorithm operates on the remaining edges of the image. Any components merged after this will have much higher threshold than normal, essentially marking it with a high degree of variance and drastically increasing the chance of it being merged again.

Whereas the component threshold in the original algorithm is strictly decreasing and inversely proportional to the size of the component, the threshold in the improved algorithm fluctuates depending on the conditions under which two components are joined together. This allows the algorithm to react to patterns that occur during the segmentation.

4.2 Comparison of the Segmentation Quality

The segmentation accuracy of the new algorithm drastically exceeds the segmentation accuracy of the original algorithm. The images in Fig. 1 are some of the examples that showcase the differences between the segmentations produced by the two algorithms.

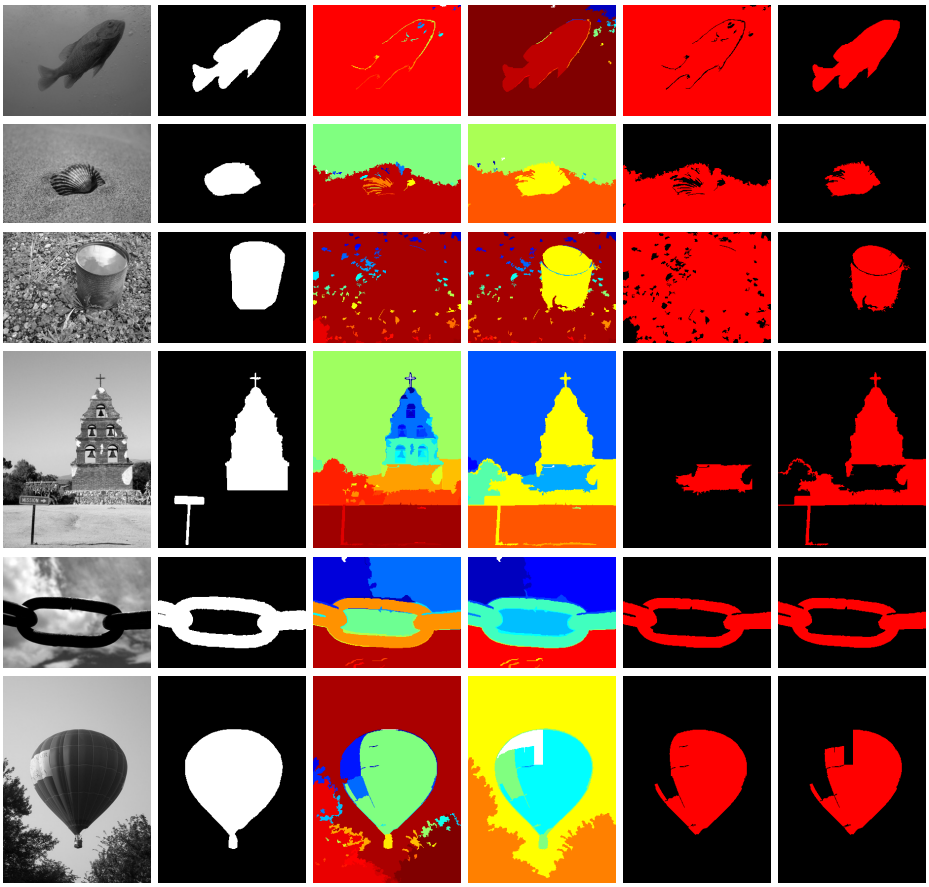

Fig. 1. A selected set of images and their segmentations. From left to right: The original images, the ground-truth images, the segmentation produced by the original algorithm, the segmentation produced by the improved algorithm, the best segment from the original segmentation, and the best segment from the improved segmentation.

The first three images are all instances where the new algorithm is far more accurate than the original. In all three images the original algorithm has joined most, if not all, of the foreground object with the background, while the improved algorithm has managed to keep them separate. The segmentation of the fourth image is also improved with the new algorithm, but not by the same amount as the first three. The original segmentation suffers from being over-segmented, while the improved segmentation is an under-segmentation. In the fourth image the score of the segmentation produced by the improved algorithm is slightly lower than the original. In the last image, the reduced quality of the improved algorithm is plainly visible – in the form of an over-segmentation.

4.3 Algorithm Benchmarks

The algorithms were benchmarked separately on both the 50 training images and the 50 test images. We used the same means of measurement as we did during the evolution; we used the F-measure as defined in (2) to determine the accuracy of each segment, and the maximum score to represent the quality of the segmentation.

We tested both algorithms on both the full-sized images and on image reduced in size, where the latter were included to establish whether or not the new algorithm is specialized to the conditions under which it was evolved. The algorithms were all tested using the same noise filter and post-processor settings as during the evolution. The remaining constant were optimized using the 50 training images.

Table 1. The average results from running the two algorithms on the full-sized images. The columns P, R and F represent the average *Precision*, *Recall* and *F-measure* respectively.

	Train			**Test**			**Total**		
Algorithm	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
New	0.79	0.86	0.79	0.81	0.82	0.78	0.80	0.84	0.79
Original	0.75	0.82	0.71	0.76	0.76	0.69	0.75	0.79	0.70

Table 2. The average results from running the two algorithms on the images that have been reduced to a quarter of their original size. The columns P, R and F represent the average *Precision*, *Recall* and *F-measure* respectively.

	Train			**Test**			**Total**		
Algorithm	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
New	0.73	0.79	0.73	0.76	0.81	0.75	0.75	0.80	0.74
Original	0.75	0.76	0.70	0.78	0.78	0.73	0.77	0.77	0.71

The average results from running both algorithms on all the full-sized images can be seen in Table 1. It is apparent from this data that the improved algorithm, on average, outperforms the original algorithm in terms of both precision and recall – yielding an average F-measure 11.5 percent, or 9 percentage points, better than the original. We also did a pairwise comparison of the two algorithms using student-t distribution on the differences, and we can say with 99 percent confidence that the new algorithm is between 1 and 17 percentage points better than the original on the test images.

The benchmark averages are directly comparable to the *one segment coverage test* in Alpert et al. [10] due to using the exact same images and performance measure. Among the algorithms in this test are both the three SWA based algorithms [10–12] and the normalized cut with gated intensity and texture cues [9]. From their results we can see that the algorithm algorithm presented here is still not as good as the two best algorithms [10, 11], but it manages to outperform the remaining two [12, 9]. All of these algorithms employ two or more cues, and they are, based on their efficiency and reported runtime, an order of magnitude slower.

Alpert et al [10] also tested another very popular algorithm [17] that only uses intensity cues to segment an image. This makes it comparable to the algorithms tested here, but the results show that the accuracy of this algorithm is far worse.

The average results from running the algorithms on the images reduced in size can be seen in Table 2. The improved algorithm outperforms the original here as well, but only by 3.4 percent. Based on the data, this seems to be due to a slight under-segmentation, when compared to the segmentations of the full-sized images.

5 Conclusions

We have successfully been able to improve a leading image segmentation algorithm by using automatic programming, and the new algorithm is both small, efficient and superior to comparable algorithms. The algorithm evolved by ADATE has kept the runtime efficiency of the original algorithm, and the segmentation quality has been improved by 12 percent on full-sized images. This improvement has been achieved without adding any additional visual cues. Instead it has been made possible by the adaptive mechanisms automatically invented by the ADATE system.

The success of our attempt at using ADATE for this purpose provides further evidence that the system is capable of improving state of the art image segmentation algorithms – if not image processing algorithms in general. The ADATE system, through its evolutionary strategy, is highly suitable for problems, like image segmentation, where we typically are looking for the best approximation, not the exact solution. These situations typically require a good heuristic, and the ADATE system has proven several times to be capable of creating customized code to fit this need.

References

1. Felzenszwalb, P., Huttenlocher, D.: Efficient graph-based image segmentation. International Journal of Computer Vision **59** (2004) 167–181
2. Olsson, R.: Inductive functional programming using incremental program tranformation. Artificial Intelligence **74** (1995) 55–81
3. Berg, H., Olsson, R., Lindblad, T., Chilo, J.: Automatic design of pulse coupled neurons for image segmentation. Neurocomputing **71**(10-12) (2008) 1980–1993 Neurocomputing for Vision Research; Advances in Blind Signal Processing.

4. Berg, H., Olsson, R., Rusås, P.O., Jakobsen, M.: Synthesis of control algorithms for autonomous vehicles through automatic programming. In: Proceedings of the 2009 Fifth International Conference on Natural Computation - Volume 04. ICNC '09, IEEE Computer Society (2009) 445–453
5. Løkketangen, A., Olsson, R.: Generating meta-heuristic optimization code using adate. Journal of Heuristics **16** (2010) 911–930
6. Milner, R., Tofte, M., Harper, R., MacQueen, D.: The Definition of Standard ML - Revised. The MIT Press (1997)
7. Wu, Z., Leahy, R.: An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **15**(11) (November 1993) 1101–1113
8. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence **22** (2000) 888–905
9. Malik, J., Belongie, S., Leung, T., Shi, J.: Contour and texture analysis for image segmentation. International Journal of Computer Vision **43**(1) (2001) 7–27
10. Alpert, S., Galun, M., Basri, R., Brandt, A.: Image segmentation by probabilistic bottom-up aggregation and cue integration. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, IEEE (June 2007) 1–8
11. Galun, M., Sharon, E., Basri, R., Brandt, A.: Texture segmentation by multiscale aggregation of filter responses and shape elements. In: Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on Computer Vision, Volume 1, IEEE (2003) 716–723
12. Sharon, E., Galun, M., Sharon, D., Basri, R., Brandt, A.: Hierarchy and adaptivity in segmenting visual scenes. Nature **442**(7104) (2006) 810–813
13. Arbelaez, P., Maire, M., Fowlkes, C., Malik, J.: Contour detection and hierarchical image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence **33** (2011) 898–916
14. Huyen, V., Olsson, R.: Automatic improvement of graph based image segmentation. In: Advances in Visual Computing. Volume 7432. Springer Berlin / Heidelberg (2012) 578–587
15. Rijsbergen, C.J.V.: Information Retrieval. 2nd edn. Butterworth-Heinemann, Newton, MA, USA (1979)
16. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. Journal of the ACM (JACM) **22**(2) (1975) 215–225
17. Comaniciu, D., Meer, P.: Mean shift: A robust approach toward feature space analysis. Pattern Analysis and Machine Intelligence, IEEE Transactions on **24**(5) (2002) 603–619