Investigation of Image Feature Extraction by a Genetic Algorithm

Steven P. Brumby [a*], James Theiler [a], Simon J. Perkins [a], Neal Harvey [a],

John J. Szymanski[a], Jeffrey J. Bloch [a], and Melanie Mitchell[b]

[a] Los Alamos National Laboratory, Space and Remote Sensing Sciences,
Mail Stop D436, Los Alamos, NM 87545
[b] Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM 87501

ABSTRACT

We describe the implementation and performance of a genetic algorithm (GA) which generates image feature extraction algorithms for remote sensing applications. We describe our basis set of primitive image operators and present our chromosomal representation of a complete algorithm. Our initial application has been geospatial feature extraction using publicly available multi-spectral aerial-photography data sets. We present the preliminary results of our analysis of the efficiency of the classic genetic operations of crossover and mutation for our application, and discuss our choice of evolutionary control parameters. We exhibit some of our evolved algorithms, and discuss possible avenues for future progress.

Keywords: Evolutionary computation, genetic algorithms, image analysis, multi-spectral analysis.

1. EVOLVING TOOLS FOR IMAGERY FEATURE EXTRACTION

Extraction of features of interest from large and possibly heterogeneous imagery data sets is a crucial task facing many communities of end-users. From broad-area environmental monitoring, to space-based commercial prospecting, to modern cartography, workers and researchers have access to highly capable collection platforms operating in a range of spectral bands. With new distribution technologies and data formats making dissemination of this data progressively cheaper and easier, the bottle-neck to successful exploitation of this information rests more than ever on the availability of suitable analysis tools. Development of these tools is an expensive business, often requiring a significant investment of time by highly skilled analysts. National and commercial pressures have provided the necessary impetus for the development of mature tools for traditional data formats (e.g., electro-optical imagery, panchromatic and/or visible/infrared) for specific tasks of perceived importance, but novel situations can arise and existing tools may be proprietary or classified. The generality of analysis tools is itself an issue of importance. Mature tools frequently exhibit strong specialization. Further, with the advent of multi-spectral sensors platforms such as LANDSAT and SPOT, the analyst can now search for spectral, spatial, and possibly hybrid spatio-spectral signatures, requiring development of whole new tool-kits. Our own work in the field of remote sensing has led us to seek an accelerated toolmaker. Since creating and developing individual algorithms is so important and yet so expensive, we have recently begun investigating an evolutionary approach to this problem.

Over the last two decades, ideas taken from the theory of evolution in natural systems have inspired the development of a group of powerful yet extraordinarily flexible optimization methods known collectively as evolutionary computation (EC). The modern synthesis derives from work performed in the 60s and 70s by researchers such as Holland[1]; Rechenberg[2]; and Fogel et al.[3]. While the various schools founded by these pioneers have differences, their approaches share the common themes of optimization performed by a competing population of individuals in which a process of selection and reproduction with modification is occurring.

The beauty of EC is its flexibility: if we can derive a fitness measure for a problem, then the problem might be solved using EC. Many different problems from different domains have been successfully tackled using EC, including: optimization of dynamic routing in telecommunications networks[4]; designing finite-impulse-response digital filters[5]; designing protein sequences with desired structures[6]; and many others.

A crucial issue when using EC is how to represent candidate solutions so that they can be manipulated by EC effectively. We wish to evolve individuals that represent possible image processing algorithms, and so we will use a system based upon genetic programming[7]. Genetic programming (GP) is essentially a framework for developing executable programs using EC

[*] Work supported by the U.S. Departments of Energy and Defense. Further author information: (Send correspondence to S.P.B)
S.P.B.: E-mail: brumby@lanl.gov

methods. GP has been the subject of a huge amount of research this decade and has been applied to a wide range of applications, from circuit design[7], to share price prediction[8]. With particular relevance to this proposal, GP has also been applied to image-processing problems, including: edge detection[9]; face recognition[10]; image segmentation[11]; image compression[12]; and feature extraction in remote sensing images[13]. Daida's work[13] is particularly relevant since it demonstrates that GP can successfully evolve algorithms for real tasks in the remote-sensing domain.

We have called our software GENIE, which stands for Genetic Imagery Exploitation. In the next section we give an overview of our software architecture. In the following section we provide some preliminary results obtained using our prototype system.

2. DESIGN OVERVIEW

Our research follows the classic evolutionary paradigm. A population of candidate solutions is evaluated by measuring their fitness with regards to a given environment. Natural selection occurs, and the survivors are allowed to reproduce. This process continues until some predefined halting condition is satisfied.

Our particular task is to evolve image-processing tools ("candidate solutions") for imagery feature extraction. The environment is a set of training data. Selection proceeds using a fitness function, and reproduction is attended by crossover and mutation, with elitism available as an option. We now discuss each component in some detail.

2.1 Environment: Training Data

Consider a data cube of multi-spectral data. In particular, we only consider spatially co-aligned data, so that each data plane contains the same number of pixels, and spatial information is orthogonal to spectral information. Data from different sensors and/or sensor-platforms with different spatial/spectral resolutions can be combined in principle, but we assume that the co-registration of data has already taken place and that all data planes have equal spatial resolution.

For our initial work, we have relied on two major sources of imagery: MODIS Airborne Simulator (MAS) data (obtained from NASA[14]), and Digital Orthoquod Quaterquad (DOQ) data (obtained from the United States Geological Survey[15]).

The Moderate Resolution Imaging Spectroradiometer (MODIS) is an instrument designed for NASA's Earth Orbiting System (EOS) and will be the key instrument aboard the Terra (EOS AM-1) satellite (planned launch date: mid 1999). The MAS, carried by a high-altitude NASA ER-2 aircraft, simulates the full MODIS instrument, providing 50 narrow, contiguous bands of visible to infrared with 50 meter ground resolution from 20 km altitude, a wide (37.25 km) field of view, and produces 12-bit digitized data.

The DOQ data provides orthorectified aerial photography in squares of 3.75' longitude x 3.75' latitude. Panchromatic, color, and color-infrared (green/red/near infrared [0.5-0.9um]) data products are available, with approximately 1 meter ground resolution from 20,000 feet. We have used the color-infrared product, which comes in 3 band, 8-bit digitized format.

In seeking to evolve image-processing tools, the chief challenge faced by machine-learning approaches is the provision of sufficient quantities of good quality training data. That is, "ground truth" for some subset of the imagery (the training data) must be specified, either by inspection of the raw data and hand mark-up by a human analyst, or by incorporating data after actually surveying the terrain, or by application of a known algorithm. The latter case allows us to test our ability to recover known algorithms, e.g., the normalized difference vegetation index (NDVI, see e.g. Schowengerdt[16]). Specification of the ground truth by hand can be laborious, pre-supposes that the feature of interest is known beforehand, and assumes that the feature can be identified by inspection in at least one plane of the imagery data cube.

In either case, the training data cube is augmented by what we call a truth plane. This can contain a binary mask labeling each feature pixel with a 1 and each non-feature pixel with a 0. Alternatively, the truth "plane" may itself be composed of two or more planes to allow different weightings of pixels to guide the GA in finding the most important pixels. The truth values can be extended to byte or float type. In this way, our system can evolve a feature finder that provides confidence values as well as yes-no classification.

One general feature of GP is the ability of parse trees to grow in complexity via crossover. Motivated by concerns over computational resources, we decided to limit this growth of complexity by defining a fixed number of read/write memory ("scratch") planes. Data planes, on the other hand, are read only, and the number of data planes in also fixed. For our initial work, we have decided to call the final value in scratch plane A the "answer". Evaluation of the fitness of a candidate tool depends solely on the final result it has stored in the answer plane. In general there are several scratch planes available to a candidate tool, and it is interesting to ask what information we may find in those additional planes. To take advantage of this

feature, we are currently investigating more complicated fitness evaluation schemes which would evaluate the final contents of all the scratch planes.

With our scratch plane formulation, any GP parse tree can be broken into a set of single vertex/operation sub-trees (somewhat reminiscent of Koza's automatically defined functions[7]). Results stored in a scratch plane may be reused throughout the image-processing algorithm, so we are actually dealing with graphs rather than trees. We feel that this potential for reuse of intermediate results is an important capability, but also expect and find that this introduces complications for crossover, discussed below. For any GP tree there exists a minimum number of scratch planes that allows representation of the tree in our scratch plane formulation.

2.2 Genes: Image Analysis Primitives

We have introduced a simple notation for encoding individual genes, e.g., [ADDP, ra, rb, wA, 1.0], which reads data planes a and b, and writes the sum of these planes to scratch plane A. The final number in the block is a parameter allowing weighted sums of planes, i.e., a + p*b. We refer to everything within the square brackets [] as the gene.

Conceptually, any system such as ours is exploring the algebra of image operators. That is, how to form useful combinations of image processing operators, and how to identify the generators of this algebra. If this is to be considered in all its generality, then one is confronted by a large, unsolved problem of image analysis (for work in this direction, see, e.g., Ritter and Wilson[17]). While we are interested in this theoretical problem, our initial approach has been to specify a set of "useful" primitive operators and explore the space of algorithms that they generate. We have chosen a set of logical, threshholding, spatial, and spectral operators, outlined in Table 1. The user can specify which operators are available in any given run.

Logic:	AND,
	OR,
	NOT,
Threshhold:	CLIP_HI,
	CLIP_LO,
	BOOL,
	EXPAND
	GT,
	LT,
Spectral:	ADDP,
	ADDS,
	SUBP,
	SUBS,
	MULTS,
	DIVS,
	LINSCL,
	LINCOMB,
	BANDRATIO,
	NDVI
Spatial:	MEAN,
	MEDIAN,
	ERODE,
	DILATE,
	OPEN,
	CLOSE,
	OPEN_CLOSE,
	CLOSE_OPEN

Table 1. Primitive Image Operators for GENIE

Due to the nature of the image operator representation in the context of the chromosome, the set of morphological operators able to be included in the "gene pool" is restricted to function-set processing morphological operators, i.e. gray-

scale morphological operators having a flat structuring element. Within this set of operators we further restrict the usable set to those commonly used operators which can be constructed from the basic morphological operators of erosion and dilation. These operators are sometimes referred to as primary, secondary and tertiary operators, due to their construction, with respect to the basic or primary morphological operators. These morphological operators are, *primary*: erosion (ERODE) and dilation (DILATE), *secondary*: closing (CLOSE; dilation followed by erosion) and opening (OPEN; erosion followed by dilation) and *tertiary*: open-closing (OPEN_CLOSE; opening followed by closing) and close-opening (CLOSE_OPEN; closing followed by opening). The sizes and shapes of the structuring elements used by these operators, again, due to the particular genetic representation, is also restricted to a pre-defined set of primitive shapes, which includes, square, circle, diamond, horizontal cross and diagonal cross. The choice of shape for the structuring element is defined by one of the parameters of the operators genetic representation. Likewise for the size of the structuring element.

Other local neighborhood/windowing operators such as mean, median, etc. have their kernel/window size and shape incorporated into their genetic representation in a similar manner. Our spectral operators have been chosen to permit weighted sums, differences and ratios of data and/or scratch planes, and various rescalings of single data or scratch planes. Operators such as ADDP and SUBP add and subtract (respectively) pairs of data and/or scratch planes, while operators such as DIVS, MULTS, and ADDS apply scalar factors. LINCOMB calculates linear combinations of planes. NDVI is essentially a ratio of a difference over a sum of planes (i.e., (a-b)/(a+b)), which gains physical significance when the planes are chosen from certain spectral regions (red and near infrared).

Finally, we have introduced threshholding and logical operations to allow the system to match Boolean ground truth. At the moment there is no requirement that the system apply operators in any predefined order. We have decided to take advantage of this, by defining all our operators to work with float, byte or Boolean data. For example, the AND operator essentially carries out a pixel by pixel product of images, which reduces to the standard Boolean AND when both inputs are Boolean, and can give be given a probabilistic interpretation when one or both of the inputs is of type float or byte.

The complexity of our "primitive" operators is, at this stage, mostly a matter of taste. For example, operators such as BOOL, which turns a gray-scale image into a binary image using an evolvable threshhold, is quite simple to implement and quick to execute. A morphological operator such as OPEN, on the other hand, requires specification of a structuring element (which can evolve), and is in fact the composition of the ERODE and DILATE operators in our basis set. Yet we know from experience that OPEN is a frequently used morphological operator, and so we have made it available from the outset. A function such as NDVI may also be composed out of other genes in our basis set, but we include it to allow exploration of our system's ability to identify the correct inputs for this important remote-sensing algorithm.

In all of this, our desire is to give the GA a head-start by providing a rich set of building blocks, without wanting to include operators much larger and/or slower than the rest of the primitive operators. Given this arbitrariness, we have made a point of ensuring that additional operators can be added quickly and easily to the current set of basis operators. Indeed, one of our aims is to evolve useful combinations of genes that can be promoted to new, indivisible building blocks.

2.3 Chromosomes: Representation of Candidate Solutions

Chromosomes are text strings of genes, e.g.,

```
[ADDP rb rd wA 1.0][LT rA wC 0.23][NOP][MAX rA rb wA]
```

Chromosome length (measured in genes) is set at the start of a run, and is currently fixed for the duration of the run. We have included a null operator ([NOP]) to allow an effective length of less than the maximum length.

We have implemented several checks to reduce unnecessary computations. In any given chromosome a number of genes will not contribute to the final answer. For example, they may write to a scratch plane other than A, that is never subsequently used. Or they may write to a scratch plane that is overwritten before the stored result is used. These "junk" genes can be effectively stripped out of the full chromosome immediately before evaluation on the training data, by a process of checking the dependency of the answer plane on the genes. To do this, we have implemented a scheme for calculating the GP parse tree that determines the answer in scratch plane A. Genes not contributing to this tree are not evaluated. Their contents are preserved for future evolution.

More subtle redundancies are not stripped out, so for example we do not detect an operator followed by its inverse. With the checks we have introduced, after including the extra time required to strip the junk genes, we have seen reductions in time of execution of complete runs, approaching factors of 10 to 30.

Regarding programming language, we have implemented our genotype-level code in object-oriented Perl. This provides a convenient environment for the string manipulations required by the genetic operations of mutation and crossover, and simple access to the underlying operating system (Linux on high-end workstations). Evaluation of the chromosomes is the

computationally expensive part of each evolutionary cycle. For that task, we have chosen a commercial product, RSI's IDL language and image processing environment (see http://www.rsinc.com/idl). IDL offers a rapid prototyping capability, and together with RSI's ENVI package for IDL, provides support for the various remote-sensing data formats we have been using. IDL also provides a rich environment for visualizing the results of our evolution. This is not used during the evolution of algorithms, but is useful for interacting with the evolved algorithms and their components.

Within IDL, individual genes correspond to single primitive image operators, which are coded as IDL batch executables. An advantage of IDL is that implementation of most of our primitive operators require only a few lines of code, simplifying debugging. For example, the previously displayed chromosome is written to an IDL batch file containing the lines,

```
ga_addp, data_b, data_d, scratch_A, 1.0
ga_lt, scratch_A, scratch_C, 0.23 ; junk, can be ignored
; no operation
ga_max, scratch_A, data_b, scratch_A
```

In these lines of code, the ga_xxx nomenclature is designed to match the gene labeled XXX. The ";" indicates a comment in the IDL programming language.

With our checking procedures, only the first and fourth lines would be passed to the IDL session for evaluation on the training data. In our current implementation, this IDL session is opened at the beginning of the run, and communicates with the Perl code via a two-way unix pipe. This is a low-bandwidth connection; only the IDL session needs to access the training data (several hundred Megabytes in current trials), which requires a high-bandwidth connection.

2.4 Population

A population is a set of chromosomes. The number of chromosomes in the population can be specified, but is fixed throughout the run. For our initial tests, we have chosen to use populations of a 50 to a few hundred, and have done some runs with much smaller and much larger population sizes for comparison. For system development purposes, we have generally settled on population of the order of 200 candidate tools.

We keep track of individual chromosomes' scores to avoid re-evaluation on the training data. We also accumulate a "zoo" of all previously evaluated effective chromosomes (i.e. stripped of junk) and their scores, to avoid evaluating genes where reproduction has not changed the effective algorithm. Our experience has been that this check (comparing with the zoo) does significantly speed up the execution of a run by an additional factor of 2 to 5, depending on the specified maximum chromosome length and choice of available basis operators.

2.5 Selection: Choices for Crossover and Mutation

Fitness evaluation uses a fitness function based on a scaled (Manhattan metric) distance between answer plane and ground truth. We give candidate tools credit for finding the anti-feature. That is, if a tool finds everything but the feature of interest then the tool receives a high fitness (achieved by taking the absolute value of the distance). The fitness is scaled so that an exact match to the ground truth receives a score of 100%, while a totally random algorithm will receive a zero score. An algorithm that produces a constant value output will also get a score of zero.

Selection for reproduction has been implemented through simple rank selection with replacement. We intend to investigate tournament selection schemes at a latter date.

With so many genes and such different gene types (spectral, spatial, logical), we have introduced a means of handling specification of the results of a mutation event during reproduction. Essentially, we make use of the object-oriented philosophy to specify a mutation mechanism for each gene, explaining how it mutates. So, for example, a gene that adds two planes and writes to a single scratch plane can mutate by changing one or more of the input and output planes (without changing the number of inputs and outputs), or change the weighting parameter in the sum. Alternatively, it can mutate into an entirely new gene, which is then given randomly generated input/outputs and parameters (type and range are specified for each gene). A more complex operator, such as a convolution operator, would mutate with respect to both size and shape of the convolution kernel.

The mutation rate is currently fixed throughout a run, and is generally given a probability of 0.5 (probability that a single gene within the chromosome will mutate in some way; only one gene per chromosome can mutate, though other schemes

have also been implemented and are currently under investigation). Crossover has been implemented as single point crossover at a randomly generated site (with probability 0.9). This means that, in general, crossover will greatly disrupt the algorithm. We recognize that our mutation and crossover probabilities are high relative to some binary string GA research[18]

An alternative approach would be to impose a great deal of structure on the chromosome, actually specifying a tree structure and demanding the crossover conserve this global structure (ie, the predefined global topology). Preliminary work in this direction is underway within our group, and is reported elsewhere (see Porter, McCabe and Bergmann[19]).

We have also added "elitism" as an option. A selectable number of highly fit members of the current population may be passed unaltered to the next generation. There is no limit to how many generations a chromosome can live. Elitism is usually restricted to 5 or 10% of the current population. In our trials, addition of elitism has been seen to help convergence by the population on a good solution.

3. PRELIMINARY RESULTS

For our initial attempt to evolve an image-processing algorithm, we turned to the important and well-studied problem of finding open water (rivers and lakes) amidst vegetation. Figure 1 shows a near infrared channel from our MODIS airborne simulator data set (the location is an estuary leading to the Gulf of Mexico). Clearly, on inspection, the main body of water stands out to a human observer looking at this single band. Complication arises when we try to delineate the boundary of the water, and when we try to distinguish between open bodies of water, wet fields, and clouds. Also presented is the result of a human designed algorithm (Szymanski and Alferink[20]), that combines comparisons of spectral channels, and simple measures of texture (standard deviations in pixel neighborhoods).

Using the output of this algorithm for our ground truth, we have completed runs of GENIE with varying population size, different basis operators, varying mutation and crossover rate, and amount of training data. GENIE successfully evolved simple algorithms that match the ground truth at the >98% level (Fig. 2). In each case, our system had discovered a threshhold on one of the spectral channels that segmented the image in a way that closely matched the human designed algorithm. We evolved first with only spectral, and then with both spectral and spatial operators. Because of the nature of this particular test case, spectral operators alone were able to achieve high-fitness solutions. We also evolved spatial/spectral solution that achieved similar scores on our training and test data. We are now in the process of expanding our investigation to see how these algorithms fare with a wider range of data sets, and will investigate the suitability of these evolved algorithms as starting points for future runs.

As for comparison of mutation and crossover, we have consistently found that, even with our simple form of crossover and reasonably short chromosome lengths (8 to 20 genes per chromosome), crossover and mutation together consistently beat mutation alone, with respect to time to reach a high-fitness solution. We only claim this as a preliminary result, and are currently undertaking the large number of runs required for definite conclusions. We are also investigating evolution of algorithms for the DOQ data set and for other features in both data sets.

4. CONCLUSION

We have implemented a genetic programming scheme for evolving image-processing algorithms. We have initial results for some simple feature extraction tasks, and are now investigating our genetic algorithm parameter space. By imposing checks and removing junk genes, we have reduced our runtimes by a factor of order 100. Our preliminary findings are that an EC approach to accelerated image analysis tool making is possible, and that this may be a successful approach to rapidly developing algorithms for novel features and/or for novel data sets.

ACKNOWLEDGEMENTS

This work was funded by the US Departments of Energy and Defense. The authors would like to thank Steven Alferink for use of the water-finding algorithm.

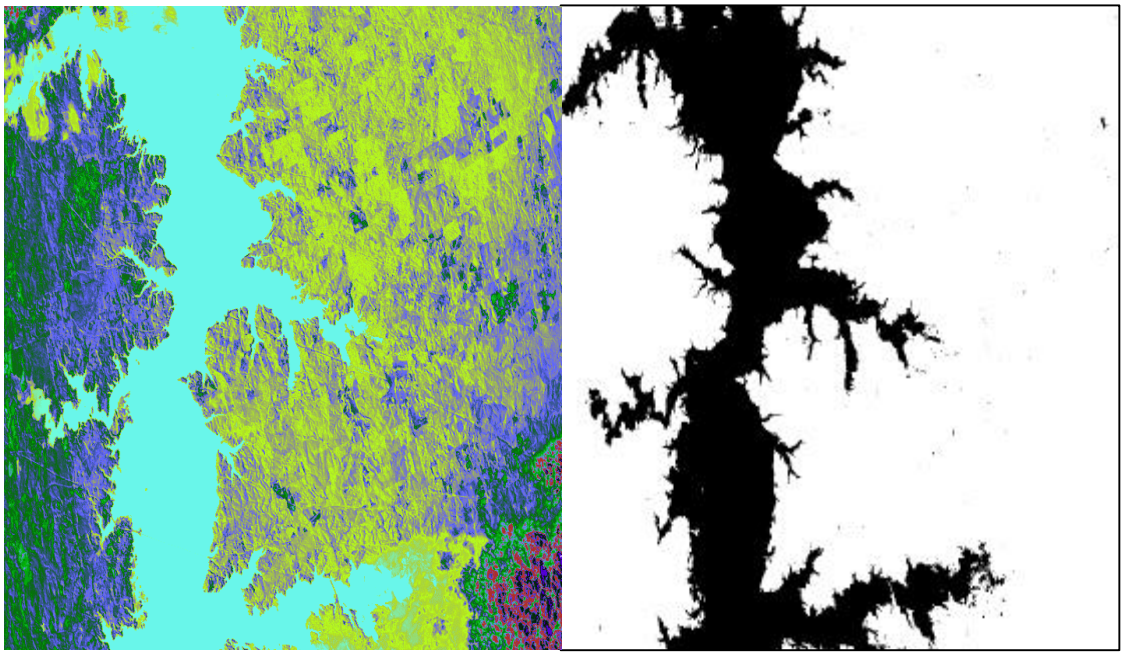

Figure 1. Water Mask. The left figure is a single plane from our multi-spectral MAS data set. The figure on the right shows the output of a human designed water-finding algorithm (Szymanski and Alferink[20])

GENIE
fitness 98.6%

```
[MIN  rd rd wC]
[MIN  rc rd wA]
[LT  rC wC 0.31]
[LT  rd wA 0.2]
[BOOL  rA wA 0.2]
[MIN  rC rd wA]
[MAX  rA rd wA]
[BOOL  rA wA 0.2]
```

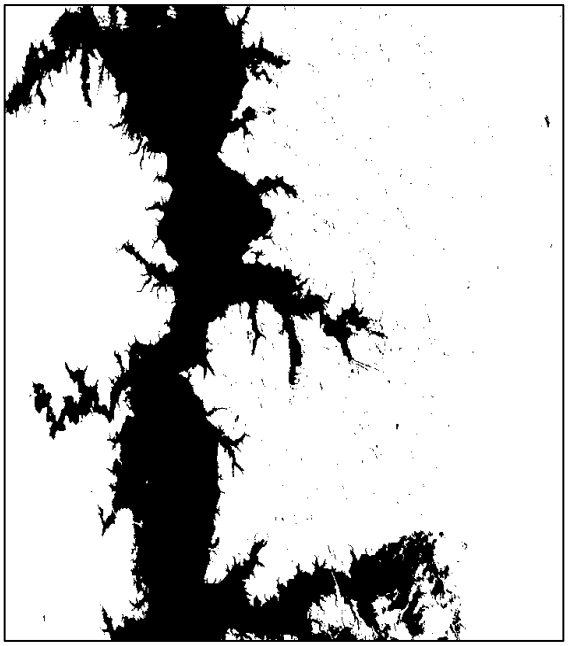

Figure 2. Evolved solution. GENIE evolved this solution using a population of 200 individuals over 30 generations. The evolved algorithm is exhibited on the left. The answer is highly redundant, and essentially consists of a threshhold on one channel (done on line 3, the Less-Than threshholding operator). This is a limitation of the chosen test case.

REFERENCES

1. J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan, Ann Arbor, 1975.
2. I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Fromman-Holzboog, Stuttgart, 1973.
3. L. Fogel, A. Owens and M. Walsh, *Artificial Intelligence through Simulated Evolution*, Wiley, New York, 1966.
4. 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, New York, 1991.
5. J. D. Schaffer and L. J. Eshelman, "Designing multiplierless digital filters using genetic algorithms," *in Proceedings of the Fifth International Conference on Genetic Algorithms*, S. Forrest, ed., pp. 439-444, Morgan Kaufmann, San Mateo, 1993.
6. T. Dandekar and P. Argos, "Potential of genetic algorithms in protein folding and protein engineering simulations," *Protein Engineering* **5(7)**, pp. 637-645, 1992.
7. J. R. Koza, *Genetic Programming: On the Programming of Computers by Natural Selection*, MIT, Cambridge, 1992.
8. G. Robinson and P. McIlroy, "Exploring some commercial applications of genetic programming," *in Evolutionary Computing, Volume 993 of Lecture Notes in Computer Science*, T.C. Fogarty, ed., Springer-Verlag, Berlin, 1995.
9. C. Harris and B. Buxton, "Evolving edge detectors", Research Note RN/96/3, University College London, Dept. of Computer Science, London, 1996.
10. A. Teller and M. Veloso, "A controlled experiment: Evolution for learning difficult image classification" in *7th Portuguese Conference on Artificial Intelligence, Volume 990 of Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1995.
11. R. Poli and S. Cagoni, "Genetic programming with user-driven selection: Experiments on the evolution of algorithms for image enhancement," in *Genetic Programming 1997: Proceedings of the 2nd Annual Conference*, J. R. Koza, et al., editors, Morgan Kaufmann, San Francisco 1997.
12. P. Nordin, and W. Banzhaf, "Programmatic compression of images and sound," in *Genetic Programming 1997: Proceedings of the 2nd Annual Conference*, J. R. Koza, et al., editors,, Morgan Kaufmann, San Francisco, 1996.
13. 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 and K. E. Kinnear, Jr., editors, chap. 21, MIT, Cambridge, 1996.
14. M. D. King, W. P. Menzel, P. S. Grant, J. S. Myers, G. T. Arnold, S. E. Platnick, L. E. Gumley, S. C. Tsay, C. C. Moeller, M. Fitzgerald, K. S. Brown, and F. G. Osterwisch, "Airborne scanning spectrometer for remote sensing of cloud, aerosol, water vapor and surface properties," *J. Atmos. Oceanic Technol.* **13**, 777, 1996.
15. Described on the U.S. Geological Survey's Global Land Information Service (GLIS) website. See: http://edcwww.cr.usgs.gov/Webglis/glisbin/guide.pl/glis/hyper/guide/usgs_doq
16. R. A. Schowengerdt, *Remote Sensing*, 2nd ed., Academic, San Diego, 1997.
17. G.X. Ritter, and J.N. Wilson, *Handbook of Computer Vision Algorithms in Image Algebra*, CRC, Boca Raton, 1996
18. M. Mitchell, *An Introduction to Genetic Algorithms*, MIT, Cambridge, 1996.
19. R. Porter, K. McCabe, and N. Bergmann, "An Applications Approach to Evolvable Hardware," to appear in *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware*, held at the Jet Propulsion Laboratory, Pasadena, 1999.
20. J. J. Szymanski, and S. Alferink, personal communication, 1998.