Program Evolution for Data Mining

Astro Teller
Carnegie Mellon University

Manuela Veloso
Carnegie Mellon University

Around the world there are innumerable databases of information. The quantity of information available has created a high demand for automatic methods for searching these databases and extracting specific kinds of information. Unfortunately, the information in these databases increasingly contains signals that have no corresponding classification symbols. Examples include databases of images, sounds, etc. A few systems have been written to help solve these search and retrieve issues. But we can not write a new system for every kind of signal we want to recognize and extract. Some work has been done on automating (i.e. learning) the task of identifying desired signal elements. It would be useful to automate (learn) not just a part of the classification function, but the entire signal identification program. It would be helpful if we could use the same learning architecture to automatically create these programs for distinguishing many different classes of the same signal type. It would be better still if we could use the same learning architecture to create these programs even for signal types as different as images and sound waves. We introduce PADO (Parallel Architecture Discovery and Orchestration), a learning architecture designed to deliver this. PADO has at its core a variant of genetic programming (GP) that extends the paradigm to explore the space of algorithms. PADO learns the entire classification algorithm for an arbitrary signal type with arbitrary signal class distinctions. This architecture has been designed specifically for signal understanding and classification. The architecture of PADO and its achievements on the recovery of visual and acoustic signal classes from test databases are the subjects of this article.

Keywords:
Machine Learning, Signal Understanding, Data Mining,
Genetic Programming, Algorithm Evolution

1. Introduction

Searching through large sets of data for small subsets that are of particular interest is a cornerstone of the information age. Until recently, most databases contained volumes of information on a scale that allowed each item to be labeled with attributes of likely interest for potential searchers. In the last few years two changes have made this less feasible. The first is the continuing increase in bandwidth and storage space. These past few years have seen humans reach the limits of their ability to keep track of what "cyberspace" contains. These limits will surely be passed in the very near future. The second recent change is the increasing number of databases full of non-text, non-numeric information. There are now a plethora of places that hold photos, still and moving video, noises, sonar, etc. by the hundreds of thousands of elements.

If we want to be able to find specific information in these large repositories of "unlabeled" information, we must either label it automatically (e.g., by applying machine learning), or design methods for automatic recovery of particular items of interest. This "label" problem is the source for a variety of current research. These databases are "unlabeled" in the sense that there are an almost infinite number of possible labels for a signal database (e.g., "pictures with cars in them," "melodic sounds," "dark videos," "pithy paragraphs," "outdoor image scenes," etc.).

The goal of the work described in this article is the automatic recovery of signal classes from a signal database. Clearly, such an automatic system should take as input *what* the user is looking for (e.g., "I want pictures that look like these ..."), not *how* to find it (e.g., "Here is a set of invariant image transformations that all pictures of interest to me preserve..."). The user should be able to use the same system for data retrieval tasks on many different kinds of signals (e.g., image, sound, text, etc). Such a system should be able to deal with any signal type of any size. A system that asks for a preprocessing algorithm to make the signals "manageable" or "understandable" is not a generally useful system. We believe that machine learning will be a crucial tool in the creation of any such general, flexible system.

PADO, Parallel Algorithm Discovery and Orchestration, is a new learning architecture specifically designed for signal understanding. Combining a powerful extension to the GP paradigm and a general program orchestration procedure, PADO can learn a complete classification system, independent of the target signal type, size, or complexity. Given a few labeled signal examples from a class, it can learn to distinguish and classify signals. PADO is custom-made to be the system a user searching in a signal database would want.

Through the course of this article, PADO's method and structure will be explained. Test experiments were performed and some results are reported here to show PADO's promise in the area of signal understanding for database retrieval. We believe that the automatic understanding of signals is one of the highest current hurdles for the information sector. PADO is a first successful step towards clearing this hurdle.

2. Background

Databases of acoustic and visual signals are becoming increasingly plentiful around the world. However, huge databases without powerful search mechanisms are almost useless (Yu and Weiyi, 1994). Realizing this, some of these signal databases continue to try to label their signals as they are put into the databases. That way, standard database search technologies can be applied to these keys (Brice and W., 1990; Krishnamurthy and Imielinski, 1991). For example there is an available database of 250,000 professional photographs (Hilts, 1994). Each entry is labeled with enough information to find images of interest along *certain* dimensions (e.g., "subject","date", etc.). 5000 more images are labeled and added each week. But this is a titanic effort. And what if you were looking for photos with old cars in them? That information is probably not available. No one will every have the time to go back and re-label all 250,000 images. Our approach would only require that the user show PADO a few positive examples (e.g., images with old cars) and a few negative examples (e.g., images without old cars) and then PADO does the rest, learning the concept and retrieving other signals of interest.

The signals in such databases exist in many forms. There is no one standard for how to encode the signals, let alone possible signal labels (Shetler, 1990). There are now immense on-line databases of multi-media information (e.g., (Birmingham et al., 1994)). As another example, the world of video dating is now online and searchable (Wexler, 1992). How can someone find people with certain facial or vocal characteristics? Programs that "recognize" beauty are not on the horizon, but programs that recognize long hair or deep voices are just around the corner. Text is certainly a signal type where this problem of unstructured information is becoming epidemic. A classic example is the large number of news articles that appear every day and must be labeled so that people can find what they are interested in. Thinking Machines' attempt to tackle this text version of the signal classification problem is an example of a learned voting scheme among a number of complex "feature" extractors (Masand, 1994). These features were designed by humans. Learning text understanding agents for taming the WEB is also being pursued (Armstrong et al., 1995). In the long run, human written features will fall hopelessly behind the number, variety, and complexity of the signal databases and the interests of the browsers.

In the recent past there have been some attempts at automating the recognition and classification process. This automation has taken the form of machine learning (e.g., (Fayyad et al., 1993; Rao et al., 1993; Pazzani et al., 1994)) and statistical techniques (e.g., (Hunter, 1990; Agarwal et al., 1992)). The goal is typically to acquire knowledge from large databases. The Skicat work is one of the few examples of machine learning applied to full resolution photo images (Fayyad et al., 1993). We know, however, of no published research that has accomplished the examination and retrieval of **arbitrary** database elements as large and unprocessed as real image and acoustic data. Any method for data mining has, at its heart, a function or algorithm for determining whether a particular database element has the right "features" that make it a candidate for retrieval. The intersection of machine learning and data mining is the learning of some (or all) of that classification function or algorithm. This determination is much harder for machine learning or statistical techniques when the signals are extremely large or unprocessed.

Because part of the PADO system is a type of genetic programming, there should be some mention here of the sort of related work that has been done within the classification paradigm. As far as we know there are no published results of the sort discussed in this article: that is, none that apply genetic programming or genetic algorithms directly to arbitrary signal types and do signal understanding on the basis of that input. For example, there has been GP vision work that seems to fall into two major categories: bitmap recognition and learned aids for vision problems (including object recognition). There have been some examples of genetic programming applied to bitmaps (usually font bitmaps) in order to do classification (e.g., (Andre, 1994; Koza, 1994)). In between, there are interesting works like (Johnson, 1994) that applied GP to a restricted subset of black and white silhouettes of a person and tried to learn where one of the hands was. Learned aids to object recognition can be seen in works like (Tackett, 1993) and (Nguyen and Huang, 1994). For example, in (Tackett, 1993), GP is used to improve the performance of an army system for locating tanks by learning to choose from among existing system components.

A final note on the traditional approach to automatic techniques for data mining is warranted. PADO is a supervised learning algorithm. It needs a few labeled examples to learn from. Unsupervised learning might be possible for some applications of data mining, but would essentially just cluster the elements into a number of groups. This is clearly useless for database elements as complex as image and acoustic signals. Given 100,000 photographs, how might one cluster them? By subject? By distance to focus? Into "Human Subject" and "Nature Subject" categories? These high level concepts, the ones database users are most likely to have interest in, are learnable with examples, but not without.

3. A Method to the Madness

The purpose of this section is not to convince, but to explain. The following section will give experimental results and make a case that the PADO system has a promising future in this area of automated signal understanding. We will first explain the general mechanism of genetic programming. Then PADO's expansion of this idea will be detailed. Finally, the section will conclude with details about exactly what sort of things PADO is evolving and what access these programs have to the signals that will be the experimental subjects in Section 4.

3.1. Genetic Programming at a Glance

Evolutionary computation is biologically motivated. In nature, we see that the combination of survival of the fittest, fitness proportionate reproduction, and genetic recombination is an extremely powerful tool for finding solutions to biological problems. Assuming a basic knowledge of the nature of such genetic evolutionary processes, this subsection gives an outline of the Genetic Programming (GP) mechanism.

Genetic Programming is a strategy for evolving functions that perform well on assigned tasks (Koza, 1992). The process of finding a GP function that is a good or perfect approximation to the target function (in the classic machine learning sense) can be summarized as follows. The population is initialized with a set of randomly generated functions. Each member of the current pool of functions is tested on the task (e.g., curve fitting) to determine its approximate error relative to the target function. It is important to understand that it is not necessary to "know" the target function in order to measure the error for each GP function (e.g., curve fitting to a set of points). A new pool is created in which the functions with lower error have higher representation. The new pool is then subjected to various kinds of genetic recombination. Two popular varieties of recombination are mutation and crossover. Over time the most successful functions in the population become increasingly accurate approximations to the correct solution to the task (i.e., the target function). This process flow is shown in Figure 1.

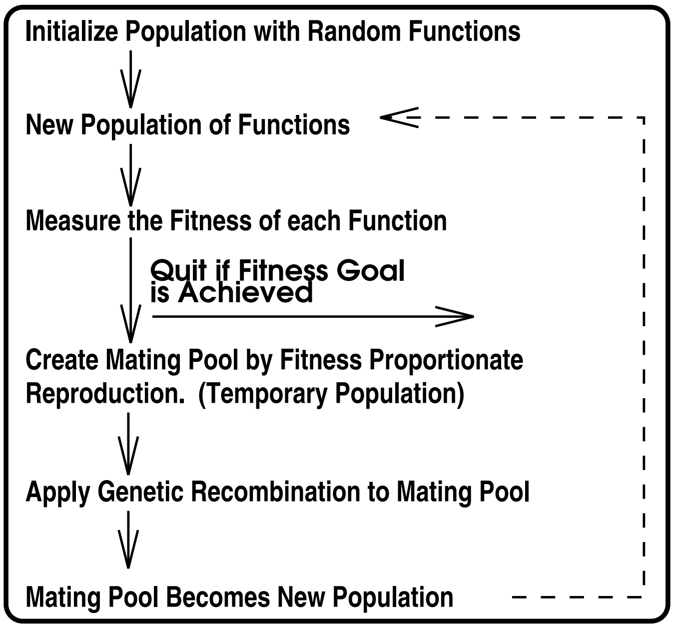

Figure 1: The typical Genetic Programming Cycle.

Evolved functions are represented in GP as Lisp-like expressions consisting of non-terminals (atomic functions) and terminals (e.g., variables and constants). Simple GP functions might look like these two examples:

- (* 2 (IF-THEN-ELSE (= 4 x) (/ x (- 4 x)) (cos (exp (+ x (* (+ 5 6) x))))))
- (exp (/ 9 x) (- (+ (+ 9 8) x) 12))

Given a sufficiently expressive set of mathematical, decision, and action non-terminals along with appropriate variables and constants, this type of "language" can represent many desired functions. GP has been successfully applied to curve fitting, simple classification, reactive agent decision making, control systems problems, and a host of other tasks.

A small percentage of the population is chosen at random from the new population and some non-terminal or terminal part of the each function in this small group is changed. This is called *mutation*. If the first function above had its third "x" changed to a "1", it would be removed from the population and the following function would be added.

- (* 2 (IF-THEN-ELSE (= 4 x) (/ x (- 4 **1**)) (cos (exp (+ x (* (+ 5 6) x))))))

In *crossover*, two functions are chosen at random from the new population and they exchange randomly selected sub-expressions. This process is often described in the GP literature as "exchanging random sub-trees" in which the sub-trees refer to the isomorphic representation of parenthesized expressions as functional trees. Crossover is done to a large percent of the new population (often 80% to 90%). The first two examples shown above might look like this after undergoing crossover.

- (* 2 (IF-THEN-ELSE (= 4 x) (/ x (- 4 x)) **(- (+ (+ 9 8) x) 12)**))
- (exp (/ 9 x) **(cos (exp (+ x (* (+ 5 6) x))))**)

Genetic Programming, as a sub-set of evolutionary computation, is a form of best-first search (Tackett, 1994; Tackett, 1995). Exponentially increasing representation is given to those functions that have highest fitness and so those points in the space are exponentially more likely to be examined next, relative to the other points under consideration (i.e. the other functions in the group).

In the vocabulary of GP, a pool of functions is called a **population**. To distinguish one population from another they are referred to as **generations**. The initial population is traditionally called "Generation 0" and each successive generation is numbered in increasing integer order.

In PADO, the things that are evolved are **programs**, not functions. This is because functions are not expressive enough to capture high level features of arbitrary sized signals in a tractable space. For a discussion on the distinction between functions and algorithms (programs) see (Teller and Veloso, 1995c) or (Teller, 1994b). The form that PADO programs take is represented in a specific language described below. The language of PADO differs from traditional GP because the Lisp-like functional language of GP, even when it is augmented to allow the evolution of algorithms, is poorly equipped for the task (Teller and Veloso, 1995b). (See Appendix A and B for example PADO programs.)

The similarity between syntax and functionality breaks down when *programs* (PADO) rather than *functions* (GP) are evolved (Teller, 1994b). This means that the PADO representation (section

3.3) is critical to PADO's success. Though more subtle and less PADO-specific, it should be remembered that the representation of the signals is also a factor in the learnability of particular concepts within a particular signal space.

3.2. PADO Learning

Part of the PADO architecture falls under the general heading of evolutionary computation. This subsection will discuss the way PADO works and how its central component is an instance of evolutionary computation. Because the PADO architecture was designed to apply to any signal type, that is how it will be introduced.

The goal of the PADO architecture is to learn to take signals as input and output correct labels. **When there are $\mathcal{C}$ classes to choose from, PADO starts by learning $\mathcal{C}$ different systems**. System$_I$ is responsible for taking a signal as input and returning a confidence that class I is the correct label. In the real world, none of the $\mathcal{C}$ systems will work perfectly; even humans make discrimination errors. Two questions of the PADO architecture recur: "How does PADO **learn** good individual, single-class discrimination systems?" and "How does PADO **orchestrate** them for maximum effect?"

Each individual System$_I$ is built out of several programs. Each of these programs does exactly what the system as a whole does: it takes a signal as input and returns a confidence value that label I is the correct label. The reason for this seeming redundancy can be seen in (Teller and Veloso, 1995c). The following is a description of a simple orchestration scheme that was used in the experiments described later in the article. System$_I$ is built from the $\mathcal{S}$ programs that best (based on the training results) learned to recognize the features of a signal that define it as an instance of class I. The $\mathcal{S}$ responses that the $\mathcal{S}$ programs return on seeing a particular signal are all weighted and their weighted average of responses is interpreted as the confidence that System$_I$ has as to whether the signal in question contains an object from class I. That is the low level orchestration. PADO does object recognition by orchestrating the responses of the $\mathcal{C}$ single class discrimination systems . The confidence response of each system is initially weighted equally. Then, on a particular test case a function F (e.g. MAX) takes the weighted confidences from each System$_I$ and selects a J $(1 \leq J \leq \mathcal{C})$ as the image object class. That is the high level orchestration. Figure 2 pictures this bi-level orchestration learning process.

In the experiments we followed an orchestration method in which the orchestration phase occured early in the test phase. During the first few test images the weights are adjusted by telling PADO after its guess whether it was right or wrong. Then, the rest of the test examples were run to compile results. Specifically, each Program$_J$ in a particular System$_I$ has its weight V_J adjusted after each of these initial tests so that its weight increases when it returns a confidence near the correct confidence and decreases if its returned confidence is far from the correct confidence. Similarly,

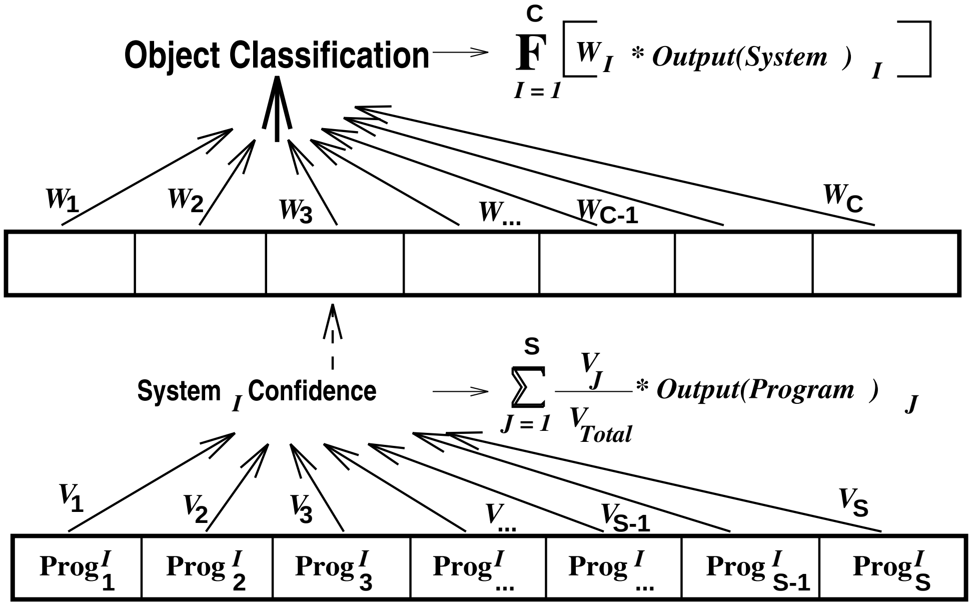

Figure 2: The weights **W** and **V** that are trained in the orchestration phase

System$_I$ has its weight W_I adjusted in the same manner. This strategy (as shown in Figure 2) is only one of many ways that the orchestration could be accomplished. A few other orchestration strategies have also been tried with similar success.

PADO evolves its programs in a PADO-specific graph structured language. At the beginning of a learning session, the main population is filled with $\mathcal{P}$ programs that have been randomly generated using a grammar for the legal syntax of the language. All programs in this language are constrained by the syntax to return a number that is interpreted as a confidence value between some minimum confidence and some maximum confidence.

During each generation, each program is presented with a few training examples from each signal class to be classified. A program's response to a signal is a confidence value. If a program is a recognizer of signal class I, then it will receive positive reward for returning high confidence values when it sees a signal from class I and low confidence values otherwise. At the end of the generation each program is placed into one of C sub-pools (recall that C is the number of classes to classify). A particular program is placed in sub-pool J if its fitness can be maximized by treating it as a recognizer of signal class J. This placement is subject to the constraint that each of the C pools must contain P/C programs. It is from the best programs in sub-pool I that System$_I$ is created.

Each sub-pool is then sorted by increasing fitness and each program is ranked accordingly. A new total population is created by putting a copy of Program$_I$ in the new population with probability $2 * rank(I)/(P/C)$. The expected number of copies of the best program in sub-pool I is 2, the expected number of copies of the median program is 1, and the expected number of copies of the worst program in sub-pool I is $2/(P/C)$. This is fitness *rank* proportionate reproduction.

Finally, 85% of the programs within each sub-pool are subjected to crossover and another 5%

are subjected to mutation. All crossovers take place between two programs in the **same** sub-pool. That means they are both recognizers of the same class. Crossover and mutation in PADO are more complicated than their standard forms in genetic algorithms or genetic programming. Both the crossover and mutation operators are "SMART" operators that are co-evolved with the main population. Further details on how these operators work is available in (Teller, 1995a).

3.3. PADO Representation

Figure 3 sketches the structure of a PADO program. Each program is constructed as an arbitrary directed graph of nodes. As an arbitrary directed graph of N nodes, each node can have as many as N *arcs* outgoing. These arcs indicate possible flows of control in the program. In a PADO program each node has two parts: an *action* and a *branch-decision*. Each program has a private stack and an indexed memory. All *actions* pop their inputs from this stack and push their result back onto the stack. These actions are the equivalent of GP's terminals and non-terminals. The indexed memory is effected in the usual way via READ and WRITE actions (Teller, 1994a).

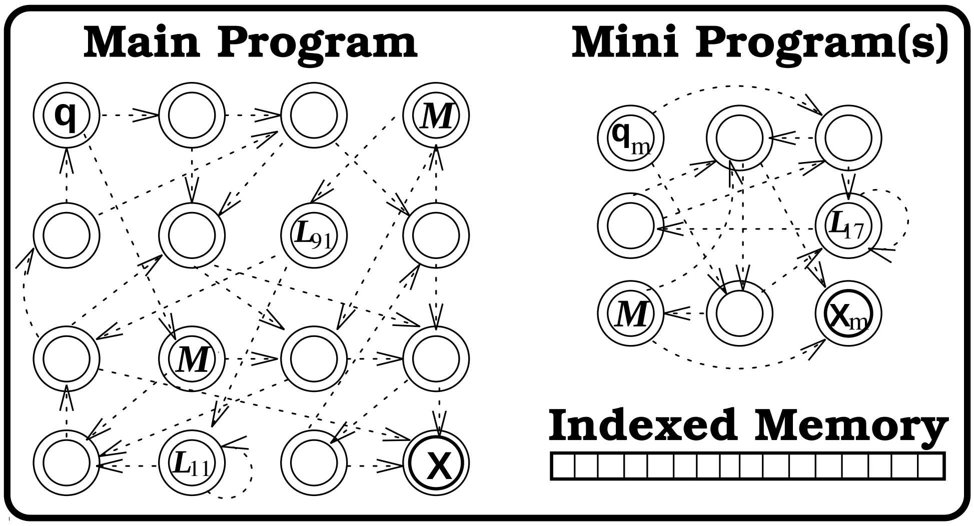

Figure 3: This is the basic structure of a PADO program. There can be one or more Mini programs for each PADO program. Each Mini program may be referenced from the Main program, another local Mini program, or a *Library* program.

After the action at node i is executed, an arc will be taken to a new node. The branch-decision function at the current node will make this decision. Each node has its own branch-decision function that may use the top of the stack, the previous state number, the memory, and constants to pick an arc.

There are several special nodes shown in Figure 3. Node q is the start node. It is special in no other way than it is always the first node to be executed when a program begins. Node X is the stop node. When this node is reached, its action is executed and then the program halts. When a program halts, its response is considered to be the current value residing in some particular memory location (e.g., response = Memory[0]). If a program halts sooner than a pre-set time threshold, it is started again at its start node (without erasing its memory or stack) to give it a chance to revise its confidence value. A weighted average of the responses the program gives on a particular execution is computed and interpreted as the answer (confidence). The weight of a response at time t_{i} is i. Later responses count more towards the total response of the program. Because of the time threshold, PADO's programs are guaranteed to *halt* and respond in a fixed amount of time.

Node M executes the private *Mini* program as its action. It then executes its branch-decision function as normal. The *Mini* program associated with each *Main* program bears similarity to the concept of ADF's (automatically defined functions) (Koza, 1994). It may be called at any point in the *Main* program and it evolves along with the *Main* program. *Mini* programs are in every way normal PADO programs. Their size is not constrained to be smaller than the *Main* programs. The name *Mini* denotes only that it is owned by the *Main* program. The *Mini* programs may recursively call themselves or the globally available *Library* programs, just like a *Main* program may.

The *Library* programs (e.g., L_{91} in Figure 3) are globally available programs that can be executed at any time and from anywhere just like the Mini programs. But unlike the Mini programs, where each Mini may be run only during the execution of the PADO program of which it is a part, the *Library* programs are available to the entire population. The method by which these *Library* programs change can be seen in some detail (Teller and Veloso, 1995c). See (Angeline and Pollack, 1993) for a similar concept.

This is certainly not the only possible structure for evolving programs. The details of how PADO arrived at this particular arrangement of syntax, Mini programs, Libraries, and SMART genetic operators are too lengthy to include here. A relevant note is that this structure was arrived at partly through intuition and has been verified largely through empirical results. In retrospect there are probable reasons for the success of this arrangement, but the justification for the PADO architecture is currently more empirical than theoretical.

Here is a brief summary of the language *actions* and their effects:

Algebraic Primitives: ADD, SUB, MULT, DIV, NOT, MAX, MIN

These functions allow basic manipulation of the integers. All values are constrained to the range 0 to 255. For example, DIV(X,0) results in 255 and NOT(X) maps $\{1..255\}$ to 0 and $\{0\}$ to 1.

Memory Primitives: READ, WRITE

These two functions access the memory of the program. Each program has a memory which is organized as an array of 256 integers that can take on values between 0 and 255. READ(X) returns the integer stored in position X of the memory array. WRITE(X,Y) takes the value X and writes it into position Y of the indexed memory. WRITE returns the OLD value of position Y (i.e. a WRITE is a READ with a side-effect). The memory is cleared (all positions set to zero) at the beginning of a program execution.

Branching Primitives: IF-THEN-ELSE, EITHER

In both cases the primitive pops X,Y, and Z off the stack and then replaces either Y or Z (not both) depending on the value of X. For IF-THEN-ELSE the test is (X greater than 0). For EITHER the test is (X less than RandomNumber) where RandomNumber varies between 0 and 255. These primitives can be used as an action or a branch-decision function. In the former case, they have no effect on the flow of control.

Signal Primitives: POINT, LEAST, MOST, AVERAGE, VARIANCE, DIFFERENCE

These are the language functions that can access the signals. In order to demonstrate PADO's power and flexibility, these same primitives were used for both the image and sound data! POINT returns the intensity value at that point in the image or sound. The other five return the respective functions applied to the region in the signal that their four parameters (from the stack) specify. For the image experiments these four number (X1,Y1) and (X2,Y2) are interpreted as the upper left and lower right corners of a rectangle in the image. For the sound experiments a start and stop position are indicated as the section. These positions are specified by 256*X1 + Y2 and 256*X2 + Y2. For both cases, if the points specify a negative area then the opposite interpretation is taken and the function is applied to the positive area region. DIFFERENCE is the difference between the average values on two halves of a line segment. This is the diagonal of the rectangle for the image experiments.

Routine Primitives: MINI, LIBRARY[i]

These are programs that can be called from the Main program. In addition, they may be called from each other. Because they are programs, and not simple functions, the effect they will have on the stack and memory before completion (if they ever stop) is unknown.

- **MINI :** Each MINI program is private to the MAIN program that uses it and can be called as many times as desired from this MAIN program. Because each MINI is an arbitrarily complex program, it has an arbitrary number of "parameters" which it can pull of the stack. For this paper, each MAIN program was given exactly one MINI. In general, a MAIN program could have many MINI programs for its private use.
- **(LIBRARY[i] X Y U V ...) :** There are 150 library programs. The **i** is not really a parameter. Instead an *Action* calling a Library program from some program's MAIN, MINI, or from another Library program might look like **Library57**. Like the MINI programs, the Library programs "take" an unknown number of parameters by popping the parameters they "expect" off the parameter stack. All 150 Library programs are available to all programs in the population.

4. Experimental Results

In order to show PADO's ability to successfully extract the right signals from a signal database, we must first produce such a database to be searched. For the purposes of this article, two databases were generated with certain properties in mind. Both of the database are made of signals from the real world (i.e., large, unprocessed data from a microphone and a video camera). In order to run a large number of trials, these experimental databases needed to be reasonably small. For experimental purposes it served to pick a few signal classes that are easy for people to distinguish but do not have simple properties through which they could be classified. One test of the non-existence of these overly simple classification concepts was that all of the individual primitives were applied to signals of each signal type. For example, we applied a global average test (e.g., Average(0,0,255,255) on signal J) to all the signals in both databases. The resulting values did not give enough information to distinguish the signals.

The first database contains 525 three second sound samples. These sounds were taken from the SPIB ftp site in Rice University (anonymous ftp to spib.rice.edu). This database has an appealing seven way clustering (75 from each class): *the sound of a Buccanneer Jet Engine*, *the sound of a Machine gun*, *the sound of an M109 Tank Engine*, *the sound on the floor of a car factory*, *the sound in a car production hall*, *the sound of a Volvo engine*, and *the sound of babble in an army mess hall*. There are many possible ways of subdividing this sound database; the classes chosen for these experiments are typical of the sort of distinctions that might interest a database searcher.

The second database, a set of 700 video images, was created by Sebastian Thrun for use in his robotics experiments. The lighting, position and rotation of the objects in the image signal varies widely. The floor and wall behind and underneath the objects are constant. Nothing else except the object is in the image. However, the distance from the object to the camera ranges from 1.5 to 4 feet and there is often severe foreshortening of the objects in the image. This database also has an appealing seven way clustering (100 from each class): Book, Bottle, Cap, Coke Can, Glasses, Hammer, and Shoe. This particular set of classes was used in these experiments because they are easily distinguished by people and they are disjoint sets. This database, like any database of complex signals, has an almost infinite number of class clusterings (e.g., "Objects farther than 3.5 feet from the camera", "Non-rigid objects", etc.) PADO can learn to classify signals from **any** of these class, not just the ones shown in this article. See Figure 4 for sample images.

For each of the following experiments, the database was broken into two halves. The first half was used to train PADO and the second half was used as a test set. From this test set, seven examples from each class were used to learn the orchestration. The rest of the test set was used to actually test PADO and produce the results shown below. Each point of each curve in Figures 5 and 6 is an average over 5 runs.

For identifying elements of a class within elements from many classes, *accuracy* and *coverage*

Sample Train Images	Sample Test Images

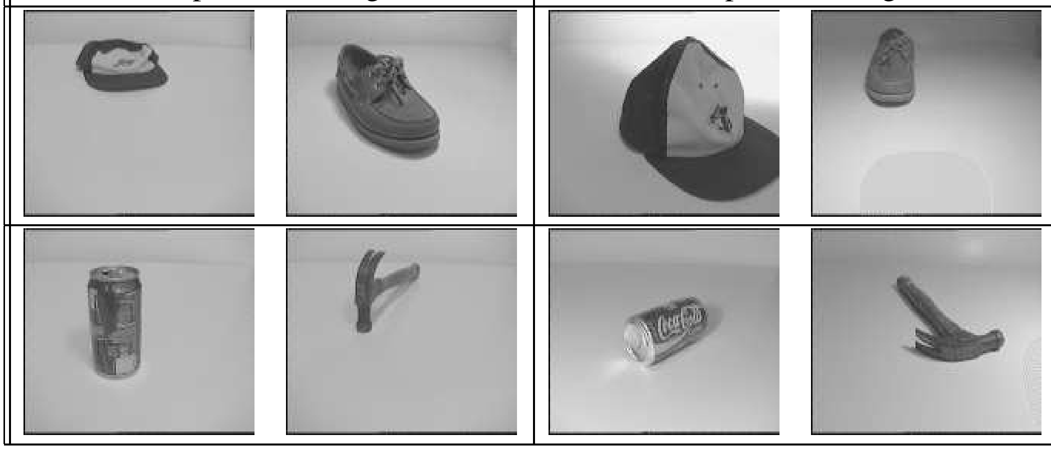

Figure 4: 4 randomly chosen images from the experimental train and test image databases.

are traditional measures of performance. *Accuracy* is the number of true positives divided by the number of false positives plus true positives. *Coverage* is the number of true positives divided by the total number of positive examples seen. The fitness used for evolutionary learning (training of the PADO programs) was based upon distance from correct confidence for each training example. So accuracy and coverage were never specifically trained for, but were effectively weighted equally in the fitness measure. In the experiments below, PADO gives exactly one positive response for each test example seen. In other words, "Sound$_I$ is *Machine Gun*" or "Sound$_I$ is *Volvo Engine*", but **not** both. Given this model of one class chosen per image, if PADO just guessed, it could achieve an accuracy of $1/7$ (14.28%) and a coverage of $1/7$ (14.28%).

4.1. Sound

Figure 5 shows PADO's performance on the sound database. In both graphs there is some discrepancy between the PADO's performance for different sound classes. This is natural since some sounds are more unique or constant. Sounds that can easily be confused with other sounds or sounds whose characteristics vary widely between samples, are necessarily harder to pick out with high accuracy and high coverage. It can not be over stressed that these are test sounds that the PADO system has never heard. The training sounds were similar, but not the same.

As an example, let's look at PADO's performance in Figure 5 with respect to the sound of the M109 tank engine. The accuracy graph shows that by generation 80 PADO has achieved a database retrieval accuracy of about 70% on M109 tank engine sounds it has never heard before. That means that 70% of the sounds it returns will in fact be sounds of an M109 tank engine. The coverage graph shows that PADO database retrieval coverage is about 60% on the previously unheard M109 tank engine sounds. That means that the pool of sounds PADO retrieved includes 60% of all the

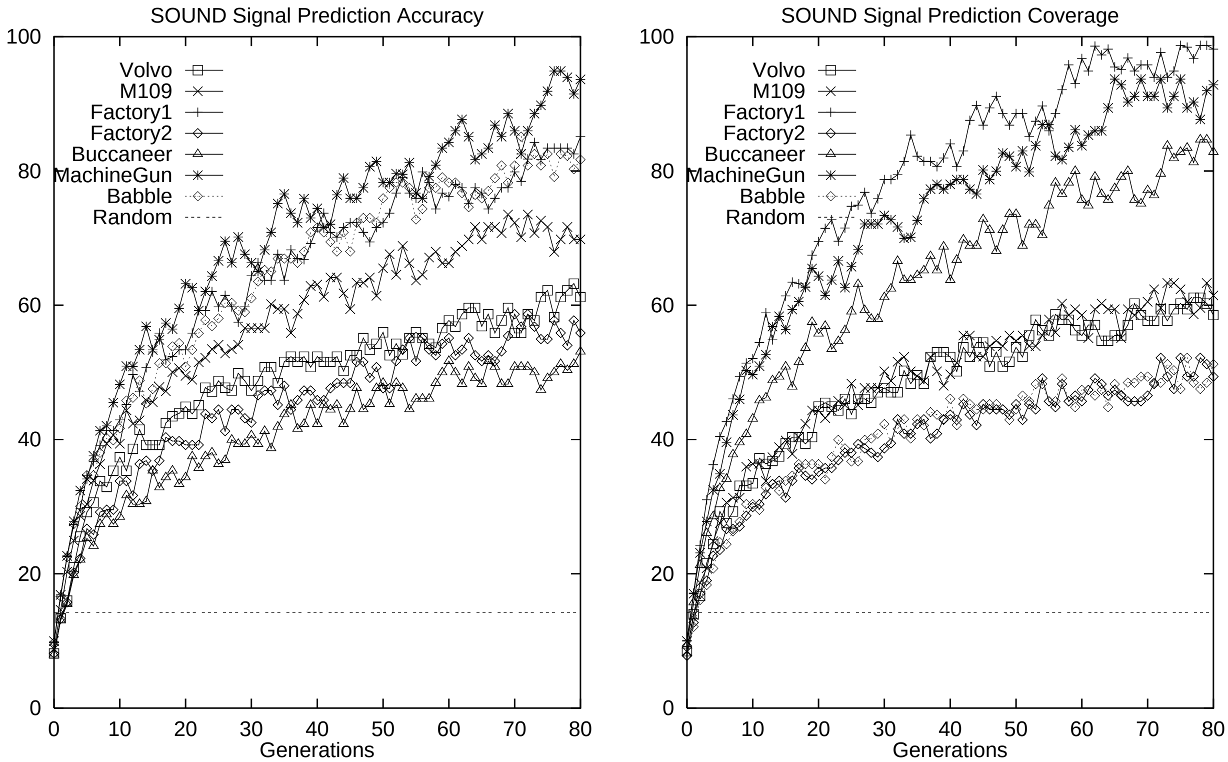

Figure 5: PADO's sound classification **accuracy** and PADO's sound classification **coverage**

tank sounds in the entire database. Both of these numbers, as mentioned before, would be about 14% if PADO had just been guessing.

Looking at these results, it is hard to remember the primitive level of access that PADO was given to these sounds. None of the many well known sound processing algorithms were used on the sounds or given to PADO to help its access of the sound signals. Being able to get no more than the most basic data about a sequence of numbers (POINT, AVERAGE, etc.), any real "features" had to be created as the PADO programs evolved. Our current work includes giving PADO signal primitives that are both more signal-specific and more powerful. Early results indicate that further PADO performance improvements are possible when PADO has to invent less for itself.

As was mentioned earlier in this article, the representation of the signals can also make a problem harder or easier. In this case, the representation for the sounds given to PADO was purposefully made **more** difficult to "understand" in order to test PADO's ability on confusing representations. The sounds are sound waves with 8 bits per sample and 19,800 samples per second of sound. So each 3 second sound sample is, therefore, about 60,000 numbers that are usually represented as numbers between -128 and +127. The sounds PADO was given access to were the same 60,000 numbers but now in the range $\{0..255\}$. So the "0 line" around which sounds are partly symmetric has been moved to 128. No information is lost in this transformation, but the important sound features may have become a little more difficult to extract.

4.2. Vision

Figure 6 shows PADO's performance on the image database. In both graphs there is some discrepancy between the PADO's performance for different image classes. As with the sound results, this is natural since some images are more unique or constant. Images that can easily be confused with other images or image classes whose characteristics varies widely between image samples are necessarily harder to pick out with high accuracy and high coverage. Again, it can not be over stressed that these are images that PADO has never seen. The training images were similar, but not the same. Figure 4 shows just how different the training and testing images are.

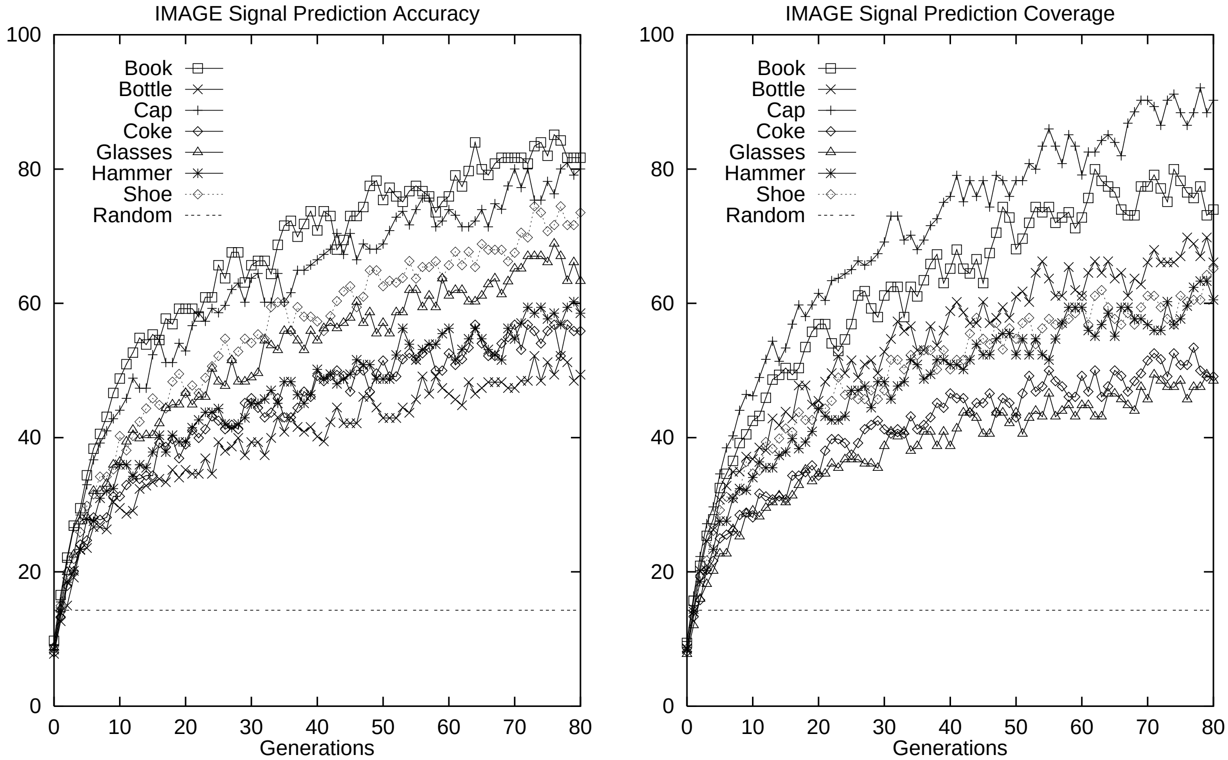

Figure 6: PADO's image classification **accuracy** and PADO's image classification **coverage**

PADO's performance data mining for images (e.g., images of shoes) is shown in Figure 6. The image accuracy graph shows that by generation 80 PADO has achieved a 75% database accuracy retrieving images of shoes it has never seen before. That means that 3 out of 4 images it returns will in fact be images of shoes. The image coverage graph shows that PADO's database coverage retrieval of previously unseen shoe images is about 65%. That means that the pool of images PADO retrieved includes 65% of all the shoe images in the entire database. Again, both of these numbers would be about 14% if PADO had been guessing.

There are innumerable image-specific algorithms for processing images to bring out particular image features. These range from edge segmentation, to vanishing point analysis, to texture region grouping. None of these image-specific algorithms were given to PADO to use in classifying the images (though PADO would certainly have done better with access to these more “helpful” algorithms). The point of this article is that PADO can automatically produce a usable system for signal database retrieval with **no** help from the user other than a few labeled examples. These results for image signal understand were accomplished with the same basic signal access primitives used for sound signals: POINT, AVERAGE, MOST, LEAST, VARIANCE, and DIFFERENCE.

These images are, by some metrics, not easy to distinguish (see (Teller and Veloso, 1995c) for further analysis). The apparent complexity of these image classes makes it seem likely that PADO is developing algorithms for focusing attention and doing more “intelligent” or complex processing of the image. Specialized, signal processing algorithms are needed in order to solve this signal understanding problem. But we do not have to figure out which algorithms are needed or give them to PADO. We only need to give PADO the most basic signal inspection primitives (e.g., AVERAGE) and PADO can learn the more complex algorithms as needed!

4.3. Discussion

An interesting feature of Figures 5 and 6 is that the classes on which PADO achieves higher accuracy results are not always the same classes on which PADO achieves higher coverage results. In other words, for some classes PADO seems to be more “picky”, choosing accuracy over coverage, and for some classes, just the opposite is true. To some extent this is a “feature” of the signal type and signal classes. Why PADO acts this way is also a subject of current research.

This article detailed two particular experiments using a sound and a vision database. PADO has, however, been applied to several other domains. For example, PADO has been given databases of images containing objects held by people (Teller and Veloso, 1995c), head and shoulder shots of people in natural cluttered backgrounds (Teller, 1995b), and even manufactured shapes with particularly interesting signal properties (Teller and Veloso, 1995a). The most exciting aspect of these results (all as positive as the results reported in Figures 5 and 6) is that they were all achieved using the same primitives (AVERAGE, LEAST, MOST, VARIANCE, DIFFERENCE) presented in Section 3 of this article for both the sound and image databases!

Suppose that we had a database of 7000 sounds from seven classes. The accuracy and coverage graphs in Figures 5 and 6 tell us that **at worst**, a search for elements of a particular class would yield about 1000 returned database entries, about 500 of which were from that particular class. In this case human post-retrieval filtering would be required, but PADO would have dropped the human’s work load by a factor of seven. In the **best case**, PADO could be expected to return about 1000 database entries, over 900 of which were instances of the desired class. In such a case PADO has made a very laborious task almost effortless.

Consider the familiar process of requesting a list of articles from an online library card catalog.

We spend a minute describing what we are looking for (e.g., "Teller/author and (genetic? or learn?)/subject"). Then we activate the search and wait. If the database is large (e.g., INSPEC) it can take a few minutes. What we get back is often a long list that we have to page through to find articles worth reading. The length and interest density of this list depends on how we specify the search, but generally we do not know exactly what we are looking for and so this human post processing is an inevitable part of the process. The insight we can draw from this familiar search procedure is that "perfect" recovery is probably not as important as the ease with which the search can be specified and time the search takes to complete. The search specification is a relatively unexplored aspect of PADO. The kind of results described in this article would require a small number of examples from each class to be distinguished. The experiments reported in this article used 14 examples from each class to train a generation. For example, if only one class were of interest, about 14 positive and 14 negative examples would suffice in order for PADO to learn to find more positive instances in a database with the accuracy and coverage seen in Figures 5 and 6.

PADO's learning and searching speeds will now be addressed. It took about 2 days on a Dec-Alpha to complete one run for the experiments above. Some of this time was taken running the actual tests, but the learning for 80 generations took at least 36 of the 48 hours. However, during this 48 hour period, seven problems were solved and tested, not just one. A typical user would probably be looking for one specific signal type in a particular search. So PADO need only label each database element as **YES** or **NO** rather than picking one of many classes for each element.

PADO, in its current incarnation, is still a research tool. However, it has been designed so that its learning and testing speeds can be significantly increased. Because both the learning and generalization performance of PADO require many independent evolutionarily "discovered" algorithms to be run, linear speed-ups can easily be achieved using parallel processors. In addition, PADO's learned programs are currently interpreted. These interpreted programs could be compiled for another considerable increase in speed.

For the learning phase, if 24 processors were available, the learning for all seven classes (all seven experiments) would have taken two hours, not two days. For the learning phase compilation of the programs under evolution would speed up the system, but would be partially offset by the time required to do a few thousand compilations each generation. Our current estimate for the training phase speed up due to compilation learning time is a factor 2. For the database retrieval phase PADO can currently look through a database of 100,000 signals in a day. With the same 24 processors at PADO's disposal, this retrieval would take an hour, not a day. Because the same few dozen programs will be used repeatedly during the database retrieval phase, compilation now becomes a larger benefit. On a single processor, if PADO's programs were compiled, the database retrieval would probably take 3 or 4 hours. Compiled PADO programs running on 24 processors might take as little as 5 to 10 minutes for the entire 100,000 signal mining expedition.

5. Future Work

The issue of scalability is critical to the success of PADO. We are trying to design a learning architecture that can build up useful systems to be applied to databases of real signals. Because PADO was designed with an eye for practical application as well as scientific advancement, it must be able to scale up to harder problems.

The three crucial areas of improvement for PADO are speed, accuracy/coverage, and the requirement of even fewer training examples. Speed, as has already been discussed, can and is being improved in a variety of ways. Using parallel processors and compiling PADO programs are both part of our near future research plan. In addition, there are a variety of environment and representation issues that are being investigated for further speed ups in PADO's learning and performance.

PADO's performance results (accuracy and coverage) are far better than random, but they are not perfect. Many applications can be useful even if there is some error, but many applications require error rates of 5% to 10% and some cannot tolerate even that. Fortunately the retrieval of signals from databases is not an all or nothing enterprise (just like the online card catalog search mentioned above) and even with the error rates shown in this article, PADO can be of significant value. Our current work now concentrates on understanding PADO better and using this understanding to improve PADO's future performance.

PADO currently requires few examples relative to most other machine learning techniques. This is due in part to added signal noise at training time (to prevent overfitting) and the exact method that PADO uses to test the fitness of each program against subsets of the training signals. For some signal understanding tasks (e.g., finding guns in the airport x-ray images), providing a large number of labeled examples to PADO would be worth the correspondingly higher accuracy and coverage rates PADO could achieve. For tasks such as data mining, a user should be asked to specify just enough examples that PADO can still perform a useful search. Therefore, finding ways for PADO to extract high level features from even a few images is an avenue of current research.

Interestingly, the number of classes is not really a scalability issue with respect to data mining tasks. For most data mining tasks there will be two or at most a few classes of interest because generally each signal will fall into the "Retrieve" or the "Don't Retrieve" classes. So even in a database of 100,000 signals, PADO will usually be determining which of a very limited number of classes each signal is. In short, the number of possible classes in a signal database is almost infinite, but a particular search will usually involve a YES/NO distinction for each signal.

6. Conclusion

The goal of this article is to have shown the reader that autonomous search in large databases of unlabeled or poorly labeled signals for subjectively specified elements is on the horizon. The motivation for this article is the following sort of scenario: a graphic designer wants to find pictures of “old cars” in a photography databases. It would save her enormous amounts of time if, rather than scrolling through 100,000 images manually, she could show a dozen example pictures of “old cars” to PADO and then allow PADO to find similar pictures. Clearly a system that could do this for a broad range of signal types and signal element specifications would be invaluable.

This article gave a brief description of PADO and the process involved in the creation of a PADO system: the discovery of algorithms that can be executed in parallel and the orchestration of these algorithms into a useful system. The representation of these algorithms, and their “discovery” through an extension of the Genetic Programming paradigm was summarized. PADO, through this process of evolutionary computation discovery and multi-level program orchestration, can take a few examples from a signal class and, through supervised learning, learn to correctly classify new signals as IN or OUT of that class. And most importantly, PADO can do this for any signal type, size or complexity with no human intervention or signal preprocessing.

Two sets of experiments were presented in which PADO was asked to search through a database of large signals and label each element with one of seven classes. Figures 5 and Figures 6 showed the accuracy and coverage achieved by PADO for each class for both sound and image signals. The results show that PADO’s ability to find specific signals of interest, while not perfect, is high enough to have real application value.

The number of applications in need of increased autonomous signal understanding is innumerable. Most of these applications don’t know in advance what form all the signals will take or what aspects of the signals will be of greatest interest. The creation of a system that can meet the needs of these applications is one of the most exciting current demands of the information age.

Acknowledgements: We would like to thank Sebastian Thrun for the permission to use his images.

REFERENCES

Agarwal, R., Ghosh, S., Imielinski, T., Iyer, B., and Swami, A. (1992). An interval classifier for database mining applications. In *Very Large Data Bases*.

Andre, D. (1994). Automatically defined features: The simultaneous evolution of 2-dimensional feature detectors and an algorithm for using them. In Kenneth E. Kinnear, J., editor, *Advances In Genetic Programming*, pages 477–494. MIT Press.

Angeline, P. and Pollack, J. (1993). Evolutionary module acquisition. In Fogel, D., editor, *Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 154–163. Evolutionary Programming Society.

Armstrong, R., Freitag, D., Joachims, T., and Mitchell, T. (1995). Webwatcher: A learning apprentice for the world wide web. In *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*. AAAI Press.

Birmingham, W. P., Drabenstott, K. M., Frost, C. O., Warner, A. J., and Willis, K. (1994). The university of michigan digital library: This is not your father's library. In *Proceedings of Digital Libraries*, pages 53–60.

Brice, R. and W., A. (1990). Finding interesting things in lots of data. In *23rd Hawaii International Conference on System Sciences*. Morgan Kauffman.

Fayyad, U. M., Weir, N., and Djorgovski, S. (1993). Skicat: A machine learning system for automated cataloging of large scale sky surveys. In *Proceedings of the Tenth International Conference on Machine Learning*. Morgan Kauffman.

Hilts, P. (1994). New software lets designers seymour. *Publishers Weekly*, 24:28.

Hunter, L. (1990). Knowledge acquisition planning for inference from large databases. In *23rd Hawaii International Conference on System Sciences*. Morgan Kauffman.

Johnson, M. P. (1994). Evolving visual routines. In Brooks, R. and Maes, P., editors, *Artificial Life IV*, pages 198–209. MIT Press.

Koza, J. (1992). *Genetic Programming*. MIT Press.

Koza, J. (1994). *Genetic Programming II*. MIT Press.

Krishnamurthy, R. and Imielinski, T. (1991). Practioner problems in need of database research: Research direction in knowledge discovery. In *Sigmod Record*.

Masand, B. (1994). Optimizing confidence of text classification by evolution of symbolic expressions. In Kenneth E. Kinnear, J., editor, *Advances In Genetic Programming*, pages 459–476. MIT Press.

Nguyen, T. and Huang, T. (1994). Evolvable 3d modeling for model-based object recognition systems. In Kenneth E. Kinnear, J., editor, *Advances In Genetic Programming*, pages 459–476. MIT Press.

Pazzani, M., Merz, C., Murphy, P., Ali, K., Hume, T., and Brunk, C. (1994). Reducing misclassification costs. In *Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kauffman.

Rao, R. B., Voigt, T. B., and Fermanian, T. W. (1993). Data mining of subjective agricultural data. In *Proceedings of the Tenth International Conference on Machine Learning*. Morgan Kauffman.

Shetler, T. (1990). Birth of the blob. *Byte*, 15:221–226.

Tackett, W. A. (1993). Genetic programming for feature discovery and image discrimination. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kauffman.

Tackett, W. A. (1994). *Recombination, Selection, and the Genetic Construction of Computer Programs.* PhD thesis, University of Southern California. Available as: Technical Report CENG 94-13. Dept. of Electrical Engineering Systems.

Tackett, W. A. (1995). Greedy recombination and genetic search on the space of computer programs. In Whitley, L. and Vose, M., editors, *Proceedings of the Third International Conference on Foundations of Genetic Algorithms*, pages 118–130. Morgan Kauffman.

Teller, A. (1994a). The evolution of mental models. In Kenneth E. Kinnear, J., editor, *Advances In Genetic Programming*, pages 199–220. MIT Press.

Teller, A. (1994b). Genetic programming, indexed memory, the halting problem, and other curiosities. In *Proceedings of the 7th annual FLAIRS*, pages 270–274. IEEE Press.

Teller, A. (1995a). Evolving programmers. In Kinnear, K., editor, *Advances in Genetic Programming II*. Submitted for review.

Teller, A. (1995b). Identifying people based on faces in a cluttered background. In *Currently Unpublished. Available on request*.

Teller, A. and Veloso, M. (1995a). A controlled experiment: Evolution for learning difficult image classification. In *Proceedings of the 7th Portuguese Conference on Artificial Intelligence*. Submitted for review.

Teller, A. and Veloso, M. (1995b). Language representation progression in PADO. In *AAAI Fall Symposium Series. AAAI Technical Report*. AAAI Press.

Teller, A. and Veloso, M. (1995c). PADO: A new learning architecture for object recognition. In Ikeuchi, K. and Veloso, M., editors, *Symbolic Visual Learning*. Oxford University Press.

Wexler, J. M. (1992). Great expectations for multimedia courtship. *Computer World*, 26:65.

Yu, C. and Weiyi, M. (1994). Progress in database search strategies. *IEEE Software*, 11:11–19.

This is a legend for both Appendix A and B.

Sign	Meaning	Sign	Meaning	Sign	Meaning
Action	Action at that node	A1	ARC_1	00	Stop Node
Branch	Branch-Decision at that node	A2	ARC_2	01	Start Node
B-C	Branch-Constant				

Appendix A: This was the machine gun recognizing program of generation 47.

MAIN

NODE	ACTION	BRANCH	A1	A2	B-C	NODE	ACTION	BRANCH	A1	A2	B-C
0	MOST	NOT-EQ	89	53	224	1	143	DIV	7	12	0
2	LESS	READ	94	4	-56	3	165	MULT	4	65	-11
4	28	MORE	11	11	-179	5	WRITE	NOT	11	63	0
6	191	SUB	9	10	88	7	111	MIN	94	2	205
8	72	NOT-EQ	8	5	117	9	READ	EQ	42	10	0
10	WRITE	NOT	3	22	0	11	VARIANCE	NOT	6	7	0
12	214	DIV	7	22	243	13	56	LESS	10	74	89
14	MOST	MIN	14	16	-130	15	MULT	EQ	20	34	-132
16	147	READ	51	39	-176	17	58	MULT	34	61	0
18	POINT	MORE	72	77	230	19	EQ	SUB	21	76	189
20	78	MIN	19	14	0	21	WRITE	MIN	32	89	5
22	169	LESS	51	18	-240	23	140	EQ	41	56	205
24	WRITE	ADD	16	81	-131	25	WRITE	SUB	0	72	0
26	86	SUB	24	50	106	29	232	NOT-EQ	10	62	-96
30	14	EQ	22	87	128	32	246	MORE	43	40	0
33	READ	EQ	69	24	-28	34	232	MAX	54	91	-164
35	LEAST	MAX	86	45	-112	36	WRITE	NOT	88	45	-242
37	240	ADD	45	66	0	38	85	MULT	47	83	69
39	113	LESS	52	85	-91	40	134	DIV	60	47	-151
41	23	NOT	71	92	94	42	4	ADD	72	70	0
43	112	READ	38	57	-38	45	216	ADD	79	70	246
46	IF-Then-Else	DIV	54	55	0	47	208	READ	26	25	0
48	MINI	MORE	70	76	-18	49	69	DIV	24	64	0
50	MAX	NOT	25	90	-117	51	MOST	ADD	85	85	0
52	168	MAX	29	93	160	53	LESS	LESS	37	16	28
54	197	DIV	70	35	-93	55	176	MULT	70	17	28
56	204	MORE	64	81	221	57	118	NOT-EQ	78	75	0
58	123	ADD	57	70	239	59	LEAST	NOT-EQ	77	46	-131
60	92	MAX	51	43	-32	61	DIV	SUB	4	58	147
62	122	SUB	71	32	0	63	243	DIV	58	68	87
64	250	MULT	69	67	245	65	254	READ	52	24	0
66	POINT	EQ	49	61	0	67	WRITE	LESS	69	53	0
68	88	MIN	42	63	11	69	239	NOT	73	45	-249
70	71	EQ	90	38	-256	71	122	NOT	54	48	-93
72	109	EQ	70	54	0	73	113	NOT-EQ	86	91	0
74	96	ADD	30	82	-55	75	NOT	MAX	38	48	0
76	EQ	MAX	36	80	-112	77	241	EQ	70	21	0
78	WRITE	NOT-EQ	87	34	-84	79	EQ	MULT	55	74	0
80	READ	READ	87	71	-242	81	161	NOT-EQ	41	33	-144
82	WRITE	EQ	54	87	0	83	126	DIV	68	63	88
84	WRITE	ADD	70	18	-124	85	AVERAGE	NOT	56	85	50
86	WRITE	MIN	45	63	123	87	237	MORE	85	59	92
88	MINI	SUB	46	61	0	89	AVERAGE	MORE	22	49	-226
90	5	SUB	95	60	0	91	163	NOT-EQ	65	70	0
92	DIV	EQ	13	71	38	93	IF-Then-Else	EQ	84	35	0
94	WRITE	NOT-EQ	15	8	0	95	EQ	SUB	15	8	189

MINI

NODE	ACTION	BRANCH	A1	A2	B-C	NODE	ACTION	BRANCH	A1	A2	B-C
0	97	NOT-EQ	8	8	235	1	IF-Then-Else	MIN	10	2	0
2	64	MULT	7	10	252	3	DIV	LESS	5	0	29
5	190	EQ	1	0	0	7	70	MULT	10	1	0
8	116	MULT	10	10	178	9	169	NOT	0	9	0
10	178	READ	9	3	187						

Appendix B: This was the best "CAP" recognizing program at generation 77.

MAIN

NODE	ACTION	BRANCH	A1	A2	B-C	NODE	ACTION	BRANCH	A1	A2	B-C
0	223	MIN	14	14	-131	1	WRITE	DIV	74	78	0
2	116	MAX	48	41	-149	3	WRITE	READ	72	54	0
4	104	ADD	40	30	154	5	104	EQ	74	2	0
6	239	DIV	37	11	0	7	LEAST	MULT	18	12	0
8	68	ADD	11	28	0	9	WRITE	SUB	7	70	0
10	LIBRARY[36]	MAX	58	73	0	11	49	MAX	35	18	69
12	164	MULT	23	34	142	13	WRITE	MAX	13	25	205
14	221	MIN	19	32	0	15	83	NOT	44	8	0
16	62	NOT-EQ	5	47	0	17	109	NOT-EQ	28	51	74
18	READ	LESS	4	48	0	19	14	SUB	33	39	150
20	WRITE	READ	49	15	0	21	147	READ	47	73	0
22	READ	MAX	78	13	-78	23	145	READ	46	40	204
24	38	MIN	36	77	0	25	READ	NOT-EQ	36	15	-155
26	WRITE	DIV	7	22	0	27	21	LESS	44	78	-7
28	169	NOT	17	13	0	29	27	ADD	8	28	-160
30	233	EQ	12	31	243	31	211	EQ	47	43	0
32	WRITE	MORE	19	16	241	33	11	MIN	6	0	0
34	MULT	ADD	14	44	-94	35	162	MORE	24	42	0
36	WRITE	DIV	70	21	0	37	LEAST	LESS	64	26	-253
38	104	ADD	9	70	154	39	24	NOT	35	7	202
40	WRITE	SUB	31	30	0	41	READ	EQ	29	18	177
42	READ	LESS	45	46	-50	43	NOT	LESS	28	20	224
44	62	EQ	12	40	-141	45	203	NOT	14	74	-19
46	109	MORE	51	51	0	47	DIV	MORE	51	50	-172
48	202	NOT	29	49	-52	49	162	MORE	17	14	-70
50	33	READ	6	31	-102	51	DIFFERENCE	LESS	22	22	0
52	WRITE	MAX	52	65	205	53	104	ADD	9	9	154
54	83	NOT	55	27	0	55	62	EQ	75	68	-141
56	2	MULT	9	38	-84	57	104	ADD	62	7	154
58	11	MIN	53	7	0	59	MULT	ADD	57	55	-94
60	READ	ADD	58	10	0	61	242	MORE	61	78	-15
62	WRITE	DIV	66	79	0	63	145	READ	71	69	204
64	READ	MIN	58	78	0	65	READ	NOT-EQ	1	54	-155
66	169	NOT	57	52	0	67	212	MORE	58	64	-43
68	104	ADD	69	38	154	69	WRITE	SUB	72	75	0
70	NOT	LESS	66	3	224	71	33	READ	76	53	-102
72	162	MORE	57	57	-70	73	SUB	NOT-EQ	32	67	0
74	34	LESS	4	78	72	75	164	MULT	63	59	142
76	162	MORE	7	9	-70	77	239	DIV	61	78	0
78	AVERAGE	LESS	56	9	-7	79	DIV	MORE	58	58	-172

LIBRARY 36 (other libraries not shown)

NODE	ACTION	BRANCH	A1	A2	B-C	NODE	ACTION	BRANCH	A1	A2	B-C
0	AVERAGE	NOT-EQ	6	12	-56	1	ADD	MIN	14	0	-202
2	192	EQ	11	20	-190	3	READ	MORE	3	6	0
4	36	EQ	14	3	-75	6	186	MIN	10	19	244
7	LEAST	MULT	16	12	-68	8	65	MAX	9	13	45
9	READ	MIN	6	4	57	10	227	ADD	10	9	0
11	236	MAX	9	0	-207	12	LIBRARY[149]	ADD	21	2	77
13	WRITE	READ	8	6	-138	14	85	MIN	9	1	0
16	89	DIV	3	14	211	17	76	NOT-EQ	16	8	-245
19	111	MAX	1	23	204	20	39	READ	7	17	-82
21	VARIANCE	NOT	12	3	-241	23	104	EQ	8	6	0