

Computationally Efficient Quantitative Testing of Image Segmentation with a Genetic Algorithm

H. Al-Muhairi, M. Fleury and A. F. Clark
University of Essex, Computing and Electronic Systems Department,
Wivenhoe Park, Colchester, United Kingdom CO4 3SQ
Email: hoalmu,fleum,alien@essex.ac.uk

Abstract—Quantitative testing of segmentation algorithms implies rigorous testing against ground truth segmentations. Though under-reported in the literature, the performance of a segmentation algorithm depends on the choice of input parameters. The paper reports wide variety both in evaluation time and segmentation results for an example mean-shift algorithm. When testing extends over an algorithm’s parameter space, then the search for satisfactory settings has a considerable cost in time. This paper considers the use of a genetic algorithm (GA) to avoid an exhaustive search. As application of the GA drastically reduces search times, the paper investigates how best to apply the GA in terms of initial candidate population, convergence speed, and application of a final polishing round. The GA parameter search forms part of a three-component computation environment aimed at automating the search and reducing the evaluation time. The first component relies on scripted testing and collation of results. The second component transfers to a commodity cluster computer. And the third component applies a genetic algorithm to avoid an exhaustive search.

I. INTRODUCTION

The contribution of this paper is to propose a computationally efficient environment for conducting quantitative algorithm testing, with image segmentation as an example. A harness in a scripting language provides a structured route to testing, in such a way that various evaluation metrics can be applied. To improve the speed of evaluation a cluster computer can act as a throughput engine. However, this assumes a full or exhaustive search is made, whereas the search can be optimized by using a genetic algorithm (GA) as a convenient tool. This paper provides guidance on how to apply a GA to the problem of finding input parameters to a segmentation algorithm. The given parameters should improve the segmentation according to an objective or cost function.

One of the problems with evaluation is that there are frequently multiple parameter settings, whereas the settings used are not always reported in the literature. Therefore, a range of parameters have to be cycled through to establish the best values. This considerably adds to the complexity of the task. To the best of the authors’ knowledge, this paper is amongst the first to consider the importance of parameter setting and the impact it has on the testing regime. An effective graphical method of displaying the parameter space of a particular algorithm is introduced and examples based on the mean-shift algorithm [1] illustrate the techniques. It should be carefully noted that the work in [1] does include parameter

settings and analysis of their effect, though with no hard recommendation for any particular setting.

Section II of this paper outlines our three component environment (scripting, cluster computer, genetic algorithm) for algorithm evaluation. Section IV considers other work on the quantitative approach to algorithm testing. Section V reports investigations into the parameter space of algorithms, while Section VI concludes the paper.

II. TESTING USING AN EVALUATION HARNESS

The first component of the evaluation environment is a scripting language harness or structure for running tests. The harness is written entirely in `Tcl`, a high-level scripting language [2][3], which means it runs essentially unchanged on all flavors of Unix (including Linux and MacOS X) and on Windows operating systems. This Section outlines how the harness is typically used, and then explains how the program works. The harness has the potential for entirely decentralized testing: tests and datasets are downloaded over the Internet as required and executed locally and securely.

A. Operating modes

There are two major ‘modes’ of operation, the first of which runs the tests against the program under test, while the second analyzes the results. This separation is deliberate as it allows different analyzes to be performed without necessitating time-consuming re-execution of the tests; and comparison is greatly facilitated, as there is a test-by-test record of the results.

Running tests In ‘run mode,’ one invokes the harness by specifying a test script on the command line, capturing its output into a file, as in:

```
harn -run hdig.harn >hdig.out
```

This may be a local file or the URL of a remote test script (only HTTP is supported). If the test script is not specified precisely, the harness will try to find it in a series of pre-defined locations; this includes a directory on the local machine and the script repository on the harness web-site. Files downloaded over the network are cached locally; by default, scripts are retained for a week and data files for a year.

The harness outputs a “transcript” which concisely summarizes each test and the output of the program under

test. The transcript also includes the name and version of the test script, the cost function (discussed below), and the elapsed time for performing the tests; the latter is not used by the harness but is often of interest to algorithm developers. The subsequent analysis stage works solely with transcript files.

Evaluating segmentation algorithms Given a transcript file, one assesses performance simply by running the harness in its default ‘evaluate’ mode:

```
harn hdig.out
```

When used this way, the harness reports evaluation statistics, which are dependent on the cost function. An alternative is to invoke the harness with more than one transcript:

```
harn hdig.out otheralg.out http://loc/hdig.best
```

In this case, the harness compares each algorithm with each other and reports the probability that the algorithms achieve statistically significant differing performances. Receiver operating characteristic (ROC) compare True Positive and False Positive counts [4] (Precision-Recall curves [5] are an ROC variant). As ROC curves give no assessment as to the statistical significance of any reported differences, another method such as McNemar’s test, which is a modified χ^2 test, may be preferred. Again there is a logistical implication, as typically more than thirty tests are needed before the results approach a normal distribution.

B. Working with the harness

Interfacing to the algorithm under test

The algorithm developer writes a short piece of Tcl code, called the *interface script*, which interfaces the segmentation algorithm under test to the harness. Generally speaking, this code will invoke an external program that implements the algorithm with specific input files and/or parameters and perform any ‘massaging’ of the program’s outputs to get them into the required form. The interface script is loaded by the harness when it is executed, and it is invoked once for each test performed.

Specifying the tests.

The test script is also written in Tcl as it may be desirable to perform computation in generating either the actual tests or in specifying the locations of any data files it may use. Downloading and executing code is, of course, a dangerous thing to do with the current Internet, so the test script is not executed by the harness itself but rather passed to a slave Tcl interpreter in which all operations that might affect the local system have been removed; this is similar to the ‘sandbox’ environment of Java interpreters in web browsers. The code executing in the slave interpreter is able to perform tests by a callback to a single routine in the main interpreter.

The test script should define two Tcl procedures, named as follows:

`harn_TestInfo`: This is invoked with one of a set of keywords and is expected to return the corresponding value. The harness uses this to obtain ancillary information about the

tests, such as the number of tests and the cost function to be used.

`harn_RunTests`: This performs the actual tests and should invoke the procedure `harn_Test` for each test.

The user must also specify a cost function (or equivalently evaluation method) that forms the basis of the comparison. Though some cost functions are built in, there is also a mechanism for providing alternative cost functions.

III. REDUCING EVALUATION TIME

A cluster computer, the second component of the evaluation environment represents an accessible resource upon which algorithmic testing takes place in batch mode. Testing is otherwise a laborious operation if only a desktop computer is available. A distributed version of the harness is able to reduce the turnaround time when testing an algorithm. The distributed version runs on a cluster, using static load-balancing and process spawning through the `rsh` utility. The harness acts as a ‘throughput engine’ delivering jobs on demand, on a per job basis. A job is defined as a set of one or more tests or applications of an algorithm to one image.

The cluster computer employed by us consists of thirty-seven processing nodes connected with two Ethernet switches. Each node is a small form factor Shuttle box (Model XPC SN41G2) with an AMD Athlon XP 2800+ Barton core (CPU frequency 2.1 GHz), with 512 KB level 2 cache and dual channel 1 GB DDR333 RAM. The nodes are connected via two 24 port unmanaged Gigabit (Gb) Ethernet switches manufactured by D-Link (model DGS-1024T). Each switch is non-blocking and allows full-duplex Gb bandwidth between any pair of ports simultaneously.

The cluster computer software environment is as important as the physical hardware. To make maintenance and cluster-wide propagation of configuration changes easier, at boot time all nodes transfer their root file system from a file server to the local disk, while other file systems are accessed via the Network Filing System. The development branch of Debian sid (unstable but more current) serves to manage upgrades to the Linux operating system. To (re)build nodes `tftp` and `udpcast` (both Linux utilities) are employed in a manner akin to a simplified `SystemImager` (software that automates Linux installs, software distribution, and production deployment). A customized script is available to synchronize packages and configuration with the clusters master template, whenever it is inconvenient to reboot and rebuild. This operation is simply accomplished through an Advanced Packaging Tool cache that is shared between nodes and a shared `debconf db` (a configuration database), and runs in parallel across the cluster by means of `dsh` (an implementation of a wrapper for executing multiple remote shells, such as the `rsh` or `remsh` or `ssh` commands). Nodes rarely need to be added or removed, though there is a script to do that too. The cluster is monitored with the scalable, distributed system `Ganglia` (refer to <http://ganglia.info>).

As a third component of the environment, a Genetic Algorithm (GA) [6] search module lowers the processing time for

the search as a whole. The GA has been integrated within the harness, with a particularly use in optimizing the selection of parameters. As is well-known, a GA emulates to some extent the supposed process of genetic evolutionary adaptation. For example, the algorithm employed allows mutation of variable values as a means of preventing chromosomes becoming too close to each other and remaining in local minima. While in genetics a chromosome is a molecular package containing genes, in a GA a chromosome becomes a vector containing a set of variable elements.

The first stage of the GA is to create a population of chromosomes for a predetermined population size. Given the range of parameter values, random values of each parameter make up the genes of each chromosome. The GA employed in this paper is a real-valued GA following from Polheim's GEATbx, a GA toolbox for Matlab [7] (though the output may be integer-valued parameter settings). In the second stage of the GA, pairs of parent chromosomes are selected according to a fitness or cost function value. Then the number of siblings a parent acquires is statistically proportional to the weighting of that value. For segmentation, the cost function's value is determined by evaluation of the segmentation. In the third stage, each pair of parents is combined to reproduce itself with two child chromosomes. The genes of the chromosome pairs are firstly recombined and then mutated. GEATbx employs extended intermediate recombination [8], wherein an offspring gene g_O is conceived from its two parent genes g_1 and g_2 by $g_O = \alpha g_1 + (1 - \alpha)g_2$, where α is a scaling factor in the range $[-d, 1 + d]$. Assuming that the N genes in a chromosome form an N -dimensional hypercube then, when $d = 0$, g_O lies along the straight line between g_1 and g_2 ; if $d > 0$, extrapolation is permitted. In this work, $d = 0$. Real-valued mutation then takes place by which randomly chosen low values are added to genes with a low probability or mutation rate. Once the new generation's members are found, the GA returns to stage two.

As there can be only one solution to choice of segmentation parameters, the best chromosome is chosen at the end of a run according to its cost function value. It is possible to select a stopping point by checking when the difference in the cost function's value between successive generations passed below a minimum value. However, because of the relatively lengthy time taken to evaluate each application of a segmentation algorithm to an image, stopping after a given number of generations is a practical alternative. The rate of convergence to a global minimum value of the cost function for the mean-shift algorithm in the tests in Section V was found to be determined by the size of the population at each generation. However, the convergence over many generations is also examined.

IV. RELATED WORK

The image-processing and computer vision communities have developed both *de jure* (e.g., IPI-PIKS) and *de facto* (e.g., IUE, Target Jr, VXL, RAVL, Tina, ITK, and the Intel Open Computer Vision Library) facilities, containing collections of algorithms. In particular, ITK hosts the code

for numerous image segmentation algorithms intended for medical applications. (For surveys of segmentation algorithms refer for example to [9][10].) The value of these would be enhanced if there existed a direct route to selecting one or more algorithms for a particular application, as it is difficult to assess the suitability of a segmentation algorithm without detailed comparison through testing. Contrasting image segmentation to recognition tasks such as the use of handwriting, and face databases, the authors of [11] remark "Typically [in segmentation] researchers will show their results on a few images and point out why the results 'look good'". Part of the problem is the difficulty of establishing metrics, whether pixel-based figures of merit [12] or rankings against a set of criteria [13] or through precision-recall curves [5]. However, part of the problem may also be the logistics of performing a large number of tests. Additionally, a user needs guidance in the form of a testing structure or evaluation framework.

A. Quantitative evaluation

There has been an emphasis in the past on algorithmic novelty, whereas increasingly the importance of systematic validation on sufficient datasets is stressed. For a given input, the corresponding correct output is also known, either because the input is simulated or by virtue of either expert opinion or the presence of a ground truth such as hand-segmented images. The developer aspires to make the algorithm agree with the correct output for every corresponding input. Implicit in this procedure is that the finite set of test data is representative of variation in the real world and that the number of samples is large enough [14].

At the University of Berkeley, CA Martin [11] supervised the creation of a database of 12,000 hand-labelled (from a pool of 30 human subjects) segmentations of images taken from the Corel dataset of 1,000 images. The smaller Sowerby database [15] is an earlier collection of hand-segmented images which has been used to evaluate edge-detection algorithms [16]. The work in [17] also used ground truth from a set of fifty images and applied analysis of variance to five evaluation metrics for object recognition algorithms. The Berkeley database [11] encourages users to download benchmarking code as well as 200 training images and a further 100 test images of size 240×160 pixels. The originators of the database ran a multiple cue segmentation algorithm of their own using this database. A number of pattern classifiers were applied to combining the cues prior to benchmarking with times reported on a 1 GHz Pentium IV ranging from several minutes to several hours (for a Support Vector Machine) per image.

B. Evaluation structures

The PCCV project on benchmarking vision systems extended a software harness for evaluation and testing (HATE) [18], for running quantitative and statistical tests on the results of different image-processing algorithms. HATE can also automate the application of an algorithm across a large enough set of images to be statistically significant. However, if these tests are performed on a PC alone, even a high-end PC, then the time consumed is likely to be large. A set of 1,441

cloud images, size 768×576 pixels, on a dual-processor 1.8 GHz Athlon MP 2200+ (256 KB L2 cache) took 22 hours to process. Processing consisted of applying an algorithm to detect the amount of cloud cover and the approximate height of the clouds in images taking from a camera looking in the direction of a satellite uplink.

Algoval [19] provides a testing framework. Researchers upload their software to a website, where it is run against a series of tests and the results are reported on the website. This approach has the advantage that memory usage and speed can be compared in addition to algorithmic performance; however, this must be balanced by: the need to write the software in Java (and to interface to the Algoval testing classes); the centralized nature of the testing; the need to upload the software for testing — and the possible embarrassment resulting from poor performance. The need to upload code, even in compiled form, makes this type of system unattractive to industrial users, who need to preserve commercial confidentiality.

C. Evaluation methods

Evaluation methods for segmentation can be divided into those that are independent of ground truth and those that are not. An early survey of independent segmentation evaluation methods was completed in [20]. Apart from a categorization into five types of evaluation, [20] also investigated the methods for their ability to discriminate between different segmentation thresholds. An interesting feature of this comparison was that, while no reference was made to ground truth, the methods did agree on which of the alternative segmentations was preferable. An entropy-based method [21] was introduced at a later date to Zhang's survey. Its authors advocate its use in preference to those from the quantitative school, because the latter tend to be empirically based.

The global and local consistency measures introduced in [11] produce high scores for segmentations by humans and, hence, will rate a computer-generated segmentation highly if it corresponds to a human segmentation. Because these measures are tolerant to refinements, over and under segmentation is accepted, in the sense that the scores do not vary greatly when the same algorithm has been applied with the different parameters. Over segmentation may not be a problem if subsequent processing is able to aggregate regions but if segmentation is regarded as the end product it is an issue. Another measure based on Hamming distance [22] may be used as a corrective. Both [11] and [22] are intended for the quantitative approach considered in this paper. However, the main purpose herein is not to compare evaluations, though in the next Section, the evaluation methods appear as cost functions which are inserted into an evaluation harness.

V. RESULTS

The mean-shift algorithm makes a convenient example, especially as the authors have made EDISON code available at <http://www.caip.rutgers.edu/riul/research/robust.html>, for which we are grateful. Previous mode information avoided re-applying the mean-shift procedure to some data points.

The approximate cost of an exhaustive search across parameter space for a single image on a single cluster node is as follows. The processing time on a single cluster node takes between a minimum of 907 ms (less than 1 s) and a maximum of 26799 ms (about 27 s), depending on the parameters selected for a particular image. The average time taken was 5937 ms. Assuming 6 s per test with each of three parameters varied between 1 and 20, *i.e.* 8000 tests, then a complete run would take 13.3 hours. Evaluation of each result, the cost function after normalization, consists of pixel-by-pixel comparison with a ground truth image from the Berkeley database. However, this calculation took less than 1% of the processing time. The cluster was employed as a throughput engine, in the sense that no exploitation of internal parallelism took place. In the unlikely event that all nodes were available and disregarding input/output overhead the per image run time for a single image is still 22 min. For the same image, a run with a GA took 130635 ms, *i.e.* 2.2 min, which on a cluster takes about 35 s. This is, of course a considerable reduction on an exhaustive search.

Table I is an illustrative rather than representative run showing the generation-by-generation most fit selection of 'chromosomes'. Three parameters formed the chromosomes: `radiusR` (the range radius of the mean-shift sphere in color space), `radiusS` (the spatial radius in grid space), and `colorDistance` (defining the region merge threshold). There was a population of just five for each generation, which explains why the cost function's value does not reach a minimal value (as shown next). Two GA parameters, the recombination rate, *i.e.* α in the line recombination, and the mutation rate were set to 0.6 and 0.8 respectively. The latter governs the ability to escape from a local minimum. As with such evolutionary algorithms, it is not otherwise possible to directly govern their behavior, which is probably their principal disadvantage. However, the reduction in duration at reaching a selection adequately compensates for that. For the image in this test, the average time for each call to the segmentation algorithm was 6 s, with the maximum time at 21 s and the minimum time at 0.9 s. The total time taken in running the calls was 298 s, while the total time for all processing was 299 s, *i.e.* about 5 min.

A disadvantage of truncating the search after a fixed number of generations is that there is no guarantee that the GA could be (say) performing hill climbing to leave a suspected local minimum. In Table I, the seventh generation value is less than the eighth and, therefore, had the search been concluded at the eighth unhelpful parameter values would be selected. A possible solution to this problem is to include a refinement or polishing stage in which the path taken by the GA over each generation is inspected to more closely approach a global minimum. This issue is returned to at the end of this Section in relation to graph-based segmentation. Experimenting with the GA as in Table II with a population of ten chromosomes in each generation gives rise to interesting observations. The first point to notice is that the cost function's value remains the same throughout the tests. The rapid descent to a global

Gen.	Cost	Chromosomes		
		radiusS	radiusR	colorDistance
1	18.63	3	8	16
2	14.17	3	14	16
3	15.27	3	16	14
4	14.61	1	16	17
5	16.74	11	4	17
6	13.76	11	18	19
7	11.87	3	17	19
8	14.26	12	17	19
9	11.19	2	15	19

TABLE I

EXAMPLE OUTPUT OF THE EVALUATION OF MEAN-SHIFT PARAMETERS WITH THE GA MODULE.

Gen.	Cost	Chromosomes		
		radiusS	radiusR	colorDistance
1	4.71	2	5	16
2	4.71	2	16	11
3	4.71	2	16	11
4	4.71	2	4	17
5	4.71	2	2	17
6	4.71	16	2	19
7	4.71	2	10	16
8	4.71	2	18	15
9	4.71	2	16	12
10	4.71	2	17	12
11	4.71	2	10	19
12	4.71	2	10	19
13	4.71	2	10	17
14	4.71	2	15	15
15	4.71	3	11	16
16	4.71	3	18	19
18	4.71	4	15	19
19	4.71	8	2	19
20	4.71	1	3	19

TABLE II

TRIAL OUTPUT OF THE EVALUATION OF MEAN-SHIFT PARAMETERS WITH THE GA MODULE.

minimum arises because a population of ten was taken for each generation. The other important point is that although radiusR and radiusS do not converge to produce an optimal set of parameters, colorDistance is never lower than 11. We found that these two points are always true for the mean-shift algorithm over a range of different images, and although the lowest score reached will be different for each image, it can be concluded that radiusR and radiusS do not make a significant difference to the result as long as the colorDistance parameter is larger than 8.

The more common alternative to truncating the search after a fixed number of generations with a large initial population is to complete the search after a convergence criterion had been met. How the search is conducted is a pragmatic decision, as in essence a GA is simply a more disciplined way than random probing to explore a large problem space. To examine the behavior over successive generations a very low population of just two members was deliberately chosen to give slow convergence. The recombination rate was set to 0.6 and the mutation rate was set to 0.2 for the mean-shift segmentation parameters. Fig. 1. A best-fit linear regression line is found by a standard numerical method, without any claims for goodness-of-fit. The line shows that the cost function values has a slight negative trend showing continuous improvement, though with

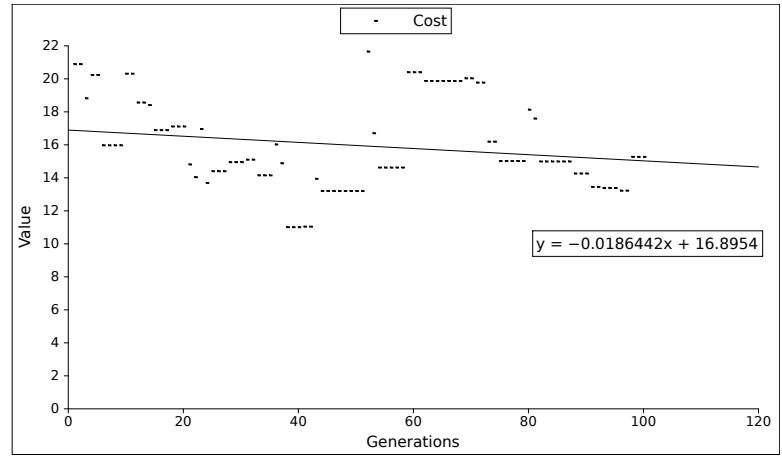


Fig. 1. Mean-shift parameter convergence with linear trend for 100 generations of the cost function value.

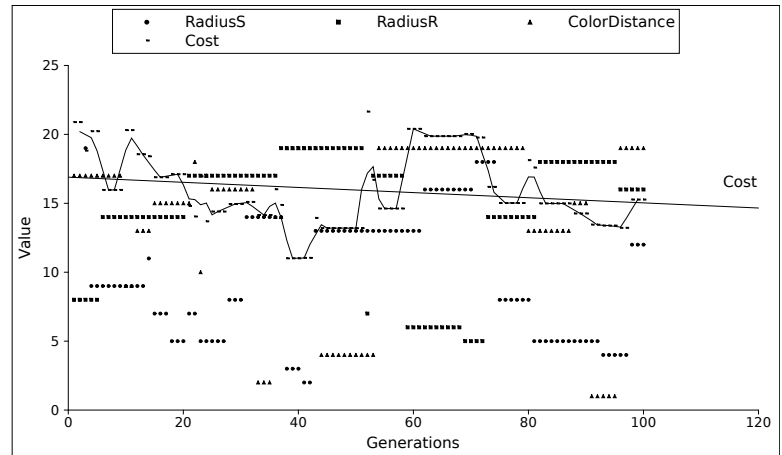


Fig. 2. Mean-shift parameter convergence with linear trend of the cost function value, including the behavior of contributory parameters.

significant divergences and even lower values of the cost function at around 40 generations. In the unlikely event that a population of just two was used then linear convergence to the evident minimum at around 40 generations would be slow. In Fig. 2, the values of the contributory parameters are superimposed. The values of the cost function are connected by moving average line, with the average taken over three data points. In broad terms, the behavior of the values chosen by the GA for the other parameter is oscillatory, displaying what approaches a systematic sampling of parameter space.

Fig 3 shows results of varying the mean-shift parameters. Higher values of radiusR results in less regions, while higher values of radiusS effectively results in more computation but smoother region boundaries. Fig. 4 shows the results of an exhaustive search across a set of images from the Berkeley database to further explore the parameter space of the mean-shift algorithm. The harness was configured to output segmented regions according to their size in pixels. For fixed value of radiusS=1 and radiusR=1, the colorDistance setting governs the size of segmented region. In this example, the single parameter governs a wide variety of possible segmentations.

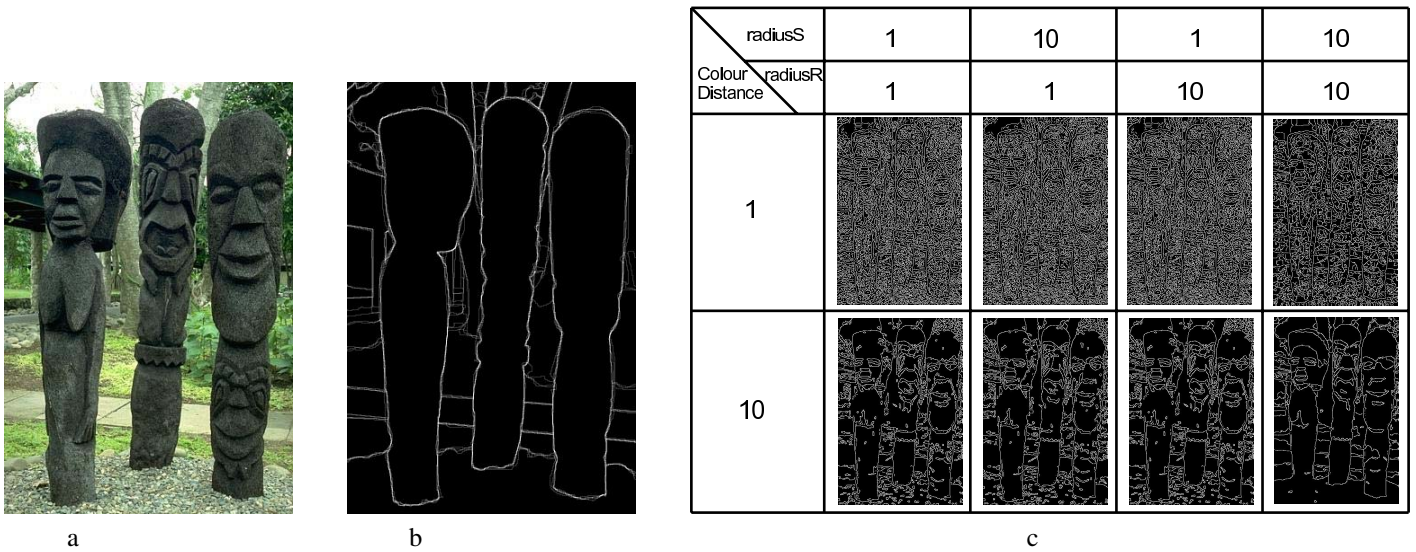


Fig. 3. (a) Example image (Easter island statues) from Berkeley segmentation database (b) human hand-segmentation (c) variation of segmentations with parameter settings.

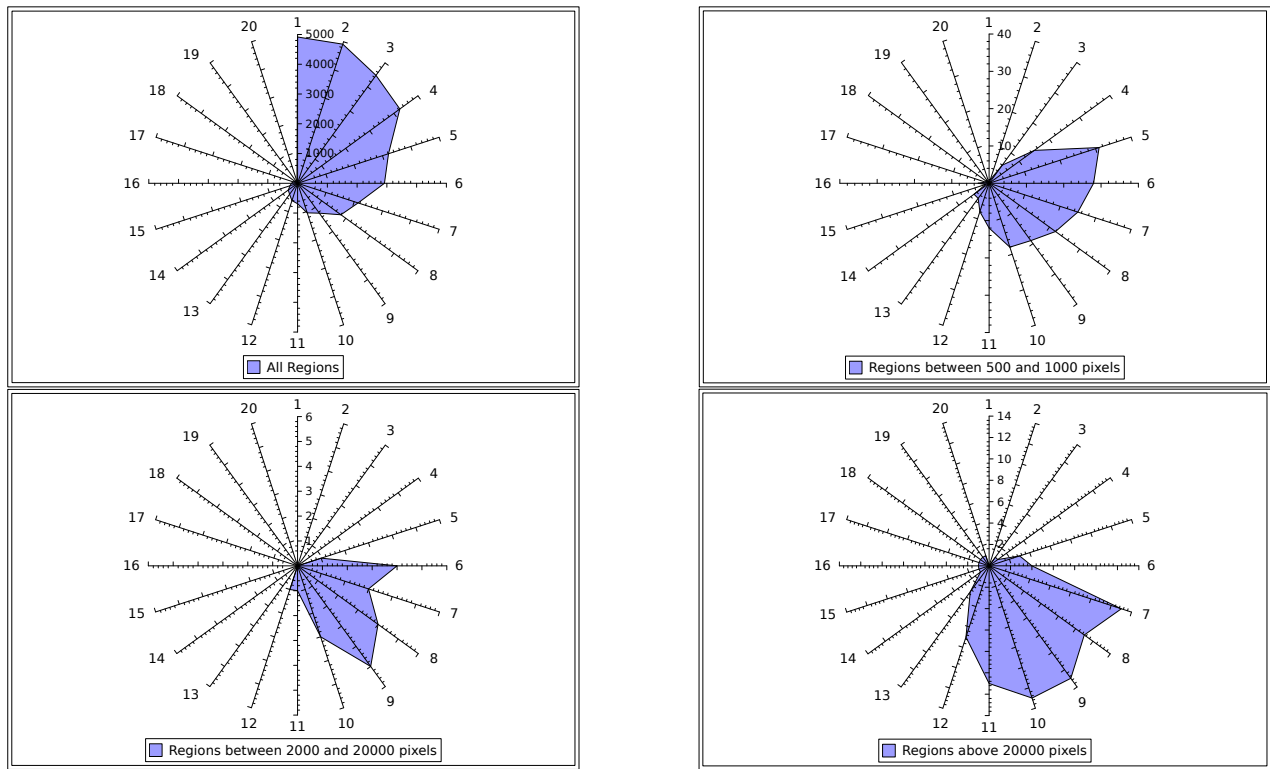


Fig. 4. The results of an extensive search of parameter space for the mean-shift algorithm grouped by segmented region size. The colorDistance parameter is varied between 1 to 20 (in the circular direction), while radiusR and radiusS are fixed with value 1. The spokes of the circle represent the frequency of regions at a particular colorDistance parameter setting.

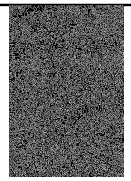


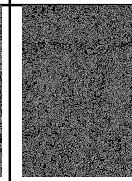

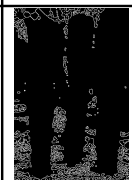


No. of C	Th	1	100	1	1
	D	1	1	10	1
	I	1	1	1	100
256					
6					

Fig. 5. Variation of segmentations of Fig. 3 with parameter settings for Watershed segmentation [23].

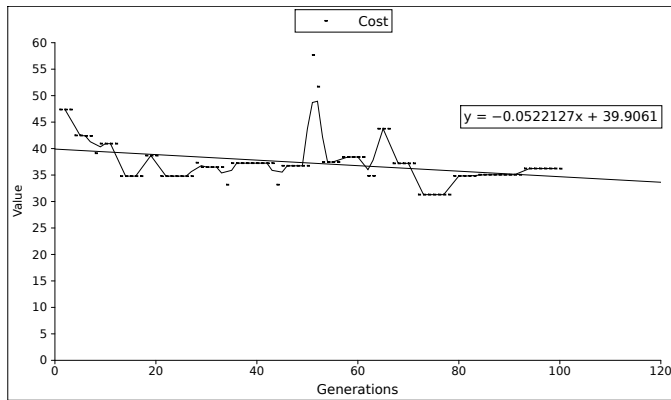



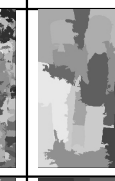






Fig. 6. Watershed segmentation parameter convergence with linear trend of the cost function value.

Experiments with the classic Watershed algorithm [23] also gave rise to a variety of results, depending on choice of parameters. A colour quantization or smoothing algorithm was added as a pre-processing step before application of Watershed. In Fig. 6: TR is the threshold; D is the δ (delta) threshold; I is the number of iterations; and C is the number of colours after colour quantization. It is apparent from Fig. 6 that already before 20 iterations the GA makes substantial progress in finding a minimum of the cost function, *i.e.* the match to the ground truth image. However, thereafter there are oscillations in value. Though there is a negative going trend to the results, the final value after 100 generations is not the minimum.

Graph-based segmentation [24] also displayed a similar dependence on choice of parameters. We gratefully acknowledge source code from <http://people.cs.uchicago.edu/~pff/segment/>. There are three input parameters available for this algorithm: 1) σ (sigma), the variance of a Gaussian smoothing filter prior to segmentation; 2) k , which sets the scale of the components

Sigma	Min	20	2000	20	2000
	K	50	50	500	500
	0.1				
1					

c

Fig. 7. Variation of segmentations of Fig. 3 with parameter settings for graph-based segmentation [24].

found through a thresholding function; and 3) *Min*, which is the minimum component size enforced by post-processing. Fig. 7 then shows the considerable variety of segmentations arising from these parameter settings for the same original image as Fig. 3. Similarly to Fig. 2 and with the same GA search parameters, Fig. 8 shows 100 generations of a parameter search, again plotting a moving average through the data points. The slight negative of the linear trend is more apparent in this representation, as is the oscillatory nature of the testing of individual parameters.

As mentioned earlier, it is possible to refine the output of the GA by application of a non-GA polishing algorithm to what is a non-constrained, non-linear optimization problem. Newtonian methods are unsuitable if the cost function to be optimized is non-differentiable. Therefore, this work used the Nelder-Mead direct search, "simplex method" [25]. In the implementation, previous values found by the GA form the input to the algorithm along with the cost function. The values form the vertices of the simplex and at each iteration the worst one of these values is replaced. This is achieved by a number of trial evaluations of the cost function at the vertex after the simplex has been reflected, expanded or contracted. Fig. 9 shows how the Nelder-Mead procedure will find a lower value than that given at the end of the GA iterations. However, in this value the parameter settings found are no lower than those of the minimum value found in the course of the GA iterations. Therefore, applying the Nelder-Mead procedure should certainly improve upon the final GA result but the effect is to deepen that result rather than find a global minimum within parameter space.

VI. CONCLUSIONS

To arrive at a best-fit parameter configuration, in the face of a diversity of parameter-dependent results is a time-consuming task, yet seems necessary if quantitative evaluation is to be

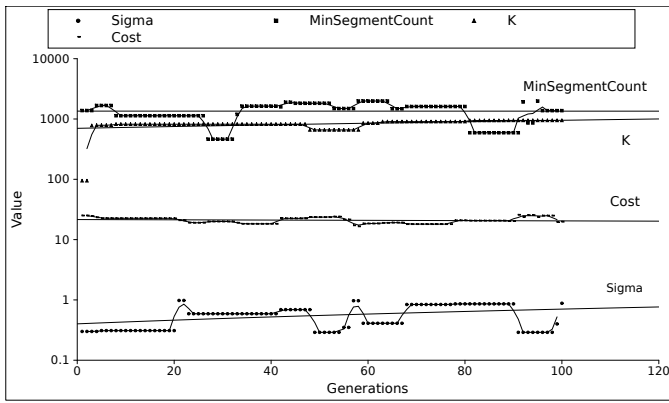


Fig. 8. Graph-based segmentation parameter convergence with linear trend of the cost function value, including the behavior of contributory parameters. (Note the logarithmic vertical axis.)

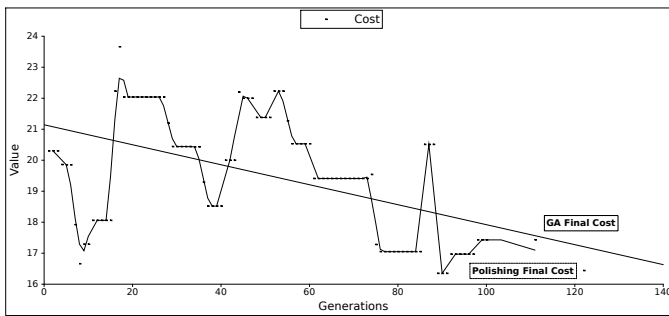


Fig. 9. Application of Nelder-Mead polishing to GA optimization, showing the final GA derived value and the value found after further iterations of the polishing algorithm.

come standard. An automated harness written in a convenient scripting language is an asset whatever the testing task. For large numbers of images then a cluster computer is certainly necessary. However, the main contribution of this paper is that, whether working in batch mode or not, application of a genetic algorithm dramatically reduces search times. In fact, using the genetic algorithm has highlighted the importance of parameter setting and the criticality of a particular parameter over another. However, a genetic algorithm is by no means guaranteed to find an optimal solution and this study experimented with a polishing algorithm to improve the solution. A true cost of applying an algorithm in batch mode consists not only of the quality of the segmentations but the time taken to evaluate the segmentations. The two factors should be traded off against each other. Future investigations consist of profiling algorithms to find their time costs, particularly in regard to pre- and post-processing times.

ACKNOWLEDGMENT

The authors would like to thank Etisalat College of Engineering and the Emirates Telecommunication Corporation (Etisalat), United Arab Emirates, for providing the financial support for this work.

REFERENCES

- [1] D. Comaniciu and P. Meer, "Mean shift: A robust approach towards feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.
- [2] J. K. Ousterhout, "Scripting: Higher-level programming for the 21st century," *IEEE Computer*, vol. 31, no. 3, pp. 23–30, 1998.
- [3] B. B. Welch and J. Hobbs, *Practical Programming in Tcl and Tk*. Prentice Hall, Upper Saddle River, NJ, 2003.
- [4] P. R. Cohen, *Empirical methods for artificial intelligence*. Cambridge, MA, USA: MIT Press, 1995.
- [5] D. R. Martin, C. Fowles, and J. Malik, "Learning to detect natural image boundaries using local brightness, color, and texture cues," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 5, pp. 530–549, 2004.
- [6] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.
- [7] H. Polheim, *GEATbx: Evolutionary Algorithm Toolbox for MATLAB, version 3.7*, 2005, online at <http://www.geatbx.com/docu/index.html>.
- [8] H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm: I. continuous parameter optimization," *Evolutionary Computation*, vol. 1, pp. 25–49, 1993.
- [9] R. Haralick and L. Shapiro, "Survey — image segmentation techniques," *Computer Vision Graphics and Image Processing*, vol. 29, pp. 100–132, 1985.
- [10] H. D. Cheng, X. H. Xiang, Y. Sun, and J. Wang, "Color image segmentation: Advances and prospects," *Pattern Recognition*, vol. 34, no. 12, pp. 2259–2281, 2001.
- [11] D. Martin, C. Fowles, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *International Conference on Computer Vision*, 2001.
- [12] W. A. Yasnoff, J. K. Mui, and J. M. Bacus, "Error measure for scene segmentation," *Pattern Recognition*, vol. 9, pp. 217–231, 1977.
- [13] M. Everingham, H. Muller, and B. Thomas, "Evaluating image segmentation algorithms using monotonic hulls in fitness/cost space," in *12th British Machine Vision Conference (BVMC 2001)*, 2001, pp. 363–372.
- [14] I. Guyon, J. Markhoul, R. Schwartz, and V. Vapnik, "What size test set gives good error rate estimates?" *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 1, pp. 52–64, 1998.
- [15] G. Rees and P. Greenway, "Metrics for image segmentation," in *ICVS Performance Characterization Workshop*, 1999.
- [16] S. Konishi, A. L. Yuille, J. Coughlan, and S. C. Zhu, "Fundamental bounds on edge detection: An information theoretic evaluation of different edge cues," in *Computer Vision and Pattern Recognition*, 1995, pp. 573–579.
- [17] S. Borra and S. Sarkar, "A framework for performance characterization of intermediate-level grouping modules," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 11, pp. 1306–1312, 1997.
- [18] P. Courtney, N. A. Thacker, and A. Clark, "Algorithmic modelling for performance evaluation," *Machine Vision Applications*, vol. 9, no. 5-6, pp. 219–228, 1997.
- [19] S. Lucas and K. Sarampalis, "Automatic evaluation of algorithms over the Internet," in *15th Int. Conf. on Pattern Recognition*, 2000, pp. 471–474.
- [20] Y. J. Zhang, "A survey of evaluation methods for image segmentation," *Pattern Recognition*, vol. 29, no. 8, pp. 1335–1346, 1996.
- [21] H. Zhang, J. E. Fritts, and S. A. Goldman, "An entropy-based objective evaluation method for image segmentation," in *SPIE Electronic Imaging - Storage and Retrieval Methods and Applications for Multimedia*, 2004, pp. 38–49.
- [22] Q. Huang and B. Dom, "Quantitative methods of evaluating image segmentation," in *Int. Conf. on Image Processing*, 1995, pp. 53–56.
- [23] L. Vincent and P. Soille, "Watersheds in digital spaces: An efficient algorithm based in immersion simulations," *IEEE Trans. on Pattern Recognition and Machine Learning*, vol. 13, no. 6, pp. 583–598, 1991.
- [24] P. F. Felzenszwalb and D. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 29, no. 2, pp. 167–181, 2004.
- [25] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, pp. 308–313, 1965.