Improving the Canny Edge Detector using Automatic Programming: Improving the Filter

Lars Vidar Magnusson
Østfold University College
Email: lars.v.magnusson@hiof.no

Roland Olsson
Østfold University College
Email: roland.olsson@hiof.no

***Abstract*—We have used automatic programming, a machine learning technique related to inductive logic programming and genetic programming, to make the Canny edge detector better at identifying contours in natural images.**

We present an improved version of the filter used in the first stage of the Canny algorithm. We show that the mean performance of the Canny algorithm with the improved filter on a popular test set of natural images has been improved by 1.4%. Our result shows that the heuristic design provides a statistically significant increase in performance—without adding extra processing steps or adding additional information. This suggests that the filter should be used as a standard part of image analysis platforms.

The inferred heuristic filter exhibits an ability to retain detail without sacrificing noise reduction. This is further evidence that automatic programming is well suited for generating heuristics for image analysis problems.

***Keywords*—image analysis; edge detection; feature extraction; automatic programming**

I. Introduction

Edge detection is a fundamental image analysis task providing the input for higher level processing such as object classification. Traditional edge detectors like the Canny edge detector [1] do not distinguish edges occurring around objects from edges occurring within textured areas. There have been proposed algorithms that are better at this, but none that offer a significant improvement without sacrificing runtime efficiency.

Evolutionary Computation (EC) has been used in the past on the problem of edge detection, but to our knowledge it has never been used to improve an existing algorithm like the Canny algorithm. A Genetic Algorithm (GA) setup has been used to evolve edge maps directly [2], and Genetic Programming (GP) has been used to create logic for finding edges using a sliding window approach [3] [4]. Harris and Buxton [5] and Lee et al. [6] have used GP to improve a filter mask like the one used in [1]. Both have used small datasets with a mixture of synthetic and natural signals, and both evaluate using a custom evaluation method.

Automatic programming is a machine learning technique related to inductive logic programming and genetic programming. Automatic Design of Algorithms Through Evolution [7], [8] (ADATE) induce programs by a process inspired by natural evolution. As such it is similar to GP, but unlike GP the search process is systematic. This allows the system to invent both non-trivial recursive patterns and auxiliary functions. It has been established that ADATE is capable of inferring interesting *heuristics* [9]–[12], and that it is well suited for image analysis problems [13]–[15].

In this paper we use ADATE to make the Canny edge detector better at identifying contours in natural images by improving the filter used in the first stage of the algorithm. This overlaps with the work done by both [5] and [6], but there are important differences. We will use a popular dataset of natural images with ground truth annotations by multiple subjects [16], and we will use the F-measure—a well known performance metric in the field of information retrieval. We will also incorporate the entire Canny algorithm, with the ultimate goal of improving the algorithm as a whole.

II. Automatic Design of Algorithms Through Evolution (ADATE)

Automatic Design of Algorithms Through Evolution (ADATE) [7], [8] takes a systematic approach to evolution instead of relying on random mutations, which allows for complex logic that would otherwise be impossible. ADATE has been written in SML [17], and it infers programs in a subset of SML, called ADATE ML.

A. Transformations

The evolution process is driven by four types of transformations. The first type, *Replacement* (**R**), is responsible for replacing an expression in the program. The new expression can either be an entirely new expression, or it can reuse parts of the original. All replacements are tested to identify those that do not make the program worse. These are marked as *replacements preserving equality* (**REQ**s), and they are essential for exploration of plateaus in the search landscape. An *Abstraction* (**ABSTR**) converts an existing expression into a new function and inserts a function call at the location of the source expression. *Case-Distribution* (**CASE-DIST**) and *Embedding* (**EMB**) change the scope of variables and functions, and the domain of an auxiliary function respectively.

None of the transformations except for replacements change the semantics of a program, instead they are grouped together with at least one replacement to form *compound transformations* such as **ABSTR REQ REQ R**.

B. The Overall Search for Programs

ADATE maintains a hierarchical structure called a *kingdom* inspired by Linnean taxonomy [18]. At the top level there are *families* containing *genera* of the same syntactic complexity. Each *genus* contains one or more potential parent programs, and each *species* contain programs that have been created from the same parent program. An essential principle in the organization of the kingdom is that a new program is inserted into the kingdom only if it is better than the existing programs with similar syntactic complexity. When a program is inserted into the kingdom, all programs bigger but not better are removed.

ADATE employs an iterative deepening search strategy where each program has an associated *cost limit* which determines how many programs that should be synthesized from the program. The following steps are repeated until the user decides to terminate.

1) Find the program in the kingdom with the lowest cost limit and syntactic complexity.
2) Synthesize a number of number of new programs based on the program selected and the corresponding cost limit, and insert any program that fulfills the requirement described above into the kingdom.
3) Double the cost limit of the selected program.

III. The Canny Edge Detector

The Canny algorithm [1] goes through three separate stages, where the first stage is the target of our investigation. This stage removes noise and produces a *gradient magnitude* image by convolving the input image with a filter. In the original description, the filter used is the first order derivative of a Gaussian noise reduction filter.

There are a number of implementations of the Canny algorithm available, but we decided to base our work on the implementation in Matlab. The filter stage of the implementation produces two gradient images; one for each dimension. The implementation uses two one-dimensional filters instead of one two-dimensional to save computation time during convolution.

IV. Experimental Setup

The ADATE specification, containing the SML implementation, and other relevant resources used in the experiments can be found at our web site [19].

A. The Target Program

The start program is listed in Fig. 1. This represents all the logic needed to create the two one-dimensional filter masks. We have kept all but the actual value calculations outside the learning environment.

The function takes one argument *m* of type *mode*; a custom data type that specifies what value to calculate and provides the input arguments needed.

The first mode *size* return the number of elements to use. The input argument *Sigma* (σ) is the standard deviation used for the Gaussian noise reduction filter. The default behavior is to return $8\lceil\sigma\rceil$. The second mode *value* has an additional argument i which specifies the position in the Gaussian noise reduction filter mask. The default behavior is

$$value(\sigma, i) = \frac{1}{\sqrt{2\pi}\sigma}\mathrm{e}^{-\frac{i^2}{2\sigma^2}}. \tag{1}$$

```
fun f( m : mode ) : real =
  case m of
    size sigma => 8.0*( realCeil sigma )
  | value( sigma, i ) =>
      ( 0.3989422804014327 * 1.0 / sigma ) *
      exp(
        ~( ( i*i ) /
          ( ( sigma*sigma )+( sigma*sigma ) ) ) )
  | gradBorder( x1, x2 ) => x2-x1
  | grad( x3, x4, x5 ) =>
      ( x5-x3 )/ 2.0
```

Figure 1. The original program written in ADATE ML.

The two remaining modes, *gradBorder* and *grad*, are used for creating the gradient filter mask. The input arguments $x_1 \ldots x_5$ contain values from the Gaussian filter mask. The variables x_1 and x_2 provide the values for the two border cases, and the default behavior is to return $x_2 - x_1$. For the remaining positions, x_3, x_4 and x_5 provide the left, middle and right filter values respectively, and the default behavior is to return $(x_5 - x_3)/2$. The middle value x_4 has been included to allow new programs to use the value if needed.

The final filter masks are normalized so that the Gaussian filter mask sums to one and the gradient filter sums to zero—a restriction that will apply to all synthesized programs as well. The filter masks created by the original program with $\sigma = 3.25$ can be seen in Fig. 2.

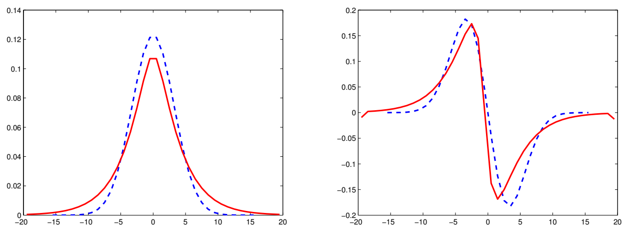

Figure 2. The two filter masks used in the algorithm. The masks generated by the ADATE-improved algorithm in red, and by the original Canny algorithm in dotted blue.

B. The Dataset

We decided to use the Berkeley Segmentation Data Set [16] (BSDS)—a high quality dataset of natural images with ground truths designed for training and evaluating both contour detectors and image segmentation algorithms. The dataset is split into 200 training images, 100 validation images, and 200 test images.

In ADATE, the training set is evaluated at a much higher frequency than the validation set, so we ran the experiments using only 50 of the training images for training, and both the remaining training images and the validation set for validation.

C. Evaluating the Inferred Programs

BSDS [16] contains ground truth annotations by on average 5 subjects for each image. It also provides a benchmark based on a version of F-measure [20] adapted for multiple ground truths.

We incorporated this evaluation method into our experiments, but we decided to use a slightly different approach for evaluating a set of images. In [16] they accumulate the counts and sums for precision and recall over the entire set and calculate the final F-measure score once all images have been evaluated. We instead evaluate the score per image and use the average F-measure on the set. This allows each image to be equally important to the score.

We also decided to reduce the cardinality of the ground truth set during evolution to speed up evolution. We decided on an approach where we select the *best* ground truth, where the *best* is defined as the ground truth that achieves the best score when evaluated against the others in the set.

D. Selecting the Constants

There are three constants that determine the return characteristic of the algorithm; *sigma* is the standard deviation of the Gaussian filter, *high* and *low* are the thresholds used during *hysteresis thresholding* [1]. The constant values were found using a simple grid search.

V. Results

A. The Improved Program

```
fun f m =
  case m of
    size sigma => 10.01 * realCeil sigma
  | value( sigma, i ) =>
        ( 0.398942280401 / sigma ) *
        exp(
          ~ ( ( i * i ) /
              ( ( ( i / tanh i ) * sigma ) +
                ( ( sigma * sigma ) - sigma ) ) ) )
  | gradBorder( x1, x2 ) => tanh( ~ x1 )
  | grad( x3, x4, x5 ) =>
      ( x5 - tanh x4 ) / 2.0
```

Figure 3. The improved program.

The best program found is listed in Fig. 3. The overall structure of the program is the same as the original, but all the modes of the program have been modified. The *size* mode now returns $10.01\lceil\sigma\rceil$, and the *value* mode has been changed into

$$value(\sigma, i) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{i^2}{\frac{i\sigma}{\tanh i}+\sigma^2-\sigma}}. \quad (2)$$

The *gradBorder* mode now returns $\tanh(-x_1)$, and *grad* returns $(x_5 - \tanh x_4)/2$. The combined effect of the changes can be seen in Fig. 2.

B. Benchmarks

Both algorithms were benchmarked using the 200 test images and the original evaluation method used in the BSDS. This allows us to compare our results directly to the results from several other algorithms. We used a simple grid search to find optimized constant values both for the entire test dataset (OD) and for each image (OI).

The results can be seen in Table I; the results with OD constants are listed under *ODF*, and the results with the OI constants are listed under *OIF*. For the sake of comparison we have included the SCG algorithm ([21]), the current best algorithm for the dataset. The ADATE-improved algorithm is still worse than the SCG algorithm, but this is to be expected when considering that the SCG algorithm is significantly more complex, both in terms of information schema and processing.

TABLE I
The Evaluation Results

	ODF	**OIF**
SCG	0.71	0.73
ADATE-Improved	0.615	0.656
Canny	0.606	0.646

The receiver operating characteristics (ROC) curves in Fig. 4 show a noticeable improvement. We have performed both a *paired student-t* test and a Wilcoxon signed rank test, which gave a p-value of 8.348×10^{-5} and 1.164×10^{-5} respectively. We can therefore conclude that the ADATE-improved algorithm is better than the original algorithm on the test set.

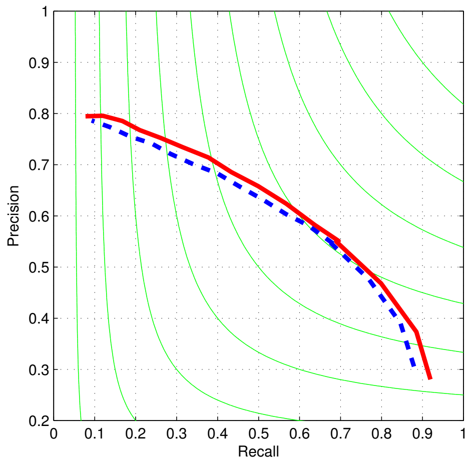

Figure 4. The ROC curve for the original algorithm in dotted blue, and the improved algorithm in solid red.

C. Visual Inspection

We have provided some example images from the test set along with the corresponding edge maps from the two algorithms in Fig. 5. The edges were produced using the OD constants for the respective algorithms.

Based on the two first images it is apparent that the ADATE-improved algorithm is better at retaining the structure of detailed areas. This is especially noticeable in the windows and doors in the first image, and in the umbrella in the second. In the last image we see an example where the improved algorithm is able to reduce false positives in the rocky regions.

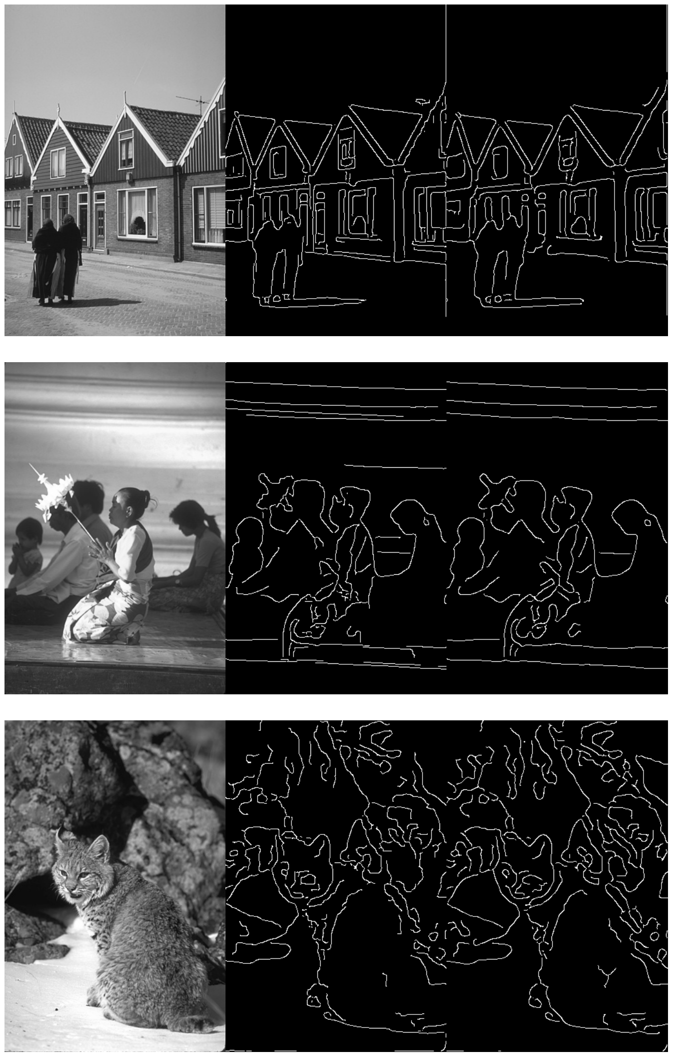

Figure 5. A few example images from the test set where the ADATE-improved algorithm performs better than the original algorithm. The left edge maps belong to the improved algorithm.

VI. Conclusions

We have improved the convolution filter of the Canny edge detector using automatic programming. The performance of the algorithm with the improved filter has increased by 1.4% on a test set of natural images. This is significant since it proves that a heuristic design can outperform a formal design without adding extra information or processing steps. Based on a paired student-t test and a Wilcoxon signed rank test we can say that the improvement is statistically significant.

The new algorithm exhibits an increased ability to both retain detail and smooth over textured areas. It is likely that the new version of the filter could increase the performance of the Canny algorithm in general, but further testing is required. Our findings are further evidence that automatic programming is capable of improving image analysis algorithms.

We are currently investigating the possibility of improving other parts of the Canny algorithm using similar setups, and we are looking into building new theory from the findings. We are also planning to use automatic programming to infer or improve an algorithm for determining the two thresholds automatically, and to find an efficient way of incorporating texture information into the algorithm.

References

[1] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 6, pp. 679–698, 1986.

[2] S. M. Bhandarkar, Y. Zhang, and W. D. Potter, "An edge detection technique using genetic algorithm-based optimization," *Pattern Recognition*, vol. 27, no. 9, pp. 1159–1180, 1994.

[3] R. Poli, "Genetic programming for feature detection and image segmentation," in *Evolutionary Computing*. Springer, 1996, pp. 110–125.

[4] Y. Zhang and P. I. Rockett, "Evolving optimal feature extraction using multi-objective genetic programming: a methodology and preliminary study on edge detection," in *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. ACM, 2005, pp. 795–802.

[5] C. Harris and B. Buxton, "Evolving edge detectors with genetic programming," in *Proceedings of the 1st annual conference on genetic programming*. MIT Press, 1996, pp. 309–314.

[6] M. Lee, S. Leung, and H. Cheung, "Edge detection by genetic algorithm," in *Image Processing, 2000. Proceedings. 2000 International Conference on*, vol. 1. IEEE, 2000, pp. 478–480.

[7] R. Olsson, "Inductive functional programming using incremental program transformation," *Artificial Intelligence*, vol. 74, pp. 55–81, 1995.

[8] R. Olsson, "Population management for automatic design of algorithms through evolution," in *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*. IEEE, 1998, pp. 592–597.

[9] A. Løkketangen and R. Olsson, "Generating meta-heuristic optimization code using ADATE," *Journal of Heuristics*, vol. 16, pp. 911–930, 2010.

[10] R. Olsson and A. Løkketangen, "Improving state-of-the-art 3-sat solvers using automatic design of algorithms through evolution," *Artificial Evolution 2011 (Evolution Artificielle 2011)*, 2011.

[11] R. Olsson and A. Løkketangen, "Using automatic programming to generate state-of-the-art algorithms for random 3-sat," *Journal of Heuristics*, vol. 19, no. 5, pp. 819–844, 2013.

[12] S.-E. Hansen and R. Olsson, "Improving decision tree pruning through automatic programming," in *Proceedings of the Norwegian Conference on Informatics (NIK-2007)(November 2007, Holmenkollen Park Hotel Rica, Oslo)*, 2007, pp. 31–40.

[13] L. V. Magnusson and R. Olsson, "Improving graph-based image segmentation using automatic programming," in *Applications of Evolutionary Computation*. Springer, 2014, pp. 464–475.

[14] K. Larsen, L. V. Magnusson, and R. Olsson, "Edge pixel classification using automatic programming," *Norsk Informatikkonferanse (NIK)*, 2014.

[15] H. Berg, R. Olsson, T. Lindblad, and J. Chilo, "Automatic design of pulse coupled neurons for image segmentation," *Neurocomputing*, vol. 71, no. 10-12, pp. 1980–1993, 2008, neurocomputing for Vision Research; Advances in Blind Signal Processing.

[16] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, pp. 898–916, 2011.

[17] R. Milner, M. Tofte, R. Harper, and D. MacQueen, *The Definition of Standard ML - Revised*. The MIT Press, 1997.

[18] C. Linnaeus, *Systema naturae per regna tria naturae secundum classes, ordines, genera, species,...* impensis Georg Emanuel Beer, 1788, vol. 1.

[19] L. V. Magnusson, "Image analysis & machine learning," 2016-07-7. [Online]. Available: http://www.it.hiof.no/iaml/

[20] C. J. V. Rijsbergen, *Information Retrieval*, 2nd ed. Newton, MA, USA: Butterworth-Heinemann, 1979.

[21] R. Xiaofeng and L. Bo, "Discriminatively trained sparse code gradients for contour detection," in *Advances in neural information processing systems*, 2012, pp. 584–592.