# MT-CGP: Mixed Type Cartesian Genetic Programming

Simon Harding
simon@idsia.ch

Vincent Graziano
vincent@idsia.ch

Jürgen Leitner
juxi@idsia.ch

Jürgen Schmidhuber
juergen@idsia.ch
IDSIA-SUPSI-USI
Galleria 2
6928 Manno, Switzerland

## ABSTRACT

The majority of genetic programming implementations build expressions that only use a single data type. This is in contrast to human engineered programs that typically make use of multiple data types, as this provides the ability to express solutions in a more natural fashion. In this paper, we present a version of Cartesian Genetic Programming that handles multiple data types. We demonstrate that this allows evolution to quickly find competitive, compact, and human readable solutions on multiple classification tasks.

## Categories and Subject Descriptors

I.2.2 [**ARTIFICIAL INTELLIGENCE**]: Automatic Programming; D.1.2 [**Software**]: Automatic Programming

## General Terms

Algorithms

## Keywords

Cartesian Genetic Programming, Classifiers

## 1. INTRODUCTION

The use of Genetic Programming (GP) for classification tasks is a reasonably well studied problem [7]. Typically, the objects to be classified are represented by vectors in some n-dimensional space. In this scenario, a program is found that takes up to n-inputs, each a real number, and outputs a single value which is used to represent the class of the input object. That is, each component of the vector is presented independently to the program as a real and the functions of the program operate on pairs of reals and output reals. Although this approach has some benefits, it ultimately imposes some severe limitations.

One benefit is that GP can discover which components of the input are important for the classification problem and use precisely those. This is convenient if there is redundancy in the input values, or if the entire vector is not needed to solve the classification problem.

However, if all the components are required to classify the objects, then evolution has to find a way to incorporate each component individually. Successful classification of objects in high-dimensional space could unnecessarily require very large and very complicated programs. Consider the following example: Suppose we have a classification task that is solved by thresholding the norm-squared of the vector representation of the object. Evolution is then faced with the task of finding a program that individually squares each of the inputs and then sequentially sums the results. Such a program is not easily found. The approach does not scale well; typical evolutionary approaches are likely to fail in these settings. Further, even in the case that a solution is found it is unlikely to be compact or human readable—losing one of the attractive features of GP.

When hand building classifiers, the natural approach is to gather statistics on the entire object and then later combine these values with the values of specific components. For example, we might define a pixel of an image as important if it is outside one standard deviation from the mean pixel value. An implementation of this program would treat the entire image as a vector to produce a real value, and then compare this value to each pixel value for classification. This example suggests the method we introduce in this paper—allow GP to use mixed data types, to work with data in the same manner that a human programmer does, performing operations on vectors and reals in tandem.

The PushGP [25] implementation handled multiple data types by using multiple stacks. Floating point numbers and vectors were allocated their own stacks, and depending on the calling function, values are popped from the appropriate stack. This was further expanded in Push3 [27], where support for additional data types (and appropriate stacks) was added, including: integers, floating point numbers, Boolean values, symbolic names and code (PushGP supports the ability to generate new programs). The authors validated this approach on various problems, such as sorting, parity circuits and generating the Fibonacci sequence [25, 27].

In Strongly Typed Genetic Programming (ST-GP) [22], multiple data types (floats, integers, vectors) are used within a strongly typed representation. Functions have expected input types, and these define the output type of that function. In order for valid programs to be generated, trees are constructed so that the input values are of expected types.


*GECCO'12,* July 7-11, 2012, Philadelphia, Pennsylvania, USA.


**Figure 1:** An example of an MT-CGP genotype. Not all of the nodes in this genotype are connected. This is an example of neutrality in CGP. A mutation at the DIVIDE node could result in a connection to the COS node, making it part of the program. The inputs to the COS node are not shown since the node is not active.

Functions can be 'generic' (or overloaded) so that varying data types can be handled. To demonstrate its efficacy, ST-GP was tested on a regression problem and on evolving a Kalman Filter. The author compared the strongly typed representation with a dynamically typed version, and found that the strongly typed solved problems more efficiently [22].

PolyGP is a 'polymorphic' GP built using a $\lambda$-calculus approach [29, 5]. The approach specializes in list processing tasks, including the ability to execute a function on each element on a list (i.e. a map operation). Although the system should be general purpose, it only appears to have been used to evolve simple list processing operations, such as returning the $N$-th element from a list. A similar system, also built on $\lambda$-calculus improves on PolyGP by allowing the GP to explicitly evolve the types in tandem with the program and allows for arbitrary types to be used [2]. This approach was successfully used to evolve recursive list operations and basic Boolean operations.

In this paper we present Mixed Type Cartesian Genetic Programming (MT-CGP). As the name implies, this is an extension of the familiar CGP representation. Functions dynamically select the data type of their inputs, which in turn determines the output type. If they are unable to find suitable input types, a default value is returned. This ensures that all randomly generated programs are valid. We show that this method produces compact, human readable (C-style) programs that are able to efficiently solve a number of classification problems.

## 2. CARTESIAN GENETIC PROGRAMMING

Cartesian Genetic Programming (CGP) is a form of Genetic Programming in which programs are encoded in partially connected feed forward graphs [18, 19]. The genotype, given by a list of nodes, encodes the graph. For each node in the genome there is a vertex, represented by a function, and a description of the nodes from where the incoming edges are attached. This representation has a number of interesting properties.

For instance, not all of the nodes of a solution representation (the genotype) need to be connected to the output node of the program. As a result there are nodes in the representation that have no effect on the output, a feature known in GP as 'neutrality'. This has been shown to be very useful [21] in the evolutionary process. Also, because the genotype encodes a graph, there can be reuse of nodes, which makes the representation distinct from a classically tree-based GP representation. See Figure 1.

More recent versions of CGP have borrowed some key features of the Self Modifying CGP (SMCGP) representation [14, 15]. In keeping with SMCGP, each node contains four genes. One specifies the function that the node performs. Two genes are used to specify the nodes from which the function obtains its inputs. These connection addresses are defined relative to the current node and specify how many nodes *backwards* in the genotype/graph to connect. The final gene in each node is a floating point number that can be interpreted as either a parameter to a function, or used to generate values within the program. The input/output strategy used here is the same as SMCGP, where special functions are used to indicate which node to use as an output or how to obtain an input.

If a relative address extends beyond the extent of the genome it is connected to one of the inputs. If there are no output nodes in the genotype, then the output is taken from the last node in the genome. An illustrative example is shown in Figure 2.

## 3. CGP WITH MIXED DATA TYPES

To generate programs with multiple data types, MT-CGP uses an internal data type that can be cast to either a real number (double precision) or a vector of real numbers. The type of casting performed is dependent on a flag stored in the data type. That is, the casting used is explicit.

In MT-CGP the functions inspect the types of the input values being passed to it, and determine the most suitable operation to perform. This in turn determines the output type of the function. A default value is returned in cases where there is no suitable operation for the types being passed to the function.

How this works in practice is best illustrated with an example.

Consider the 'addition' function. This function takes two inputs and returns one output. There are four input cases that need to be considered: two reals, two vectors of the same dimension, a real and a vector, and two vectors of different dimension. The first two cases are trivial: the sum of two reals and vector addition. If the inputs are a real and a vector the function returns a vector has the real value added to every component of the vector. That is, $x+[a,b,c] = [a+x, b+x, c+x]$. In the case that the vectors lie in two different spaces the vector from the higher dimensional space is projected down into the space of the other vector. Vector addition is then performed and the output of the function is a vector. For example, $[a,b,c]+[d,e,f,g,h] = [a+d, b+e, c+f]$.

In general, each function will have a number of special cases that need to be carefully handled. Typically this is handled by limiting the calculations to the number of elements in the shortest vector, as was done in the above example.

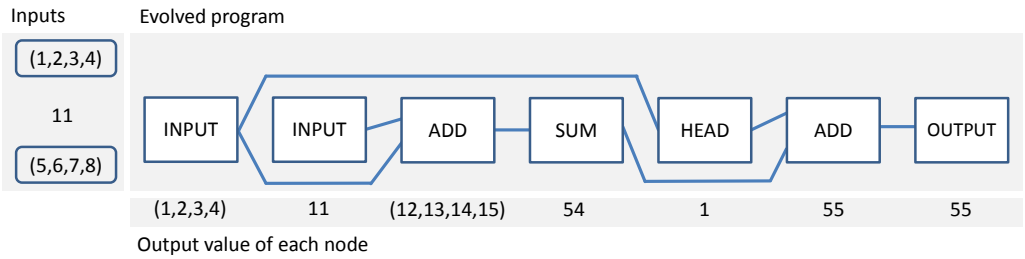


**Figure 2:** A worked example of an MT-CGP genotype. Each node has a function, and is connected to at most two other nodes. All the nodes in this example graph are connected, however not all input values are used. Data moves from left to right; the node labeled 'Output' specifies which node produces the program's output. The text beneath each node shows its output. The first two nodes are of type 'Input' which indicates they each read the next available input value. The 'Add' node then adds element-wise to the components of the vector, producing another vector. 'Sum' returns the total of all the values in the vector. 'Head' returns the first value in the vector. The second 'Add' node receives two real numbers as inputs, and therefore returns a real number.

Since MT-CGP can handle multiple and mixed data types it enjoys the property that it can draw from a wide-range of functions . Table 1 reproduces the complete function set used in the experiments in this paper. Clearly, this is only a rudimentary function set for the MT-CGP and it can be expanded to include domain specific functionality.

As with all forms of GP, MT-CGP can simply ignore functions that are not useful. By analyzing the evolved programs it is possible to see how the use of mixed data types is beneficial to CGP. We carry out such an analysis in Section 6.

## 4. EVOLUTIONARY ALGORITHM AND PARAMETERS

CGP is often used with a simple evolutionary strategy (ES), typically $1+n$, with $n = 4$. Here, parallel populations of evolution strategies are used. Individuals migrate between the populations through a single computer server node, and not 'peer-to-peer'.

Replacement in the evolutionary strategy has two important rules.

As usual, the best performing individual is selected as the parent. However, if the best performance is achieved by more than a single individual the ES chooses the one with the shortest program. In cases that this choice is not unique, one of newest individuals is selected. Choosing the shortest program (i.e. the individual with the fewest connected nodes) pushes evolution to find more compact programs. The preference of newer individuals has been shown to help CGP find solutions [20].

In keeping with the $1+n$ evolutionary strategy, MT-CGP does not use crossover. This 'mutation-only' approach is typical in CGP and SMCGP. The introduction of mixed data types does not preclude its usage however. As with SMCGP, it appears a high mutation rate (10%) performs best. The mutation rate is defined as the probability that a particular gene will be modified when generating an offspring.

MT-CGP requires relatively few parameters, even in a distributed system. Table 2 lists the most important parameters. It is important to note that these values have not been optimized, and therefore it may be possible to improve performance by selecting different values.

**Table 2:** Key Parameters in MT-CGP:

| Parameter | Value |
|---|---|
| Number of populations | 24 |
| Maximum evaluations | 10,000,000 |
| Synchronization interval | 100 evaluations (per client node) |
| Genotype length | 50 nodes |
| Mutation rate | 10% |
| Runs per experiment | 50 |

## 5. EXPERIMENTS

To demonstrate the validity of the mixed data type approach we have tested MT-CGP on four well-known binary classification tasks: Wisconsin breast cancer dataset, phoneme_cr dataset, diabetes1 dataset, and heart1 dataset. The parameters for the MT-CGP are unchanged across all experiments and are as given in Table 2. Likewise the function set is the same for all experiments and is reported in full in Table 1.

For each of the classification tasks, the inputs presented to the program were both the vector representing the object as well as the individual component values of the vector. For example, the Wisconsin breast cancer data uses datapoints in 9-dimensional space. The evolved programs were given these values as a vector and also as 9 individual real numbers, making a total of 10 inputs available to the program. Evolved programs are then able to select the most appropriate inputs to use in the classifier.

We use the Matthews Correlation Coefficient (MCC) [28] to measure the fitness of our classifiers. First the 'confusion matrix' is found: true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). The MMC is calculated as follows:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

An MCC of 0 indicates that the classifier is working no better than chance. A score of 1 is achieved by a perfect classifier,

**Table 1:** Functions available in MT-CGP. Due to space issues, the complete behavior of each function cannot be presented here. Arity is the number of inputs used by the function. A 'yes' value in the parameter column indicates that the function makes use of the real number value stored in the calling CGP node.

| Function Name | Arity | Uses parameter | Description |
|---|---|---|---|
| | | List Processing | |
| Head | 1 | | Returns the first element of a vector |
| Last | 1 | | Returns the last element of a vector |
| Length | 1 | | Returns the length of the vector |
| Tail | 1 | | Returns all but the first element of a vector |
| Differences | 1 | | Returns a vector containing the differences between each pair in the input vector |
| AvgDifferences | 1 | | Returns the average of the differences between pairs in the vector |
| Rotate | 1 | Yes | Rotates the indexes of the elements by the parameter's value |
| Reverse | 1 | | Reverses the elements in the list |
| PushBack | 2 | | Inserts the values of the second input to the end of the first input |
| PushFront | 2 | | Inserts the values of the second input to the front of the first input |
| Set | 2 | | If one input is a vector, and the other is a real, return a new vector where all values are the real input value |
| Sum | 1 | | Sums all the elements in the input vector |
| | | Mathematical | |
| Add, Subtract, Multiply, Divide, Abs, Sqrt, Pow, PowInt, Sin, Tan, Cos, Tanh, Exp, <, > | 2 | | Returns mathematical result. Element-wise if at least one input is a vector. |
| | | Statistical | |
| StandardDeviation, Skew, Kurtosis, Mean, Median | 1 | | Returns the given statistic for an input vector. |
| Range | 1 | | Returns the max value - min value of a vector |
| Max1 | 1 | | Returns the maximum value of a vector |
| Max2 | 2 | | Element wise, returns max of two vectors or a vector and real |
| Min1 | 1 | | Returns the minimum value of a vector |
| Min2 | 2 | | Element wise, returns min of two vectors or a vector and real |
| | | Misc | |
| VecFromDouble | 1 | | Tries to cast the input to a double, and then casts that to a single element vector |
| NOP | 1 | | Passes through the first input |
| GetInput | 0 | | Returns the current input, moves pointer forward |
| GetPreviousInput | 0 | | Moves the inputs pointer backwards, returns that input |
| SkipInput | 0 | Yes | Moves the input pointer parameter positions, returns that input |
| Const | 0 | Yes | Returns the parameter value |
| ConstVectorD | 0 | Yes | Returns a vector of the default length, all values set to the parameter value |
| EmptyVectorD | 0 | | Returns an empty vector of a default length |
| Output | 1 | | Indicates that this node should be used as an output |

a score of $-1$ indicates that the classifier is perfect, but has inverted the output classes. Finally, the fitness of an individual is given by:

$$fitness = 1 - |MCC|\,,$$

with values closer to 0 being more fit.

The MCC is insensitive to differences in class size, making it a nice choice for many binary classification tasks. The confusion matrix also allows for easy calculation of the classification rate as a percentage of correctly classified fitness cases.

## 6. RESULTS

MT-CGP was run on four different datasets. A summary of the results can be seen in Table 3. Table 3 also shows how many evaluations were required to find the best generalizing individual. Detailed analysis of the results, and comparisons to other work, can be found in Table 4.

To ensure accurate statistics are found, experiments were run 50 times. Again, the parameters and function set were kept consistent through each experiment.

**Table 3:** Results from the validation set of each of the data sets explored.

| | **Cancer** | **Phoneme** | **Diabetes1** | **Heart1** |
|---|---|---|---|---|
| Max MCC | 0.98 | 0.575 | 0.55 | 0.72 |
| Min MCC | 0.943 | 0.434 | 0.44 | 0.51 |
| Avg. MCC | 0.959 | 0.515 | 0.50 | 0.61 |
| S.D. | 0.009 | 0.0384 | 0.0233 | 0.0394 |
| **Max Acc.** | **99.3** | **80.4** | **79.2** | **85.3** |
| Min Acc. | 97.9 | 75.9 | 75.0 | 76.0 |
| Avg. Acc. | 98.5 | 78.2 | 77.3 | 80.2 |
| S.D. | 0.3 | 1.2 | 0.77 | 1.8 |
| Min Evals | 7k | 17k | 35k | 25k |
| Max Evals | 1,997k | 1,973k | 9,540k | 9,647k |
| Avg. Evals | 1,204k | 879k | 3,204k | 1,539k |
| S.D. | 555k | 601k | 2,94k | 2,011k |

### 6.1 Wisconsin Breast Cancer Dataset

The Wisconsin Breast Cancer Dataset is a well tested benchmark in classification and is part of the UCI Machine Learning Repository [9]. The dataset is split into 400 training examples and 283 validation examples (examples with missing values are discarded). It was found that MT-CGP works well for this problem, and compares well to other recent work published on the same dataset. A summary of the statistical results can be seen in Table 3. In Table 4 the results are compared to a range of other recent approaches. (Note: there are different methodologies for dividing the data into training and validation sets. The different approaches report mean results and therefore may not be directly comparable. Further, classifiers can be sensitive to how the dataset is split when there is a class imbalance or the problem is relatively easy to solve).

**Table 4:** Comparison of various classifiers on the datasets.

| **Breast Cancer** | % Accuracy |
|---|---|
| Radial Basis Function Networks [16] | 49.8 |
| Probabilistic Neural Networks [16] | 49.8 |
| ANN (Back Propagation) [16] | 51.9 |
| Recurrent Neural Network [16] | 52.7 |
| Competitive Neural network [16] | 74.5 |
| Learning vector Quantization [16] | 95.8 |
| Support Vector Machine [17] [1] | 96.9 |
| Memetic Pareto ANN [1] | 98.1 |
| ANN (Back Propagation) [1] | 98.1 |
| Genetic Programming [12] [2] | 98.2 |
| Support Vector Machine [13] [2] | 98.4 |
| MT-CGP Maximum Accuracy | 99.3 |
| MT-CGP Minimum Accuracy | 97.9 |
| MT-CGP Avg. Accuracy. | 98.5 |
| **Phoneme_CR** | % Accuracy |
| Linear Bayes [6] | 73.0 |
| Quadratic Bayes [6] | 75.4 |
| Fuzzy Classifier [6] | 76.9 |
| Quadratic Bayes [10] | 78.7 |
| Neural network [6] | 79.2 |
| Piecewise linear [6] | 82.2 |
| C4.5 [6] | 83.9 |
| MLP Neural Network [10] | 86.3 |
| k-Nearest Neighbor [10] | 87.8 |
| MT-CGP Maximum Accuracy | 80.4 |
| MT-CGP Minimum Accuracy | 75.9 |
| MT-CGP Avg. Accuracy. | 78.2 |
| **Diabetes1** | % Accuracy |
| Self-generating Neural Tree (SGNT) [8] | 68.6 |
| Learning Vector Quantization (LVQ) [8] | 69.3 |
| 1-Nearest Neighbor [8] | 69.8 |
| Self-generating Prototypes ( SGP2) [8] | 71.9 |
| Linear Genetic Programming [3] | 72.2 |
| k-Nearest Neighbor [8] | 72.4 |
| Self-generating Prototypes (SGP1) [8] | 72.9 |
| Gaussian mixture models [8] | 72.9, 68.2 |
| Neural Network [3] | 75.9 |
| Infix Form Genetic Programming [23] [3] | 77.6 |
| Support Vector Machine [13] [2] | 77.6 |
| Principal Curve Classifier [4] | 78.2 |
| MT-CGP Maximum Accuracy | 79.2 |
| MT-CGP Minimum Accuracy | 75.0 |
| MT-CGP Avg. Accuracy. | 77.3 |
| **Heart1** | % Accuracy |
| Neural Network [24] | 80.3 |
| Linear GP [3] | 81.3 |
| Support Vector Machine [13] [2] | 83.2 |
| Infix Form GP [23] | 84.0 |
| MT-CGP Maximum Accuracy | 85.3 |
| MT-CGP Minimum Accuracy | 76.0 |
| MT-CGP Avg. Accuracy. | 80.2 |

[1] Results are based on leave-one-out validation, and so may not directly comparable.

[2] Results are based on 10 fold validation, and so may not directly comparable

[3] This paper does not use the pre-defined training and validation split, so the comparison may not be accurate

Below is the program, found by evolution, that gives the best result (99.3% accuracy) on the validation data:

```
node0 = Inputs[0];
node1 = Multiply(node0,node0);
node2 = Mean(node1);
node4 = Sum(node2);
node39 = Max1(node4);
node45 = ConstVectorD(5.98602127283812);
node49 = PowInt(node45, node39);
if ((int)Head(node49) <=0)
  return 1;
else
  return 0;
```

The 'if' statement at the end of the program is from the fitness function, and each of the other lines corresponds to a node in the genotype. The genotype contains 50 nodes, but only 7 of them were connected to form a program. 'nodeN' can be treated as a variable which can be either a vector or a real number. The first line sets the variable node0 to the input vector. The values are then squared, and stored in node1. node2 is the average value of the elements in the vector node1. node4 is largely redundant, as node2 is a single value. node45 is a vector which is the same length as the input vector, but all elements are set to a constant value. node49's value is a vector, made by raising each element in node45 to the truncated (integer) value of node39. The 'if' statement is used to turn the actual output into either 0 or 1, which is then compared against the expected class. In this example, the (int) type cast in the 'if' statement turns out to be a crucial step. All values of node49 are positive and much larger than 0. However, when casting to an integer, the *very large* values cannot fit and cause an overflow. This causes the integer version of some values to be less than 0.

## 6.2 Phoneme_CR

The phoneme_cr dataset is another benchmark data set for classifiers [11][1]. It consists of 5 real number input variables and one binary class. There are 5,404 fitness cases, and this is split in two equal parts (at random) to produce a training and validation set. From previous results (and by design), it can be seen that this is a more challenging data set, where the highest classification accuracy is 87.8%.

From Table 4 it can be seen that whilst MT-CGP is competitive, it does not produce the best results. This may be because the parameters chosen do not allow MT-CGP to find a good program. The best solution is 20 operations long, and therefore is quite large with respect to the size of the genotype. Previous experience with CGP indicates better performance may be achieved by using a larger genotype.

## 6.3 Proben Diabetes1

This dataset is 'diabetes1' from the Proben machine learning repository. It is a binary classification problem consisting of 576 training cases and 192 validation test cases. Each case contains 8 real numbered inputs. Table 4 shows that MT-CGP does well compared to previous work.

[1]`http://www.dice.ucl.ac.be/neural-nets/Research/Projects/ELENA/elena.htm`

## 6.4 Proben Heart1

'Heart1' is another proben1 dataset, based on data for predicting heart disease [2]. It consists of 690 training examples and 230 validation cases. There are 35 real valued inputs, and is a binary classification problem.

Table 4 shows that MT-CGP performs favorably to previous methodologies.

# 7. PROGRAM LENGTH

The ES used prefers the shortest program among those of equal fitness. The aim was to get MT-CGP to find solutions that are parsimonious. The are two immediate advantages for such solutions: the chances that humans can understand and learn from the resultant programs are increased [26], and shorter, simpler solutions tend to generalize to larger problems more easily [30].

Consider the following program:

```
node0 = Mean(input[0]);
node1 = Exp(node0);
node49 = Pow(node0, node1);
if ((int)Head(node49) <=0)
  return 1;
else
  return 0;
```

On the breast cancer dataset it achieves a classification accuracy of 97.9%, competitive with the state-of-the-art classifiers on this dataset. Although this wasn't the best performing MT-CGP program on the dataset, the evolved program is extremely simple and is easy for humans to follow (Note: it also abuses the double-int conversion feature, discussed above).

For the problems investigated here, there was a general trend that the best performing individuals had longer programs. An example of this can be seen in Figure 3 where the average classification error for various program lengths is plotted. Although longer programs generally do better, a trade off may be made in terms of generalization, human interpretability, and processing speed. It may be acceptable, and indeed preferable, to use shorter programs even if they do not perform as well.

# 8. CONCLUSIONS

The results show that MT-CGP is a promising technique for use in classification problems. In the example datasets it performs competitively with, and in some cases outperforms, other established machine learning techniques. In principle, MT-CGP has some advantages over previous methodologies. It is able to produce human readable output. In the case of the breast cancer data, the program showed that a good classification rate can be achieved through a simple calculation.

In all the programs inspected, evolution was seen to take advantage of the vector as an input. The ability to collect statistics on the vector input and then perform further processing on components of the vector appears to be very useful. This suggests that GP methods which mix data types

[2]Data collected by: Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D., University Hospital, Zurich, Switzerland: William Steinbrunn, M.D., University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D. and V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.

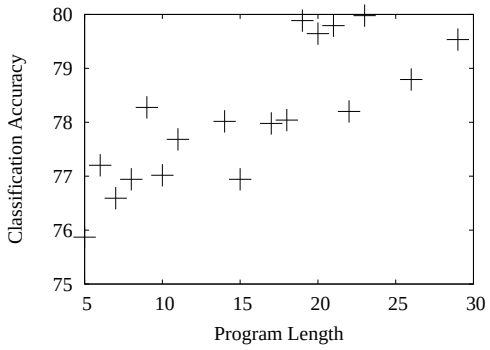


**Figure 3:** For the best individuals found by evolution, the classification accuracy is related to the program length. The plot shows average classification error versus program length on the phoneme dataset. Although longer programs generally do better, a trade off may be made in terms of generalization, human interpretability, and processing speed.

will—in most cases—have an edge over GP techniques that cannot. The latter would have to build significantly more complex programs to achieve the same functionality.

MT-CGP should also be successful on problems other than classification. Indeed, it should work on any problem where CGP has been used. Given its ability to statistically analyze streams of values and simultaneously handle vectors of different dimension, problems such as time series prediction seem to be an ideal candidate for future exploration.

There are two aspects of the algorithm that we plan to examine in detail in the future. By incorporating domain specific functions into the set, we expect to sharpen previous results and to be able to tackle more difficult tasks. For example, there are many metrics used in finance, e.g., volatility, whose incorporation could prove useful in highly difficult financial series classification problems.

More generally, we will investigate how best to constrain the ES so that program length is optimized. The solution used herein is likely to be too aggressive. Initially, it seems better to explore programs of varying lengths, and then only later exploit the knowledge contained in the gene pool to find short solutions. We anticipate that borrowing ideas from [26] will help us to better balance performance with program length.

## 9. REFERENCES


[1] H. A. Abbass. An evolutionary artificial neural networks approach for breast cancer diagnosis. *Artificial Intelligence in Medicine*, 25:265–281, 2002.

[2] F. Binard and A. Felty. Genetic programming with polymorphic types and higher-order functions. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 1187–1194, New York, NY, USA, 2008. ACM.

[3] M. Brameier and W. Banzhaf. A comparison of linear genetic programming and neural networks in medical data mining. *Evolutionary Computation, IEEE Transactions on*, 5(1):17 –26, feb 2001.

[4] K. Chang and J. Ghosh. Principal curve classifier-a nonlinear approach to pattern classification. In *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on*, volume 1, pages 695 –700 vol.1, may 1998.

[5] C. Clack and T. Yu. Performance enhanced genetic programming. In *In Proceedings of the Sixth Conference on Evolutionary Programming*, pages 87–100. Springer, 1997.

[6] A. Eftekhari, H. A. Moghaddam, M. Forouzanfar, and J. Alirezaie. Incremental local linear fuzzy classifier in fisher space. *EURASIP J. Adv. Signal Process*, 2009:15:1–15:9, January 2009.

[7] P. Espejo, S. Ventura, and F. Herrera. A survey on the application of genetic programming to classification. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 40(2):121 –144, march 2010.

[8] H. A. Fayed, S. Hashem, and A. F. Atiya. Self-generating prototypes for pattern classification. *Pattern Recognition*, pages 1498–1509, 2007.

[9] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[10] G. Giacinto and F. Roli. Dynamic classifier selection based on multiple classifier behaviour. *Pattern Recognition*, 34:1879–1881, 2001.

[11] A. Guerin-Dugue and et al. Deliverable r3-b4-p task b4: Benchmarks. Technical report, Technical Report, 1995.

[12] P.-F. Guo, P. Bhattacharya, and N. Kharma. Automated synthesis of feature functions for pattern detection. In *Electrical and Computer Engineering (CCECE), 2010 23rd Canadian Conference on*, pages 1 –4, may 2010.

[13] P. Hao, L. Tsai, and M. Lin. A new support vector classification algorithm with parametric-margin model. In *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 420–425. IEEE, 2008.

[14] S. Harding, J. F. Miller, and W. Banzhaf. Developments in cartesian genetic programming: self-modifying CGP. *Genetic Programming and Evolvable Machines*, 11(3-4):397–439, 2010.

[15] S. Harding, J. F. Miller, and W. Banzhaf. A survey of self modifying cgp. *Genetic Programming Theory and Practice, 2010*, 2010.

[16] R. Janghel, A. Shukla, R. Tiwari, and R. Kala. Intelligent decision support system for breast cancer. In Y. Tan, Y. Shi, and K. Tan, editors, *Advances in Swarm Intelligence*, volume 6146 of *Lecture Notes in Computer Science*, pages 351–358. Springer Berlin / Heidelberg, 2010.

[17] H. X. Liu, R. S. Zhang, F. Luan, X. J. Yao, M. C. Liu, Z. D. Hu, and B. T. Fan. Diagnosing breast cancer based on support vector machines. *Journal of Chemical Information and Computer Sciences*, 43(3):900–907, 2003.

[18] J. F. Miller. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *Proceedings of the 1999 Genetic and Evolutionary Computation Conference (GECCO)*, pages 1135–1142, Orlando, Florida, 1999. Morgan Kaufmann.

[19] J. F. Miller, editor. *Cartesian Genetic Programming*. Natural Computing Series. Springer, 2011.

[20] J. F. Miller, D. Job, and V. K. Vassilev. Principles in the evolutionary design of digital circuits - part I. *Genetic Programming and Evolvable Machines*, 1(1):8–35, 2000.

[21] J. F. Miller and S. L. Smith. Redundancy and computational efficiency in cartesian genetic programming. In *IEEE Transactions on Evoluationary Computation*, volume 10, pages 167–174, 2006.

[22] D. J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.

[23] M. Oltean and C. Grosan. Solving classification problems using infix form genetic programming. In *Programming, The 5 th International Symposium on Intelligent Data Analysis, M. Berthold (et al), (Editors), LNCS 2810*, pages 242–252. Springer, 2003.

[24] L. Prechelt. PROBEN1 — A set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany, Sep 1994.

[25] A. Robinson and L. Spector. Using genetic programming with multiple data types and automatic modularization to evolve decentralized and coordinated navigation in multi-agent systems. In E. Cantú-Paz, editor, *Late Breaking Papers at the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 391–396, New York, NY, July 2002. AAAI.

[26] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81, 2009.

[27] L. Spector, J. Klein, and M. Keijzer. The push3 execution stack and the evolution of control. In H.-G. Beyer, U.-M. O'Reilly, D. V. Arnold, W. Banzhaf, C. Blum, E. W. Bonabeau, E. Cantu-Paz, et. al., editors, *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 2, pages 1689–1696, Washington DC, USA, 25-29 June 2005. ACM Press.

[28] Wikipedia. Matthews correlation coefficient — wikipedia, the free encyclopedia, 2011. [Online; accessed 27-January-2012].

[29] T. Yu and C. Clack. Polygp: a polymorphic genetic programming system in haskell. In *Proc. of the 3rd Annual Conf. Genetic Programming*, pages 416–421. Morgan Kaufmann, 1998.

[30] B.-T. Zhang and H. Muhlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3:17–38, 1995.