Improving the Canny Edge Detector Using Automatic Programming: Improving Non-Max Suppression

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

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

ABSTRACT

In this paper, we employ automatic programming, a relatively unknown evolutionary computation strategy, to improve the non-max suppression step in the popular Canny edge detector. The new version of the algorithm has been tested on a dataset widely used to benchmark edge detection algorithms. The performance has increased by 1.9%, and a pairwise student-t comparison with the original algorithm gives a p-value of 6.45×10^{-9}. We show that the changes to the algorithm have made it better at detecting weak edges, without increasing the computational complexity or changing the overall design.

Previous attempts have been made to improve the filter stage of the Canny algorithm using evolutionary computation, but, to our knowledge, this is the first time it has been used to improve the non-max suppression algorithm.

The fact that we have found a heuristic improvement to the algorithm with significantly better performance on a dedicated test set of natural images suggests that our method should be used as a standard part of image analysis platforms, and that our methodology could be used to improve the performance of image analysis algorithms in general.

1. INTRODUCTION

The Canny edge detector [3] is a popular algorithm that can be found in most image analysis platforms. The algorithm is based on the gradient image, which is calculated by convolving the image with a filter designed to approximate either the first or second order derivatives. This is a common approach in traditional edge detectors, so the success of the Canny algorithm can be attributed to the other stages of the algorithm; *non-max* suppression and *hysteresis* thresholding.

It is a well known fact that traditional gradient based edge detectors like the Canny algorithm can suffer from poor performance in textured image regions. The best algorithms today handle this issue by incorporating more complex information and processing schemas. There is also an increased tendency to utilize some form of machine learning strategy to help distinguish between different textures [1, 17].

There have been several attempts at improving the Canny algorithm. Wang and Fan [16] introduced an adaptive filter to replace the standard filter in the algorithm. They also introduced an additional noise reduction technique, and they have replaced non-max suppression with *morphological thinning*. They demonstrate the performance by qualitative analysis of a single image and a simple evaluation function. Ding and Goshtasby [4] proposed an improvement to the *non-max suppression* stage in the Canny algorithm based on an observation that the original approach will incorrectly suppress corner edges. They introduce an extra processing stage to identify what they refer to as *minor* edges, which are joined together with the edges produced by the original Canny algorithm. There have also been attempts to create an adaptive threshold mechanism for the Canny algorithm [7].

Evolutionary computation (EC) strategies such as genetic algorithms (GA) and genetic programming (GP) have been used to evolve improved filters [6] and to infer logic filters for sliding windows [14, 18], but, as far as we know, it has never been used to evolve improved solutions for the remaining stages in the Canny algorithm.

We want to investigate the possibility of making the *non-max suppression* stage in the Canny edge detector better at suppressing edges in textured areas, and to increase the ability of the algorithm to identify contours around objects. We aim to do this without adding additional information or processing—to make the original algorithm better by changing the internal calculations. We have employed automatic programming (AP), a relatively unknown evolutionary computation strategy related to genetic programming. AP has the ability both to invent brand new solutions and to improve existing algorithms by evolving suitable heuristics, and it has been used with success on similar problems in the past [2, 10, 8].

2. AUTOMATIC PROGRAMMING

Automatic programming is like Genetic Programming (GP) an evolutionary computation strategy, but, unlike GP, automatic programming systems use a systematic search process rather than a random one. A major benefit of this approach is the ability to manage complex code like loops and recursive patterns in a better way. In this paper we have employed Automatic Design of Algorithms Through Evolution (ADATE) [12, 13] which has been continuously developed

GECCO '16, July 20 - 24, 2016, Denver, CO, USA

ISBN 978-1-4503-4206-3/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2908812.2908926

over the last two decades.

The ADATE system infers and evaluates new programs based on *problem specifications* written in a combination of Standard ML [11] and ADATE ML, where ADATE ML is a minimum subset of SML used for evolution to keep the syntactic complexity to a minimum.

2.1 The Evolutionary Search Strategy

ADATE maintains a hierarchical structure called a *kingdom* that contains the individuals during evolution. The structure is inspired by Linnean taxonomy [9], and it has three levels; *families*, *genera* and *species*. A *family* is a collection of *genera* that has comparable syntactic complexities. A *genus* is a set of one or more potential parent programs. A *species* is a group of programs created from the same parent program in one of the *genera*.

At each step in the evolution the individual in the kingdom with the lowest *cost limit* and syntactic complexity is selected. The *cost limit* associated with each program controls the number of programs to create from a parent program, and it is doubled every time a program is expanded. Based on the *cost limit* and syntactic complexity of the selected program a number of new programs are synthesized by transforming the program using *compound transformations*. Each new program is evaluated to determine whether it should be inserted into the *kingdom* or not. The search process then proceeds to select the next program to expand, and the process continues in this manner until the user terminates.

A program is inserted into the kingdom if it is smaller and better than any of the other individuals already in the kingdom. When a program is inserted, all programs that are bigger without being better are removed. The result is a tree structure with genealogical chains of gradually bigger and better programs.

2.2 Program Transformations

Programs are transformed using so called *compound* transformations, which are sequences of *atomic* transformations. There are four *atomic* transformation types; *replacement* (R), *abstraction* (ABSTR), *case-distribution* (CASE-DIST), and *embedding* (EMB).

All of the *atomic* transformation types maintain the semantics of the parent program except for Rs. This makes them essential to the evolution of new individuals, but also makes them the most computationally challenging. The system infers replacement transformations by searching through type correct replacements of increasing complexity. All replacements are tested to identify replacements that do not make the program worse. Replacements that fulfill this requirement are marked as *replacements preserving equality* (REQs), and they play an important role in the exploration of plateaus in a fitness landscape.

New auxiliary functions are created by the ABSTR transformation. A source expression is extracted from the program and inserted into a new function with an appropriate domain. A function call to the new function, along with the corresponding arguments, is inserted in the location of the source expression. The two remaining transformation types, CASE-DIST and EMB, are responsible for changing the scope of function and variables, and for changing the domain of an auxiliary function, respectively.

Compound transformations group together *atomic* transformations according to a set of pre-calculated *forms* designed to create sequences of transformations that are aimed towards a common target. To illustrate this concept consider the example of a R following an ABSTR. The ABSTR creates a new function, and the R will be applied to the right hand side of the new function. With this setup, only the first transformation in a compound transformation can chosen freely; the remaining transformations are restricted to the ones allowed by the pre-calculated forms.

Figure 1: A test image similar to the one used by Ding and Goshtasby [4] and the result of running the Matlab implementation of the Canny algorithm on the image.

3. THE CANNY EDGE DETECTOR

The Canny edge detector [3] was introduced three decades ago, and it is still the standard detector in most image analysis platforms. In essence, it was designed to disregard discontinuities caused by noise and to give a single response to each edge. In this respect, it is an effective algorithm, both in terms of the quality of the response and the processing time needed.

There are three major steps in the algorithm. The first step is responsible for producing the gradient image, which is calculated by convolving the input image with one or more filters both to remove noise and to find the differences. The gradient image provides the input to *non-max suppression*—the target of our investigation. The final step is called *hysteresis thresholding*. It uses the output of *non-max suppression* to identify the *strong* and *weak* edges, corresponding to pixels greater than a *high* and *low* threshold respectively. The final edge image contains all the *strong* edges in addition to all 8-connected *weak* edges.

3.1 Non-Max Suppression

Non-max suppression was designed to reduce multiple responses to a single edge [3], and it has played an essential role in the success of the Canny edge detector. So much so that the step has become a standard post-processing step for edge detectors in general.

The idea is to suppress gradient magnitudes that are less than either of the magnitudes along the gradient angle. There are several ways this could be done, but due to both the quality and the popularity of the implementation we have decided to use the implementation in Matlab as a reference.

We tested the implementation to see if it has problems with corner edges as observed by Ding and Goshtasby [4], and the results can be seen in Figure 1. While it is clear that a few pixels have been erroneously suppressed, the problem is not as significant as illustrated in [4].

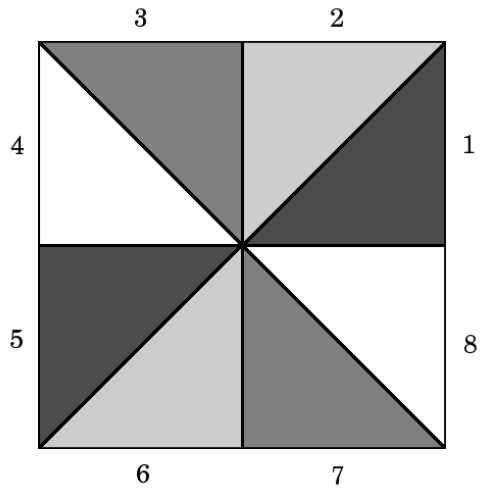

Figure 2: The eight possible sectors for the gradient angle. Two and two sectors are grouped together to form 4 possible configurations.

The Matlab implementation generates two gradient images, one for each axis. These are used both to find the final magnitude and to determine the angle of a gradient. The angle belong in one of the eight sectors that can be seen in Figure 2. The angles in sector 1 are equivalent to the ones in sector 5, the angles in sector 2 with the ones in sector 6, and so on. For the purpose of this discussion we define the x-axis to point right, and the y-axis to point up. If the y gradient d_y is positive and the x gradient d_x larger than d_y, or if d_y is negative and d_x is less than d_y, the gradient angle is in sector 1 or 5 respectively. In this case, the neighbor magnitudes are calculated by means of linear interpolating between neighboring magnitudes with the interpolating parameter t set to d_y/d_x. The first neighbor magnitude is found by interpolating between the right and upper-right neighbors, and the second neighbor by interpolating between the left and the lower-left. The tests to determine if a gradient magnitude falls into the remaining sectors and the corresponding linear interpolations between the neighbor magnitudes are conducted in a similar manner.

4. EVOLUTIONARY PREREQUISITES

The evolutionary prerequisites presented in this section define the world in which new programs will be evolved by the ADATE system. The final problem specification, along with all the relevant resources, can be found on our web site.

4.1 The Image Dataset

There are several available datasets that could be used to train object contour detection, but the BSDS500 dataset [1] stand out when it comes to the quality of the ground truth annotations. It has also been widely used as a benchmark for both edge detectors and image segmentation algorithms.

The BSDS500 is a dataset of 500 natural image with ground truth annotations by multiple human subjects, and the images feature a number of different motives and scenarios. There are 200 training images, 100 validation images and 200 test images, both in grayscale and color. We have selected to use only the grayscale images in our experiments,

```
fun f( d1, d2, m, m1, m2, m3, m4 ) =
  let
    fun lerp( ( x, y, t ) =
      x*( 1.0-t )+y*t
  in
    case abs( d1/d2 ) of t =>
    case lerp( m1, m2, t ) of tm1 =>
    case lerp( m3, m4, t ) of tm2 =>
    case m < tm1 of
      false => (
        case m < tm2 of
          false => m
        | true => 0.0 )
    | true => 0.0
  end
```

Figure 3: The base individual written in ADATE ML.

since this will help keep the runtime low and speed up evolution. This can be changed in future experiments to allow the ADATE system access to more information to utilize in the evolved programs.

ADATE evaluates the performance of the programs on the training set at a higher frequency than on the validation set. Previous experience with using the dataset has shown good overfitting characteristics, which has allowed us to use a smaller training set than intended by the dataset creators. We used 50 of the training images for training. The remaining 150 training images were used along with the 100 validation images for validation.

4.2 The Fitness Function

The fitness function used for the experiments is inspired by the benchmark provided in the BSDS500 [1]. They use the popular *F-measure* [15] to evaluate the performance of edge detectors by accumulating the *precision* and *recall* counts over all the images, and calculates the *F-measure* based on the total counts. While this is a reasonable approach, we decided instead to use the average *F-measure*. This makes each image equally important to the final score, regardless of the number of edge pixels in the image.

The dataset contains on average 5 ground truth annotations for each image. Determining whether or not a proposed edge pixel matches an edge pixel in one of the ground truth annotations is reduced to an assignment problem and solved using the CSA algorithm [5]. However, due to the size of the graphs, evaluating an edge map against a single ground truth is significantly slower than finding the edges. We therefor decided to minimize the time needed to evaluate each image by using only the *best* ground truth during evolution. We define the *best* ground truth to be the ground truth with the highest F-measure when evaluated against the other ground truths in the set. This approach was used during evolution to speed up the process, but the final evaluation of a program was conducted using the entire ground truth set.

4.3 The Base Individual

The base individual for the original non-max algorithm is listed in Figure 3. The program is invoked once per pixel in the gradient image. The arguments passed to the program depend on the gradient angle. The are four possible con-

figurations, corresponding to the eight possible directions that can be seen in Figure 2. Determining which of the four configurations that apply to the current position in the gradient image is left outside the evolution environment. This ensures that the overall structure of the algorithm maintains the same.

To illustrate the alternative argument configurations for a particular position we will give two examples, representing two different variants. In the first configuration, where the gradient angle is in either sector 1 or 5, $d1$ and $d2$ will contain the gradient for the y and x direction respectively. The parameter m will contain the magnitude of the current pixel, and the remaining parameters $m1$, $m2$, $m3$ and $m4$ will contain the magnitude of the right, the upper-right, the left and the lower-left neighbor respectively. In the second configuration, where the gradient angle is in either sector 2 or 6, the $d1$ and $d2$ parameters have switched contents so that $d1$ contains the gradient in the x direction and vice versa. The m argument is still the magnitude of the current pixel, but $m1$, $m2$, $m3$ and $m4$ have changed to the magnitude of the upper, the upper-right, the bottom and the bottom-left neighbor. This pattern is repeated for the two remaining configurations, i.e., $d1$ will contain the lower of the two axis gradients, $m1$ and $m3$ will contain the magnitudes of the axis-aligned neighbors and $m2$ and $m4$ will contain the magnitude of the diagonal neighbors.

4.4 Functions and Constructors

We decided to restrict the functions and constructors available during evolution to include only the functions and constructors already present in the base individual. The only exception is the inclusion of the hyperbolic tangent function, which has proved to be useful in several experiments in the past.

4.5 The Constants

The return characteristic of the Canny algorithm is dependent on three constants. The first is the standard deviation to use for the noise reduction filter in the first stage of the algorithm, and the remaining two are the *high* and *low* thresholds used during *hysteresis* thresholding. Note that none of these are used directly in the base individual, or in the non-max algorithm, but they still affect the overall performance. We evaluated different values for these constant using both the training and validation images along with a simple grid search algorithm to find the configuration used during evolution.

5. RESULTS

In this section the ADATE-improved non-max suppression algorithm is described and analyzed, and we present the results of running the Canny algorithm with the improved non-max suppression algorithm on a popular test set.

5.1 ADATE-Improved Non-Max Suppression

The improved individual can be seen in Figure 4. At first glance it is quite similar to the original program listed in Figure 3, but there are several important differences.

The ADATE-improved algorithm is slightly smaller than the original. The first reason for this is that the calls to *lerp* have been moved into the *case*-expressions which are dependent on their return. This is just a syntactic rewrite and has no effect on the performance of the algorithm. The

```
fun f( d1, d2, m, m1, m2, m3, m4 ) =
let
  fun lerp( x, y ) = x*( 1.0-m3 ) + y*m3
in
  case m < lerp( m1, m2 ) of
    false => (
      case m < lerp( m3, m4 ) of
        false =>
          abs( m/tanh( m/d2 ) )
      | true => 0.0 )
  | true => 0.0
end
```

Figure 4: The improved program.

second reason however is that the calculations of t—the interpolation parameter in the original algorithm—has been removed. The calculation performed by *lerp* no longer uses t to interpolate between the values, but instead uses the contents of the $m3$ parameter. This has a profound effect on the behavior of the algorithm, as we will discuss below. The final change is the return value when the current magnitude m is bigger than both neighbors. The original behavior is to return m, but this has been changed to $m/(\tanh(m/d2))$, which has introduced a dependency on $d2$—the largest of the two axis gradients.

In order to make the difference between the two implementations more clear we have plotted the differences of the most significant values, when running on three separate images with OD constants, in Figure 6. The histograms show the distributions of $m3$ in red and t in blue. Based on the histograms we can see that the $m3$ parameter is close to normally distributed, while t is considerably more uniform. It is clear that on average $m3$ is significantly smaller than t. This has the effect that the linear interpolation will prioritize the axis-aligned neighbors over the diagonal, which in turn will cause fewer pixels to be suppressed—particularly gradients with angles close to the diagonal.

The scatter plots illustrate the difference between m and $m/(\tanh(m/d2))$. There is considerable correlation between the two, but the latter is slightly larger than the former. The consequence is that the magnitudes that pass the ADATE-improved non-max suppression will on average be slightly larger than in the original algorithm. The variation between the two is caused by the denominator $\tanh(m/d2)$. The value of this expression will increase as $m/d2$ increases, which depends on the other axis gradient $d1$. The result is that a non-suppressed value will be largest when the gradient angle is axis aligned and reduced when approaching the diagonal.

This is an extremely interesting heuristics. It will suppress fewer pixels with steep angles due to the change in interpolation, and at the same time reduce the resulting magnitude of these pixels due to the change in the return of non-suppressed magnitudes.

5.2 Performance

The performance of the algorithms have been measured on the test set in BSDS500 [1]. We have copied their evaluation function in order to for the results to be comparable to previous results on the dataset. We test each of the algorithms using two constant configurations; one optimized for the entire dataset (OD), and one optimized for each image

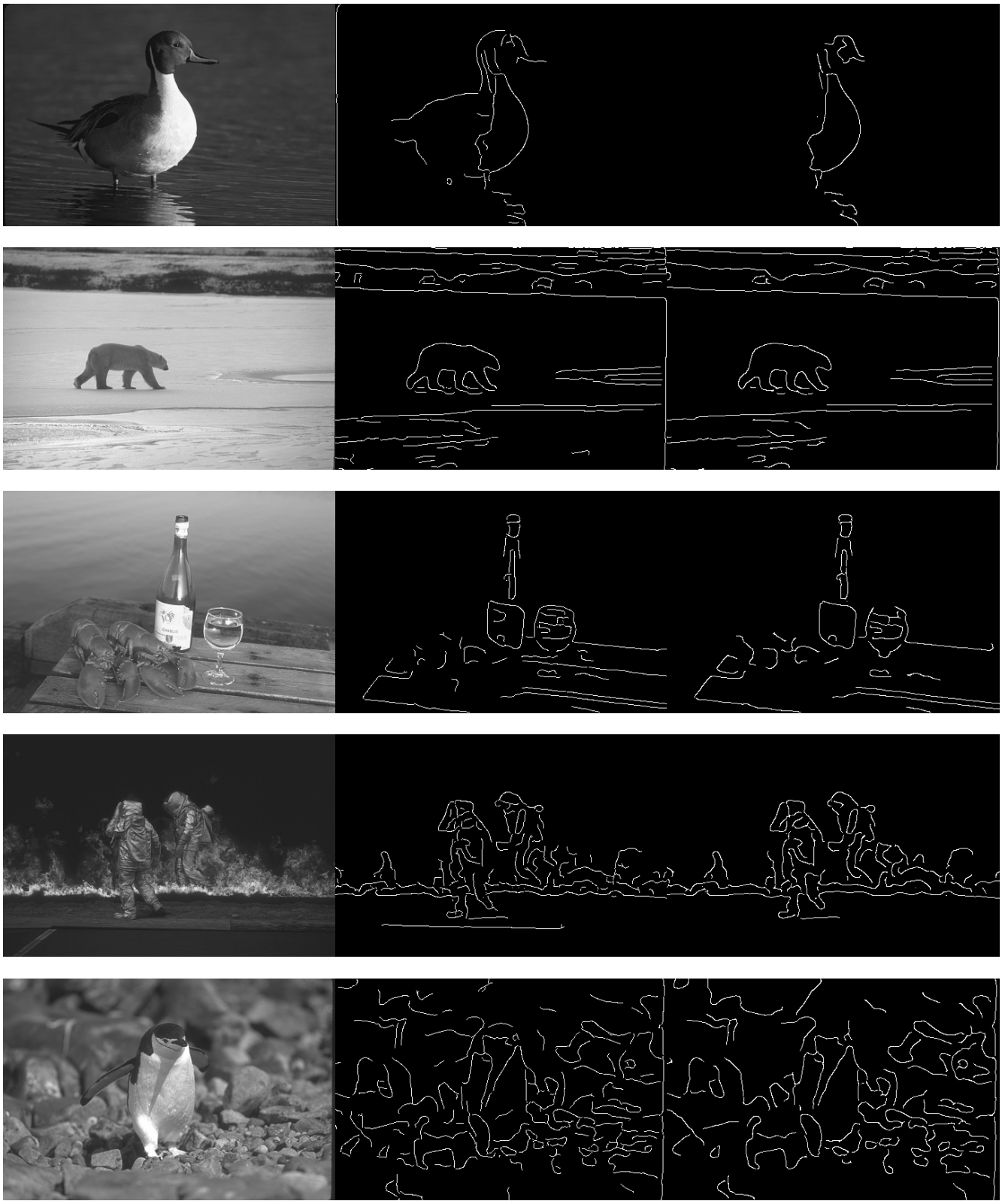

Figure 5: A collection of images and corresponding edge maps where the ADATE-improved algorithm is better than the original algorithm. The edge maps produced with the ADATE-improved algorithm are in the middle.

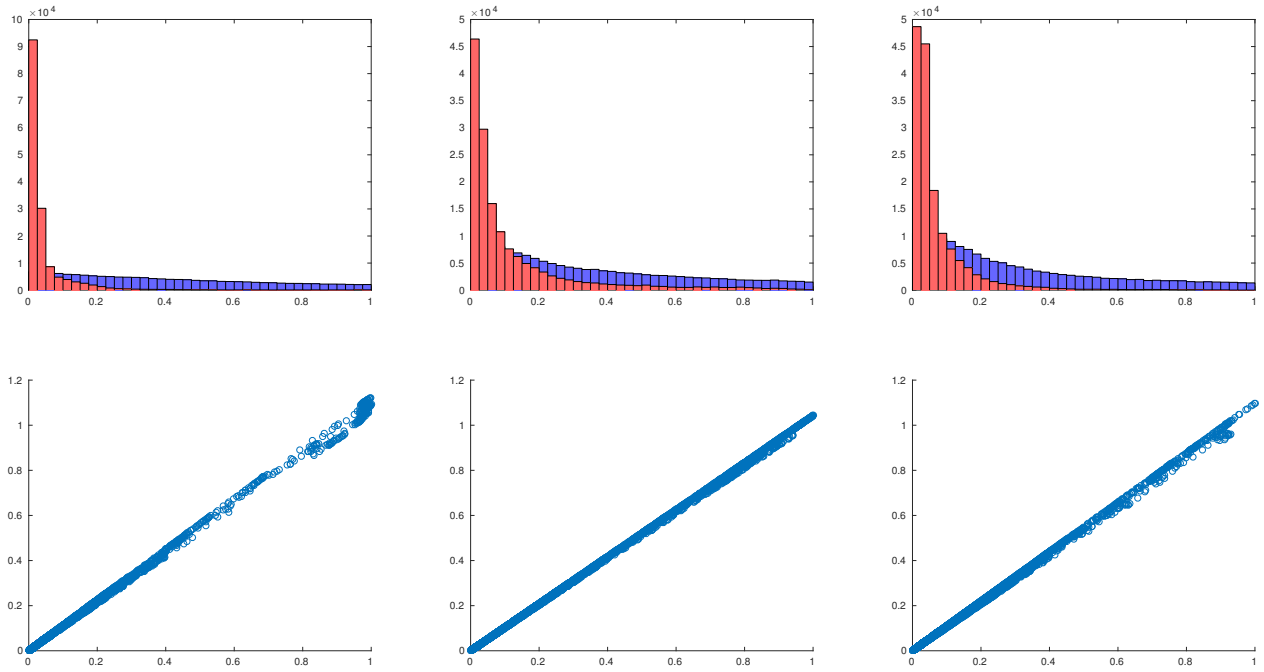

Figure 6: Histograms and scatter plots of the differences between the two algorithms on three images. The histograms contains the distribution of $m3$ in red and t in blue. The scatter plots compares m to $m/(\tanh(m/d2))$. The three images used are the same as the three first images in Figure 5.

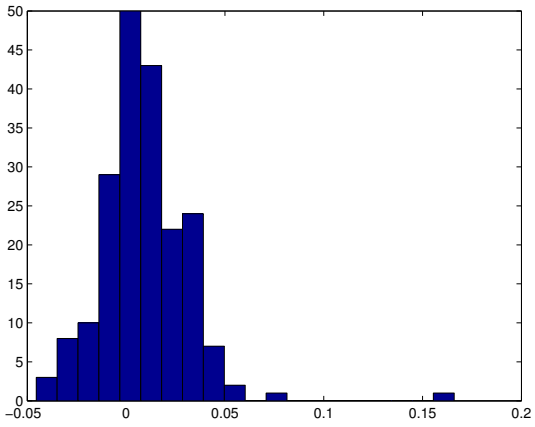

Figure 7: An histogram plot of the difference in F-measure between the two algorithms on each of the images.

(OI). The results for the two configurations can be seen in Table 1. The F-measure with OD constants is listed under ODF, and the F-measure with OI constants is listed under OIF. We have included the corresponding results of the best known algorithm for the dataset, the SCG algorithm [17].

Table 1: The Evaluation Results

	ODF	OIF
SCG	0.71	0.73
ADATE-Improved	0.618	0.657
Canny	0.606	0.652

The ADATE-improved algorithm has been improved by 1.1 percentage points or 1.9% with OD constants, and by 0.5 percentage points or 0.8% with OI constants. The improvement is reduced to about half using OI constant, which can be interpreted as being less dependent on the constants used, but it can also be interpreted as being optimized for the conditions under which the program was evolved. More testing is required to reach a definite conclusion. The performance gap up to the SCG algorithm is to be expected when considering that the SCG algorithm uses considerably more information and processing, but the ADATE- improved algorithm has closed about one fifth of the gap with OD constants.

We have performed two statistical tests to ensure that the improvement is statistically significant. Both algorithms were tested using their respective OD threshold setup. The differences in F-measure between the two algorithms can be seen in Figure 7. The distribution is close to a normal distribution, so for the first test we used a paired student-t test.

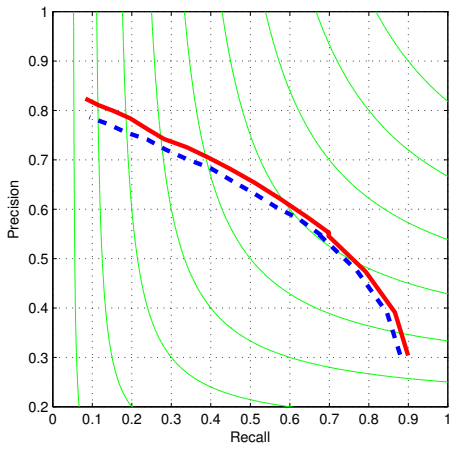

Figure 8: The receiver operating characteristics of the two algorithms. The ADATE-improved algorithm is in solid red.

This gave a p-value of 6.45×10^{-9}. We also performed a Wilcoxon signed-rank test, in case our assumption of normally distributed differences is wrong. This gave a p-value of 1.649×10^{-9}.

The receiver operating characteristic curves for the original Canny and the ADATE-improved algorithm can be seen in Figure 8. The ADATE-improved algorithm is consistently better in the entire range. The *precision* characteristics is significantly better in parts of the curve. The *recall* characteristics however is only slightly better. Based on this we can draw the conclusion that the improved algorithm is slightly better at finding edges, but that it makes noticeably fewer mistakes.

5.3 Visual Analysis

We have included a collection of images where the ADATE-improved algorithm outperform the original algorithm in Figure 5. We also have included the corresponding edge maps for both the algorithms for visual analysis. We used the OD constants for both algorithms to produce the edge maps.

The improvements made to the algorithm are easily noticeable in the first image in the collection. The improved algorithm has correctly identified a large part of the duck, while the original algorithm has only found the most prominent edges.

The improvements in the edge maps of the remaining images are far less visible, but significant nevertheless. The improvements in the second image have been in the details of the ice and landscape. In the third image the improved algorithm has identified more of the glass and some of the details in the label of the bottle. The edge on the floor in front of the men in the fourth image has been partly identified, and so has more of the penguin in the last image.

6. CONCLUSIONS

We have successfully used automatic programming to improve the non-max suppression stage in the Canny edge detector. We have evaluated the improved algorithm on a popular benchmark for edge detectors, and the performance has increased with 1.9%. Based on a paired student-t test and a Wilcoxon signed-rank test we can say that the improvement is statistically significant.

The improved algorithm contains novel and interesting heuristics that have made the overall algorithm better at identifying weak edges. This is an impressive feat considering the size of the changes—the overall design of the algorithm has been maintained. Two minor changes have caused the algorithm to suppress fewer gradients with angles close to the diagonal, and to boost the gradients that pass suppression by an amount inversely proportional to how close the gradient angle is to the diagonal.

The fact that we have used automatic programming to infer interesting heuristics that enhance the performance of the non-max algorithm on a set natural images is further evidence that automatic programming is ideally suited for image analysis problems.

We are planning to use the same methodology to improve *hysteresis thresholding*, the last step in the Canny edge detector, and we want to investigate the possibility of inferring a new algorithm for automatically determining the thresholds. There is also a possibility of trying to expand on the algorithm by including additional information to help out with textured regions.

7. REFERENCES

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

[2] H. Berg, R. Olsson, T. Lindblad, and J. Chilo. Automatic design of pulse coupled neurons for image segmentation. *Neurocomputing*, 71(10-12):1980–1993, 2008. Neurocomputing for Vision Research; Advances in Blind Signal Processing.

[3] J. Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.

[4] L. Ding and A. Goshtasby. On the canny edge detector. *Pattern Recognition*, 34(3):721–725, 2001.

[5] A. V. Goldberg and R. Kennedy. An efficient cost scaling algorithm for the assignment problem. *Mathematical Programming*, 71(2):153–177, 1995.

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

[7] Y.-K. Huo, G. Wei, Y.-D. Zhang, and L. nan Wu. An adaptive threshold for the canny operator of edge detection. In *Image Analysis and Signal Processing (IASP), 2010 International Conference on*, pages 371–374, April 2010.

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

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

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

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

[12] R. Olsson. Inductive functional programming using incremental program tranformation. *Artificial Intelligence*, 74:55–81, 1995.

[13] 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*, pages 592–597. IEEE, 1998.

[14] R. Poli. Genetic programming for feature detection and image segmentation. In *Evolutionary Computing*, pages 110–125. Springer, 1996.

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

[16] B. Wang and S. Fan. An improved canny edge detection algorithm. In *2009 second international workshop on computer science and engineering*, pages 497–500. IEEE, 2009.

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

[18] 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*, pages 795–802. ACM, 2005.