Parallel evolution of image processing tools for multispectral imagery

Neal R. Harvey, Steven P. Brumby, Simon J. Perkins, Reid B. Porter, James Theiler, A. Cody Young, John J. Szymanski, and Jeffrey J. Bloch

Space and Remote Sensing Sciences Group
Los Alamos National Laboratory
Los Alamos, New Mexico 87545

ABSTRACT

We describe the implementation and performance of a parallel, hybrid evolutionary-algorithm-based system, which optimizes image processing tools for feature-finding tasks in multi-spectral imagery (MSI) data sets. Our system uses an integrated spatio-spectral approach and is capable of combining suitably-registered data from different sensors. We investigate the speed-up obtained by parallelization of the evolutionary process via multiple processors (a workstation cluster) and develop a model for prediction of run-times for different numbers of processors. We demonstrate our system on Landsat Thematic Mapper MSI , covering the recent Cerro Grande fire at Los Alamos, NM, USA.

Keywords: multi-spectral, image processing, evolutionary computation, parallelization

1. INTRODUCTION

Vast quantities of remotely-sensed data are becoming available, from an increasing number of sensors. Feature extraction is a poweful and important technique for the exploitation of the information within such data. In order to perform feature extraction tasks, one requires suitable analysis tools. The creation of such tools is important, yet can prove extremely expensive. In order to address this problem, we have been developing an automated system for the generation of feature detection tools, which utilizes an evolutionary approach.

This evolutionary approach is attractive for its flexibility: if a fitness measure can be derived for a problem, then it may be possible to find a solution to the problem. Taking an evolutionary approach has helped in the solution of many problems, such as designing protein sequences with desired structures,[1] optimization of dynamic routing in telecommunications networks[2] and optimizing image processing filter parameters for archive film restoration.[3]

In adopting an evolutionary approach to solving a problem, the representation of candidate solutions in such a manner that they may be effectively manipulated is a critical issue. We employ a genetic programming (GP) method for candidate solution representation, because each candidate solution represents an image processing algorithm. Previous applications of GP to image processing have included feature extraction: e.g. Daida et al.,[4] Brumby et al.,[5] Theiler et al.,[6] and Brumby et al.,[7] where it has been demonstrated that real problems in remote-sensing can be successfully addressed using GP techniques.

The nature of an evolutionary algorithm is that a number of candidate solutions is maintained (a population of individuals) and each of these individuals is tested on some representative problem. Maintaining a large population leads to robust solutions in many problem domains but also leads to large computation times. However, one attraction of evolutionary approaches is that they lend themselves extremely well to parallelisation: where the assessment of individuals can be done on several computers simultaneously. In this manner, substantial speed-up can be achieved. A primary motivation of this paper is to investigate the effectiveness of parallelisation in our system.

Increased speed in algorithm development is often as much of a necessity as a desire, as made evident by the recent wild fires in Los Alamos, NM, USA. With monsoon rainfall imminent, and with fire-ravaged terrain providing a flood threat for town residents and the nearby national laboratory, there is an immediate need for information regarding location of heavy fire damage, in order to determine areas most at risk. It seemed an ideal (and very real) problem on which to demonstrate the flexibility and speed of our feature extraction tool.

Work supported by the U.S. Department of Energy.
Emails: {harve,brumby,s.perkins,rporter,jt,szymanski,jbloch}@lanl.gov.

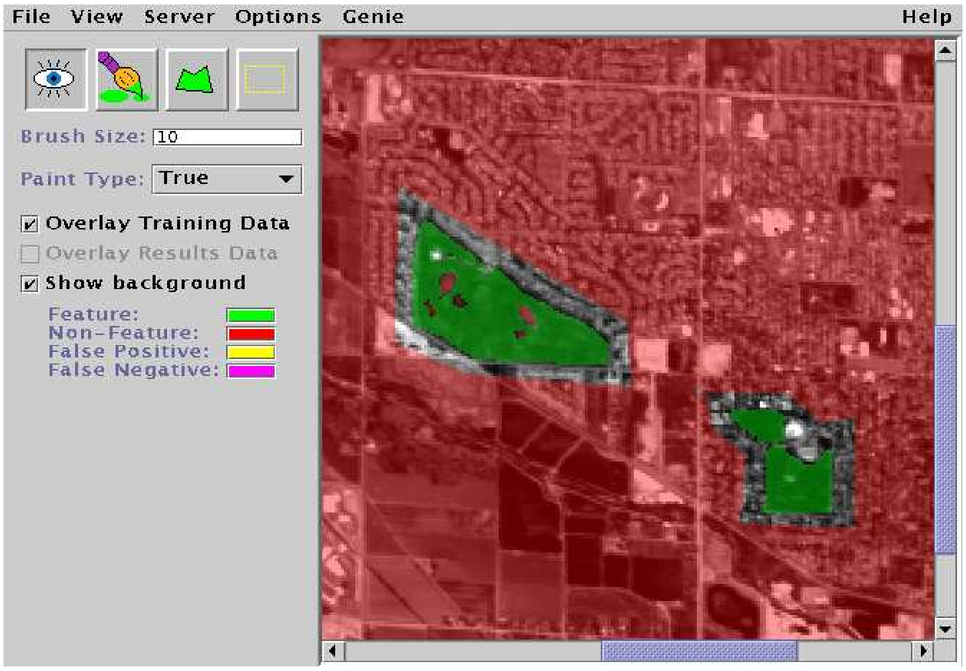


Figure 1. GUI for training data mark-up. Note that ALADDIN relies heavily on color, which does not show up well in this image. The light colored patches in the center-right and upper-right parts of the image are two golf courses that have been marked up as "true". Most of the rest of the image has been marked up as "false", except for a small region around the golf courses which has been left as "unknown".

2. OUR SYSTEM: "GENIE"

Our feature detection system, "GENIE" (GENetic Image Exploitation),[5,6,8] employs a classic evolutionary paradigm: a population of individuals is maintained and each individual is assessed and assigned a fitness value. The fitness of an individual is based on an objective measure of its performance in its environment. After fitness determination, the evolutionary operators of selection, crossover and mutation are applied to the population and the entire process of fitness evaluation, selection, crossover and mutation is iterated until some stopping condition is satisfied.

2.1. Training Data

The environment for each individual in the population consists of *data* planes, each plane corresponding to data in a specific spectral channel, together with a *weight* plane and a *truth* plane. The weight plane identifies those pixels which are to be used in the training process, and the truth plane identifies the particular features of interest within the training data. The data in the weight and truth planes may be derived from actual ground truth (collected on the ground, at the time the image was taken) or from the reasoned judgement of an analyst looking at the data. Obtaining true ground truth can be very expensive. Therefore, we provide/use a Java-based tool called Aladdin to assist the analyst in making judgements about and marking up the data. Using Aladdin, the analyst/user can view a multi-spectral image in a number of different ways, and can mark up training data by painting directly on the image using a mouse. Currently, training data is ternary-valued with the possible values being "true", "false", and "unknown". *True* defines regions where the analyst is confident that the feature of interest **does** exist. *False* defines regions where the analyst is confident that the feature of interest **does not** exist. Fig. 1 shows a screen capture of an example session. In this partciular example the analyst has marked out golf courses as of interest.

2.2. Encoding Individuals

Each individual *chromosome* in the population consists of a fixed-length string of *genes*. Each gene in GENIE corresponds to a primitive image processing operation, and hence the entire chromosome describes a more complex algorithm consisting of a sequence of primitive image processing operations.

2.2.1. Genes and Chromosomes

A single gene consists of an operator name, plus a variable number of input arguments, specifying from where input is to come; output arguments, specifying to where output is to be written; and operator parameters, modifying the operator's function. Different numbers of parameters are required for different operators. The operators used in GENIE take one or more distinct data planes as input, and generally (but not always) produce a single data plane as output. Input can be taken from any data plane in the training data image cube. Output is written to one of a small number of *scratch planes*, which are simply temporary workspaces where a data plane can be stored. Genes can also take their input from these scratch planes, but only if that scratch plane has previously been written to by another gene positioned earlier in the chromosome sequence.

The overall algorithm that a given chromosome represents can be thought of as a directed acyclic graph, where the non-terminal nodes are primitive image processing operations, and the terminal nodes are individual data planes extracted from the multi-spectral data used as input. The scratch planes are the 'adhesive' that binds together primitive operations into image processing pipelines. Traditional GP[9] uses a variable sized tree representation for algorithms. Our representation differs in that it allows the reuse of values computed by sub-trees, since many nodes can access the same scratch plane, i.e. the resulting algorithm is a graph rather than a tree. Another way it differs is that the total number of nodes is fixed (although not all of these may be actually used in the final graph), and crossover is carried out directly on the linear representation.

We have restricted our "gene pool" to a set of what we consider useful primitive image processing operators. These include spectral, spatial, logical and thresholding operators.

The notation we use for genes is most easily illustrated using an example: the gene `[DIVP rD0 rS1 wS2]` applies pixel-by-pixel division to two input planes, read from data plane 0 and from scratch plane 1, and writes its output to scratch plane 2. If a gene requires additional operator parameters, they are listed after the input and output arguments.

It should be noted that, even though all chromosomes have the same fixed number of genes, the *effective length* of the resulting algorithm graph may be less than this. For example, an operator may write to a scratch plane that is subsequently overwritten by another gene before anything reads from it. GENIE performs an analysis of chromosome graphs when they are created and only carries out those processing steps that actually affect the final result. Hence, the chromosome's fixed length acts as a maximum effective length.

2.3. Backends

Complete classification requires that our algorithm's final output be a single binary-valued output plane. It would be possible, after running the chromosome algorithm, to treat, for example, the contents of the first scratch plane as the final output from the algorithm (obviously, thresholding would be necessary to obtain a binary result). However, we have found it to be advantageous to perform final classification using a non-evolutionary algorithm.

In order to do this, a subset of the scratch planes and data planes is selected to be *answer planes*. In our experiments this subset typically consists of only the scratch planes. We then use the provided training data and the contents of the N answer planes to derive the *Fisher Discriminant*: the linear combination of the answer planes that maximizes the mean separation, in N dimensions, between those pixels marked up as "true" and those pixels marked up as "false", normalized by the "total variance" in the projection defined by the linear combination. See Ref. [9] for details of this discriminant.

The discriminant-finding phase outputs a gray-scale image, which is then reduced to a binary image by using Brent's method[11] to find the threshold value which minimizes the total number of misclassifications (false positives plus false negatives) on the training data.

2.4. Fitness Evaluation

The fitness of a candidate solution is given by the degree of agreement between the final binary output plane and the training data. This degree of agreement is determined by the Hamming distance between the final binary output and the training data, with only those pixels marked as true or false contributing towards the metric. The Hamming distance is then normalized so that a perfect score is 1000. Put in a more formal context: Let H be the Hamming distance between the final binary output of the algorithm and the training data, with only pixels marked as true

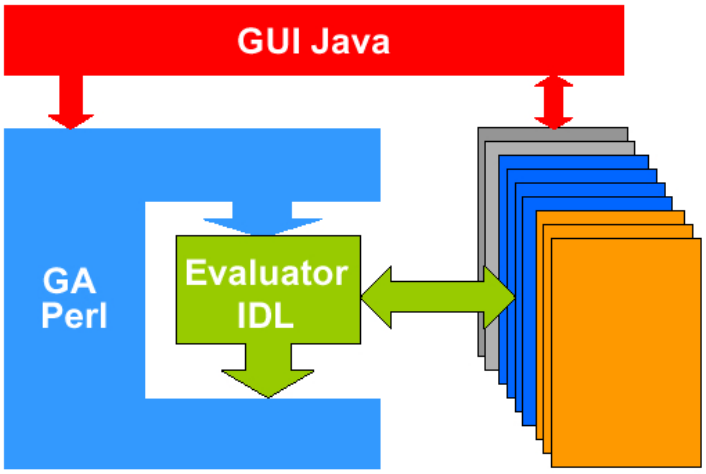


Figure 2. Software architecture of the serial system described. Note that the feature depicted on the right of this diagram represents the input data, training data and scratch planes

or false contributing towards the metric, let N be the number of classified pixels in the training image (i.e. pixels marked as either "true" or "false") and let F be the fitness of the candidate solution.

$$F = (1 - (H/N)) \times 1000 \tag{1}$$

2.5. Basic Software Implementation

The genetic algorithm code has been implemented in object-oriented Perl. This provides a convenient environment for the string manipulations required by the evolutionary operations and simple access to the underlying operating system (Linux/Solaris). Evaluation of chromosomes' fitness is the computationally intensive part of the evolutionary process and, for that reason, we currently use RSI's IDL language and image processing environment. Within IDL, individual genes correspond to single primitive image operators, which are coded as IDL procedures, with a complete chromosome representation being coded as an IDL batch executable. In the serial implementation, a single IDL session is opened at the start of a run and communicates with the Perl code via a two-way unix pipe. This pipe is a low-bandwidth connection. It is only the IDL session itself that needs access to the input and training data, which requires a high-bandwidth connection. The ALADDIN tool was written in Java. Fig. 2 shows the software architecture of the system.

2.6. Parallel Software Implementation

Fitness evaluation makes up the bulk of the computation, particularly for large training sets and therefore we concentrated our efforts on parallelization in that area. To this end, the software was set up so that, instead of a single IDL session being opened up at the start of a run, multiple IDL sessions can be opened up on a number of different, networked machines. Only the fitness evaluation part of the evolutionary process was parallelized. All other evolutionary processes are performed on the host computer. The initialisation phase is conducted serially, with each IDL session in turn loading the training data. Once the initialization phase is completed the Perl genetic algorithm (GA) then farms out a chromosome to be evaluated to an idle IDL session. If no sessions are known to be idle, then the Perl begins checking the sessions' output pipes, one at a time, to see if any have finished. If one is finished, then its pipe is read, the results are noted, and the session is marked idle. Fig. 3 shows the software architecture of the parallelized system.

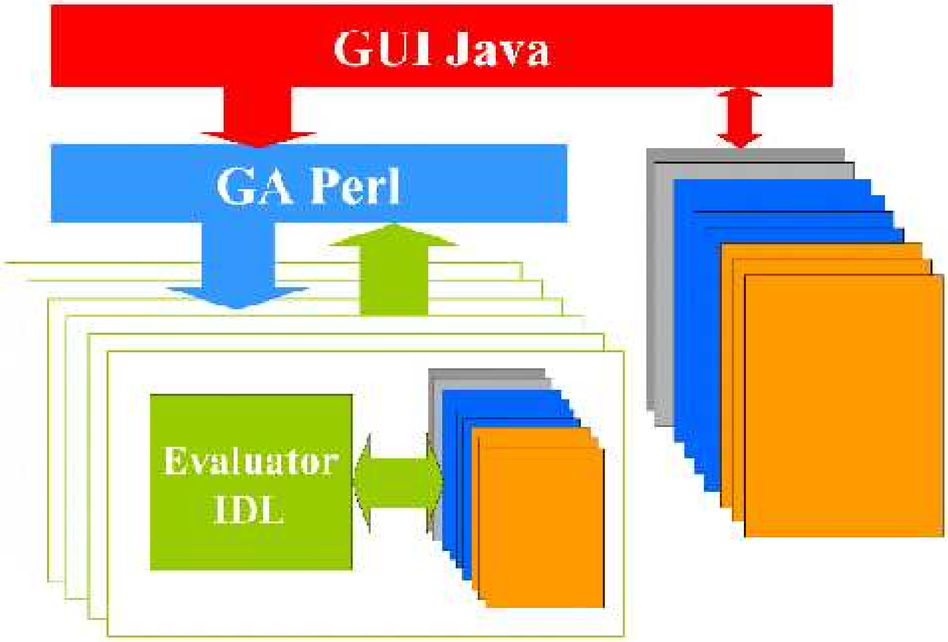


Figure 3. Software architecture of the parallelized system described, showing the multiple IDL sessions

3. PARALLELIZATION EXPERIMENTS

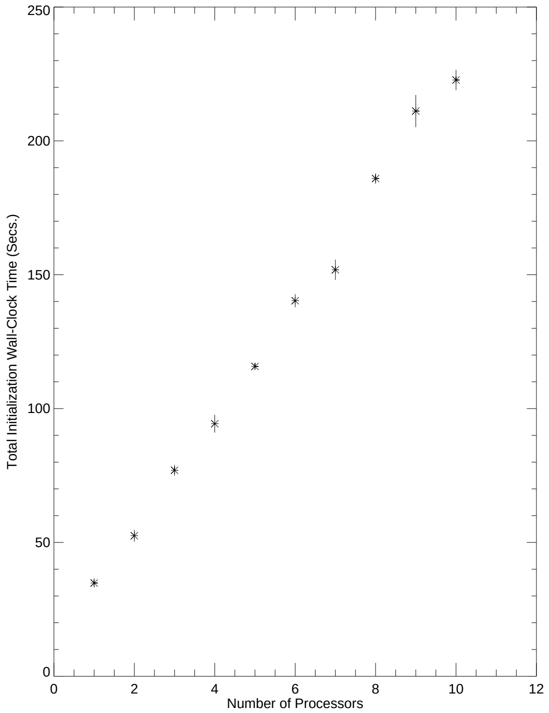
In order to determine the benefits of parallelization, experiments were run to compare the performance of GENIE, using a variable number of processors, on a test problem. GENIE was run on the same test problem, with the same parameters, for 20 runs. This process was repeated, varying the number of processors used for fitness evaluation from 1 to 10. Each processor in the experiment was a 466 MHz Intel Celeron processor having 256 MB of RAM, and running the Linux operating system. Each individual GENIE run involved evaluating 1000 chromosomes. Several timing measurements were taken: these included initialisation wall-clock time, total Perl CPU time, total IDL evaluation CPU time and total wall-clock time for the run. The average times for the 20 runs, with each processor configuration was determined and compared.

The test problem involved trying to find a golf course[8] in 10-channel simulated MTI[12] data, produced from 224-channel AVIRIS data. This problem seemed a good test case as golf courses, due to their nature and characteristics in remotely-sensed data, possess distinctive spatial *and* spectral characteristics. Therefore, a good golf course-finding algorithm has to be able to utilize both the spectral and spatial information in the image.

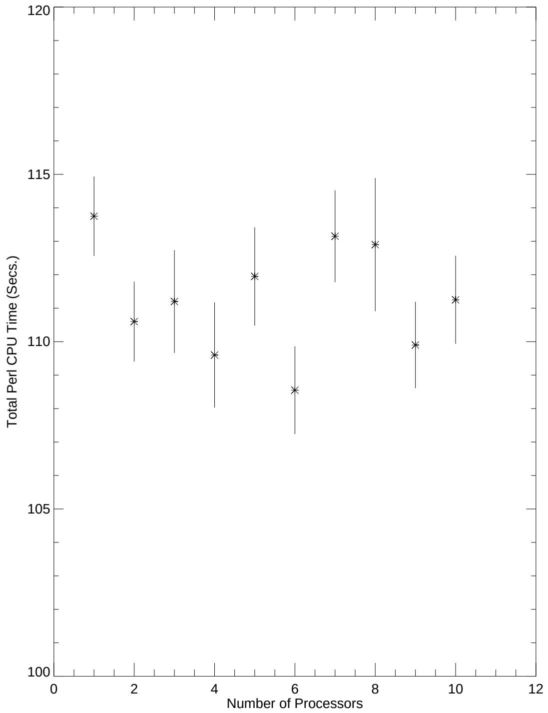
Fig. 4(a) shows the variation of initialization time with number of processors. It can be seen that total initialization time varies linearly with the number of processors. This is expected in our implementation, since the processors are initialized one at a time,

Fig. 4(b) shows the variation of total Perl CPU time with number of processors. It can be seen that there is no significant difference in total Perl CPU time for different numbers of processors. This matches with what one might expect. The Perl code has to process the same number of chromosomes, regardless of the number of processors used for fitness evaluation.

Fig. 5(a) shows the variation of total IDL evaluation CPU time with number of processors. As with total Perl CPU time, it can be seen that there is no significant difference in total IDL evaluation time for different number of processors. Once again, this matches with what might be expected. On average, 1000 chromosome evaluations will take the same total amount of CPU time, regardless of the number of processors.



(a)



(b)

Figure 4. (a) Variation of initialization time with number of processors (b) Variation of total Perl CPU time with number of processors

Fig. 5(b) shows the variation of total wall-clock run-time with number of processors. It can be seen that there is a rapid decrease in time taken to perform a run, with increase in number of processors, up to about 5 processors, which appears to be the point of diminishing returns.

We can create a simple model for the variation of wall clock time with number of processors. We initially assume zero communication overhead. Let W_t be the wall clock time for the run, N be the number of processors used for fitness evaluation, I be the rate at which the initialization time increases with number of processors (i.e. the gradient of the initialization-processor graph), P be the total Perl process time for the run (which is proportional to the number of chromosomes to be evaluated) and E be the total IDL evaluation time (again, proportional to the number of chromosomes to be evaluated). Therefore we can estimate the total wall clock time for a run as:

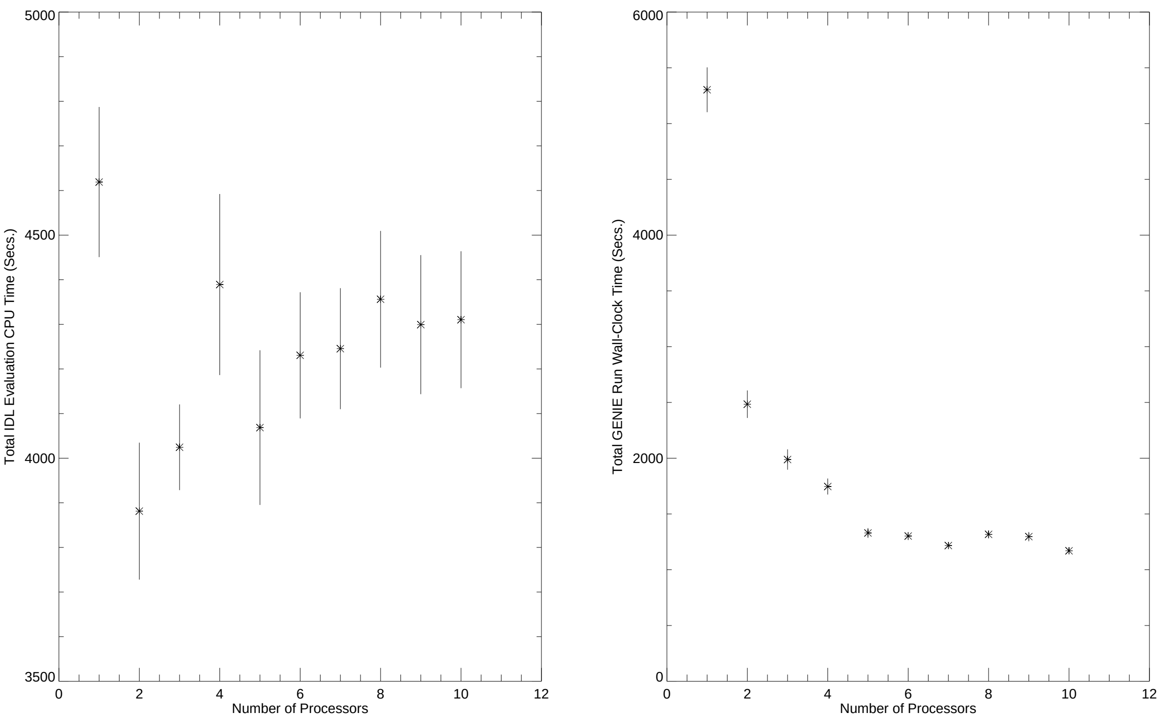
$$W_t = (N \times I) + P + (E/N) \tag{2}$$

If we examine the plot of initialization time against processor number we see that the line does not actually pass through the origin, so we include a small offset term (system overhead?), δ, giving us:

$$W_t = ((N \times I) + \delta) + P + (E/N) \tag{3}$$

We can estimate the value of I and δ by determining the best fit straight line through the plot of initialization time against processor number. We can estimate the values P as the mean of the total Perl CPU times for all runs. Likewise we can estimate the value of E as the mean of the total IDL evaluation times for all runs. This gives us the following:

$$W_t = ((21.48 \times N) + 10.52) + 111.29 + (4242.49/N) \tag{4}$$



(a) (b)

Figure 5. (a) Variation of total IDL evaluation CPU time with number of processors (b) Variation of total wall clock time with number of processors

Thus, we have an initialization time per processor of 21.5 secs., a total Perl process time of 111 secs., and a total IDL evaluation process time of 4242 secs. (70 minutes).

We can plot this function and compare it to the measured wall clock time. Fig. 6(a) shows this comparison.

It can be seen that the measured wall clock time correlates extremely well with the predicted. The slight difference in the two plots could be attributed to the fact that no account was taken for communication time. This would be an additional constant term in the predictive equation and would provide a vertical offset which could bring the two plots closer together. By including the communication time offset, (let this be denoted as Ω) in the model, we get:

$$W_t = ((N \times I) + \delta) + P + (E/N) + \Omega \tag{5}$$

We can estimate this offset, by simply taking the mean of the differences between the two plots (measured and predicted wall-clock run-times). Doing this gives:

$$W_t = ((21.48 \times N) + 10.52) + 111.29 + (4242.49/N) + 432.88 \tag{6}$$

The measured run time and predicted run time now have a much better correlation, as can be seen in Fig. 6(b):

With the above model we can predict the number of processors necessary to attain the minimum total run time. If we differentiate equation 4 with respect to N, we can determine that for a minimum wall clock time, we would require 14 processors.

We can also look at the parameters that affect the number of processors that produce the minimal wall clock time. Let C be the total number of chromosomes evaluated, D be the size of data to be processed (i.e. size of

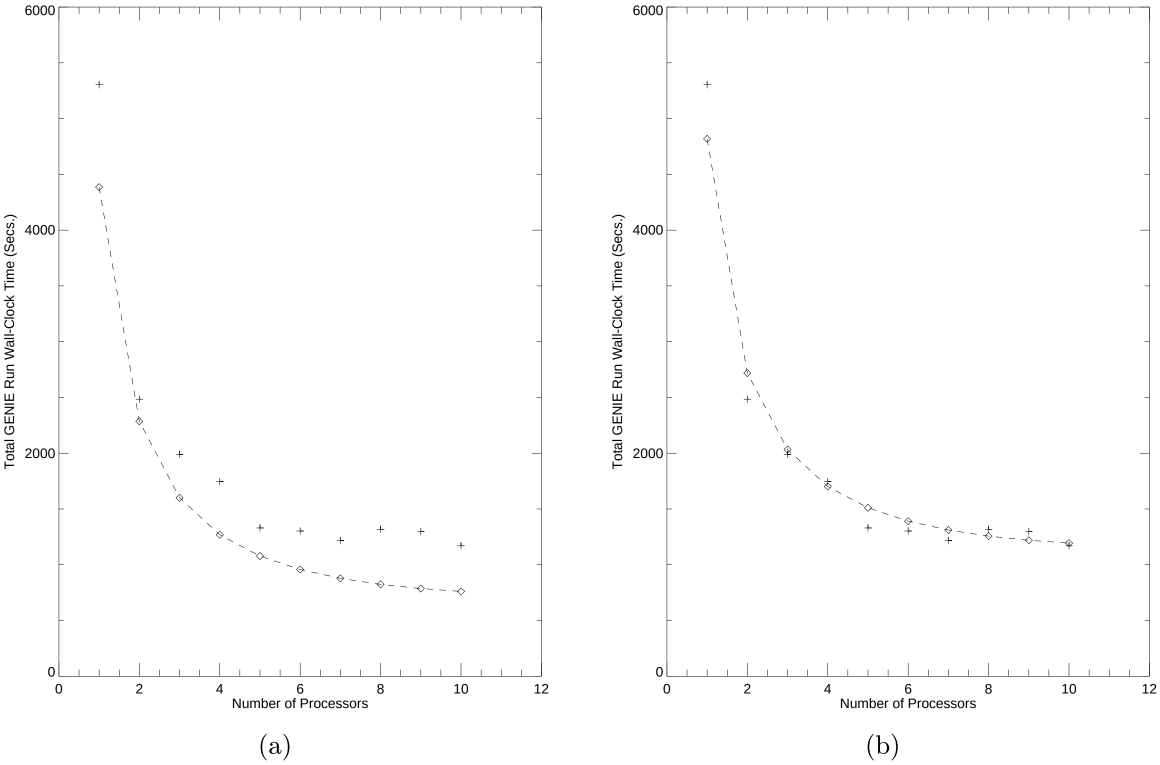


Figure 6. (a) Comparison of measured wall-clock run-time with predicted wall-clock run time, not including communication time, (b) Comparison of measured wall-clock run-time with predicted wacll-clock run-time, including communication time. (Note: crosses represent the measured wall-clock run-time, diamonds the predicted wall-clock run-time)

the problem). We can further parameterize equation 5. With reference to equation 5, in the following equation, $I = (\Phi \times D)$, $P = (\Lambda \times C)$, $E = (\Psi \times C)$ and $\Omega = (\Gamma \times C)$:

$$W_t = ((N \times \Phi \times D) + \delta) + (\Lambda \times C) + ((\Psi \times C)/N) + (\Gamma \times C) \tag{7}$$

Differentiating with respect to N and looking for a minimum, we get the following relationship for N:

$$N = \sqrt{(\Psi/\Phi) \times (C/D)} \tag{8}$$

So, we can see that the number of processors required for minimum run-time is proportional to the square root of the ratio of the number of chromosomes to be evaluated to the size of the data to be processed (i.e. the size of the problem).

4. APPLICATION

For a real-world application, GENIE was applied to the problem of finding burned regions around Los Alamos, resulting from the Cerro Grande fire. The input data to genie was actually two Landsat data collections, from different times: one collected on May 9th 2000 and one collected on June 1st 2000. These data sets were stacked together so that, instead of having two single 6-band landsat data cubes, we had one single 12-band data cube. With this data GENIE could make use of the temporal information, in addition to spectral and spatial information. Fig. 7(a) shows some of the data from the may 9th collection.

Training data was marked up using ALADDIN by an analyst, using knowledge of official burned-area data. Fig. 7(b) shows the training data, where white indicates pixels marked as "True", grey indicates pixels marked as "False" and black indicates pixels marked as "Unknown".

GENIE was run for 150 generations, with a population size of 50 chromosomes. The best algorithm found at the end of this run had a fitness of 922, with respect to the training data. This fitness score translates into a detection rate of 99.1 % and a false alarm rate of 0.6 %. The results of the resultant burn-mask is shown in Fig. 8(b).

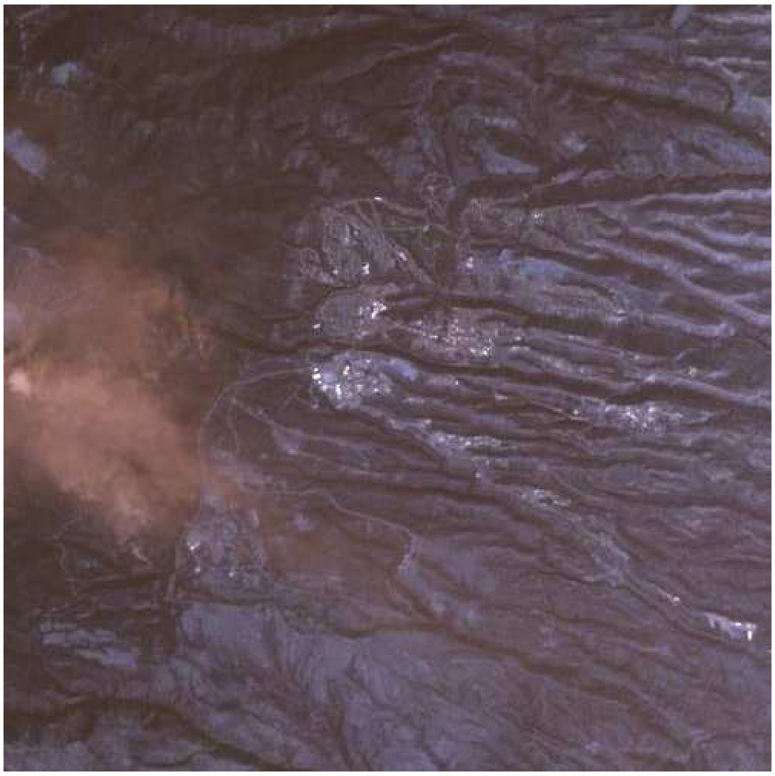
Subjectively, these results can be seen to provide an accurate map of the burned areas. Objectively, these results have had their accuracy confirmed by officials involved in the recovery operation and have been corroborated by actual ground-truth observations.

5. CONCLUSIONS

We have demonstrated the ability to significantly decrease run-times for GENIE runs, by employing a simple technique of farming out chromosome evaluations to a group of processors. We have investigated the effect of number of processors on run times and developed a mathematical model to predict process run times. We have shown that the mathematical model compares very well with the measured results. We have also shown the application of our GENIE software to a real-world problem, and confirmed the accuracy of the results with actual ground-truth.

REFERENCES

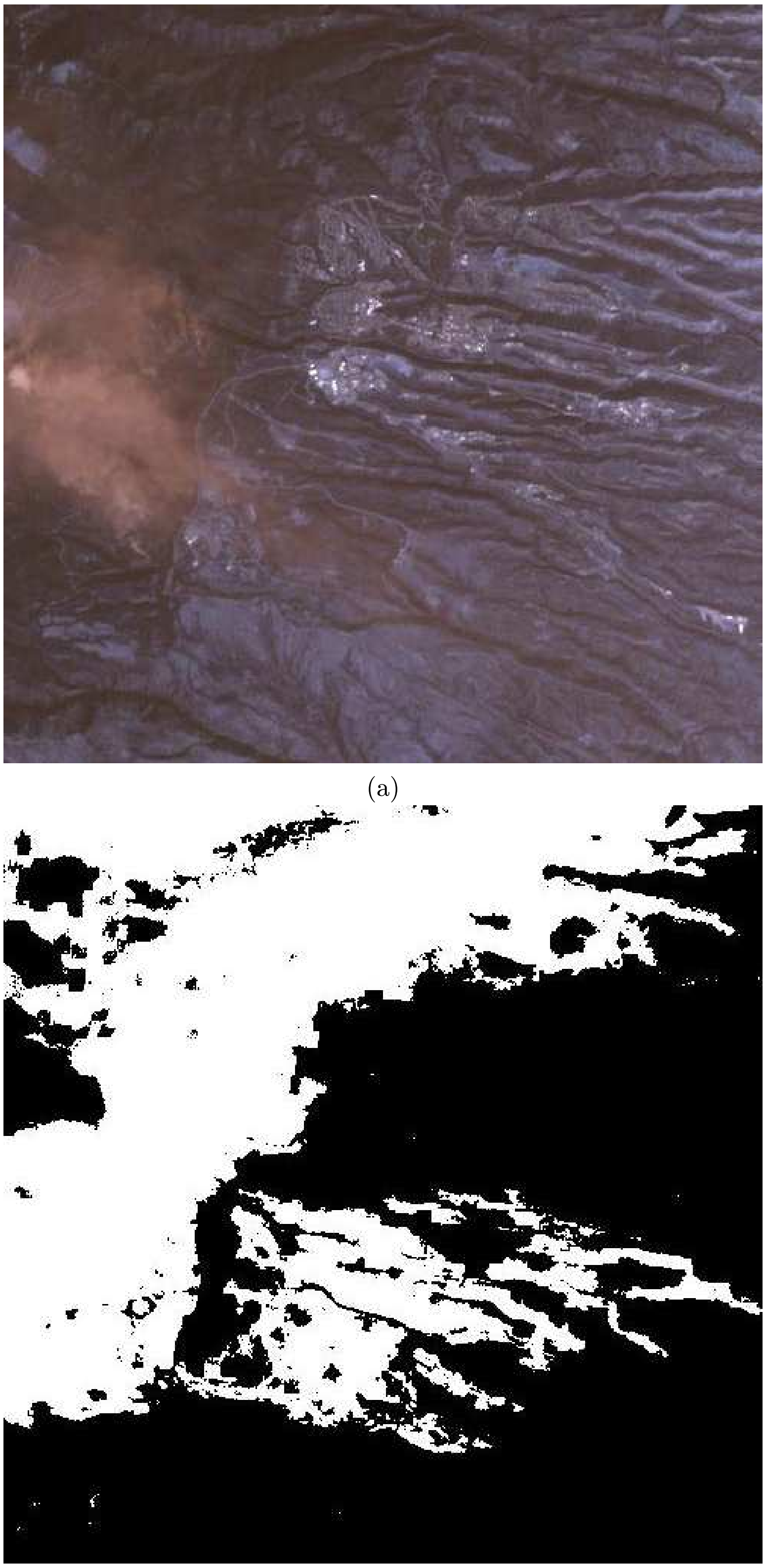
1. T. Dandekar and P. Argos, "Potential of genetic algorithms in protein folding and protein engineering simulations", *Protein Engineering*, 5(7), pp. 637–645 (1992).
2. L.A. Cox Jr., L. Davis and Y. Qiu, "Dynamic anticipatory routing in circuit-switched telecommunications networks", in *Handbook of Genetic Algorithms*, L. Davis, ed., pp. 124-143, Van Nostrand Reinhold (1991).
3. N.R. Harvey and S. Marshall, "GA Optimization of Spatio-Temporal Grey-Scale Soft Morphological Filters with Applications in Archive Film Restoration" , in *Evolutionary Image Analysis, Signal Processing and Telecommunications*, Poli et al. eds., pp. 31–45, Springer-Verlag (1999).
4. J.M. Daida, J.D. Hommes, T.F. Bersano-Begey, S.J. Ross and J.F. Vesecky, "Algorithm discovery using the genetic programming paradigm: Extracting low-contrast curvilinear features from SAR images of arctic ice", in *Advances in Genetic Programming 2*, P.J. Angeline et al. eds., Chap. 21, MIT Press (1996).
5. S.P. Brumby, J. Theiler, S.J. Perkins, N.R. Harvey, J.J. Szymanski, J.J. Bloch and M. Mitchell, "Investigation of Image Feature Extraction by a Genetic Algorithm", in *Proc. SPIE 3812* pp. 24–31 (1999).
6. J. Theiler, N.R. Harvey, S.P. Brumby, J.J. Szymanski, S. Alferink, S. Perkins, R. Porter and J.J. Bloch, "Evolving Retrieval Algorithms with a Genetic Programming Scheme", in *Proc. SPIE 3753*, pp. 416–425 (1999).
7. S.P. Brumby, N.R. Harvey, S. Perkins, R.B. Porter, J.J. Szymanski, J. Theiler and J.J. Bloch, "A genetic algorithm for combining new and existing image processing tools for multispectral imagery", in *Proc. Aerosense 2000*, Orlando, Florida, April 2000.
8. N.R. Harvey, S. Perkins, S.P. Brumby, J. Theiler, R.B. Porter, A.C. Young, A.K. Varghese, J.J. Szymanski and J. Bloch, "Finding golf courses: The ultra high tech approach", in *Evolutionary Image Analysis, Signal Processing and Telecommunications*, Poli, et al. eds., pp. ??–??, Springer-Verlag (2000).
9. J.R. Koza, *Genetic programming: On the Programming of Computers by Means of Natural Selection*, MIT Press (1992).
10. C.M. Bishop, *Neural Networks for Pattern Recognition*, pp. 105–112, Oxford University Press (1995).
11. W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery, *Numerical Recipes in C, 2nd Edition*, pp. 402–405, Cambridge University Press (1992).
12. P.G. Weber, B.C. Brock, A.J. Garrett, B.W. Smith, C.C. Borel, W.B. Clodius, S.C. Bender, R.R. Kay, M.L. Decker, "Multispectral thermal imager mission overview", *Proc. SPIE 3750*, pp. 340–346 (1999).



(a)

(b)

Figure 7. (a) Landsat data of Los Alamos, May 9th 2000, (b) Training data (white: *true*, grey: *false*, black: *unknown*



(a)

(b)

Figure 8. (a) Landsat data of Los Alamos, May 9th 2000, (b) Results from GENIE run