Evolutionary Partitioning Regression with Function Stacks

Daniel Ashlock, Senior Member IEEE and Joseph Brown

***Abstract*—Partitioning regression is the simultaneous fitting of multiple models to a set of data and partition of that data into easily modelled classes. The key to partitioning regression with evolution is minimum error assignment during fitness evaluation. Assigning a point to the model for which it has the least error while using evolution to minimize total model error encourages the evolution of models that cleanly partition data. This study demonstrates the efficacy of partitioning regression with two or three models on simple bivariate data sets. Possible generalizations to the general case of clustering are outlined.**

I. Introduction

Symbolic regression [9], [10] is the practice of using genetic programming to fit an error-minimizing model to a data set. A clear advantage of this technique is that the regression is not just fitting parameters, but searching the entire space of models, withing resource constraints on tree-depth or formula size. A clear disadvantage of the technique is also that the regression is searching the same entire space of models. In particular, when simply fitting parameters to an already selected model, it is already known what constants are needed and only their values must be discovered. When performing regression with genetic programming, the issue of setting the value of ephemeral constants is critical.

In this study we offer an intermediate step, based on genetic programming with *function stacks* a type of Cartesian genetic programming [13]. A function stack is an array of nodes of fixed length. Each node contains an arithmetic operation and pointers to either input variables or to nodes with a higher index in the array. This latter constraint makes function stacks *directed acyclic graphs* (DAGs) with an unambiguous order of evaluation, specifying potentially complex formulae. The advantage of Cartesian genetic programming over standard tree-based genetic programming lies in its ability to re-use its own code more efficiently.

Consider the problem of computing the parity of three Boolean variables. The solution is a logic function that is true when an odd number of its three inputs are also true. In the available operators are **AND**, **OR**, **NAND**, and **NOR** then Figure 1 shows a tree and a DAG-structured solution. The duplicated portions of the parity tree, highlighted with dotted lines are functionally equivalent to the leftmost three nodes in the DAG. The value computed by those three nodes is used *twice*. The practical effect is that, without a very well chosen ADF (genetic programming style subroutine), the parse tree requires a number of nodes that is an exponential function of the number of inputs while the DAG-structured solution grows linearly. This sufficient to explain the use of function stacks as the GP-technology in this study.

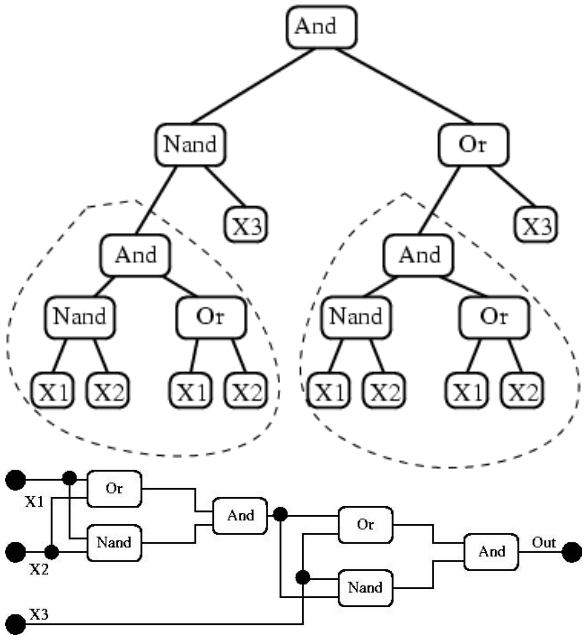

Fig. 1. A parse tree and a DAG solution to the 3-parity problem.

It remains to explain how to use function stacks for partitioning regression. The typical output value of a function stack is the value in its lowest index (zeroth) node, but any node of a function stack could be used as an output node. In order to perform partitioning regression, into k data classes, we need to have k models. For each point, the error of all the models is computed for that point, and if the error of the tth model is lowest the point is assigned to the ith cluster. We use the nodes of index $0, 1, \ldots, k-1$ as out k models. This has the twin advantages of encoding all the models in a single data structure and permitting re-use of common information in the function stack by multiple models.

A. Fitness for Partitioning Regression

The fitness function used to perform partitioning regression with is based on the same minimum error assignment used to partition data. Suppose that e_i^k is the error $y - y_{pred}$ between the data and its predicted value for data point i using model k. Then the fitness of a multi-model for n data points, encoded

Daniel Ashlock is with the Department of Mathematics and Statistics at the University of Guelph,dashlock@uoguelph.ca

Joseph Brown is with the Artificial Intelligence Games Development Lab, Innopolis University, Innopolios, Republic of Tatarstan, Russia, j.brown@innopolis.ru

The authors thank the Natural Sciences and Engineering Research Council of Canada (NSERC) for supporting this work.

as a function stack, is:

$$fitness = \sum_{i=1}^{n} min\left((e_i^0)^2, (e_i^1)^2, \ldots, (e_i^k)^2\right) \quad (1)$$

The sum of minimal squared error, minimizing across the available models. This is called the *best squared error* (BSE) function for multiple models.

The demonstration that evolutionary partitioning regression is possible with function stacks is split, in this study, into two phases. In the first, partitioning regression is demonstrated *without* function stacks by simple evolving the parameters for data sampled from two lines and data sampled from a line and a a quadratic curve. A simple evolutionary parameter optimizer is then used to fit the coefficients of two lines (four parameters) or a line and a quadratic (five parameters) using a different version of the . An example of this sort of simple evolutionary partitioning regression is shown in Figure 2. Once proof-of-concept for partitioning regression is demonstrated with the simple optimizer, then experiments with function stacks that encode multiple models are performed.

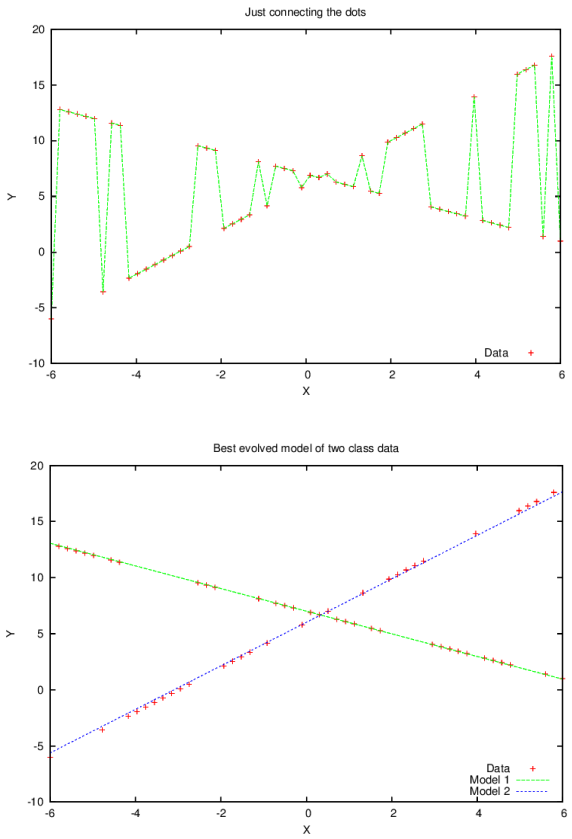

Fig. 2. The upper panel shows a data set plotted so that dots with successively larger x-coordinates are joined by lines. The lower panel shows the result of evolutionary partitioning regression fitting the models $y = ax + b$ or $y = cx + d$ optimizing the parameters (a, b, c, d) using BSE fitness.

The remainder of this study is organized as follows. Section II give background on evolutionary regression. Section III specifies the details of function stacks, gives details of the evolutionary algorithms used, and describes the experiments performed. Section IV gives and discusses the results while section V drawn conclusions and outlines potential next steps.

II. Background on Symbolic Regression

Symbolic regression is introduced in [9]. In [15] the authors present a MatLab toolbox that performs symbolic regression while making available non-linear transformations of the input variables in a selectable manner that improves performance. Cartesian genetic programming has also been used for symbolic regression [14]. *Semantic crossover* is a technique for reducing the disruption caused by sub-tree crossover in genetic programming. An example of an application of semantic crossover to symbolic regression appears in [6]. Function stacks, by using a small, fixed data structure give another method of avoiding disruptive crossover. The ability to use the output of nodes in function stack multiple times permits them to encode complex formulas without needed as much space as the traditional tree-based encoding for genetic programming.

In [16] the authors suggest a technique for dealing with data that has different functional forms on different intervals. They augment the evolvable formulas of genetic programming with evolvable breakpoints that divide the independent variable into multiple ranges, each of which gets it own symbolic model. The approach has two problems:

1) I assumes the data segregates nicely across disjoint intervals. While this happens quite often in introductory calculus classes, it is less common in applications, mostly occurring in time-series data when, for example, something breaks changing a pattern of vibration.
2) The technique generalizes very badly to more than one independent variable. "Break points" along a single axis have a very simple form. Dividing the $x - y$ plane or three-space into domains is a much more difficult proposition.

Partitioning regression can fins break-points in data sets if they exist, so it can solve the class of problems posed in [16], but it avoids both the problems above by performing the partitioning of the data implicitly rather than explicitly. A similar proposal that also incorporates scaling in a manner that improves modeling performance appears in [7].

III. Experimental Design

Two different evolutionary algorithms were used to demonstrate partitioning regression, a simple evolutionary regression algorithm that fits two line or a line and a parabola to data sampled from, respectively, two lines and a line and a parabola. This is intended as a verification-of-concept; if the simple regressor that knows the model cannot correctly partition data then a more complex algorithm using function stacks is dubious not likely to perform well.

A. The Simple Evolutionary Regressor

The simple evolutionary regressor is applied to two balanced sets of data with 60 points. The first has thirty points sampled from $y = 2x + 6$ and thirty sampled from $y = -x + 7$. The

Operation	Arguments	Definition
Neg	1	Negate argument
Scl	1	Scale argument by ephemeral constant
Sqt	1	Square root of argument
Sqr	1	Square of argument
Sin	1	Sine of argument
Cos	1	Cosine of argument
Atn	1	Arctangent of argument
+	2	Addition
-	2	Subtraction
*	2	Multiplication
Pro/	2	Protected division¡$1E \pm 6$
Max	2	Maximum of arguments
Min	2	Minimum of arguments
Wavg	2	Weighted average using positive decimal of ephemeral constant as weight for first argument

TABLE I
OPERATIONS AND TERMINALS AVAILABLE FOR USE IN FUNCTION STACKS.

x values were taken uniformly at random from the interval [-6,6]. The second data set had the same domain but was sampled from the functions $y = x + 1$ and $y = x^2 - 5$. A third experiment was performed using the same data set as the first experiment, but fitting three lines when only two are needed. For each experiment, the evolutionary algorithm is run 30 times.

The algorithm uses a population of 100 genes of the form (a, b, c, d) for the models $y_1 = ax + b$ and $y_1 = cx + d$, (a,b,c,d,e) for the models $y_1 = ax + b$ and $y_2 = cx^2 + dx + e$, and (a,b,c,d,e,f) for the models $y_1 = ax + b$, $y_1 = cx + d$, and $y_3 = ex + f$. The algorithm uses one point crossover, because the gene is short and the coefficient of the models are strongly linked, making it likely there is epistasis. Mutation consists of selecting 1-5 loci, with the number selected uniformly at random, and adding a uniform random variable in the range [-0.25,0.25] to the selected coefficient. The size of the mutation is relatively small and particular coordinates may be selected multiple times. A million updatings using a steady state algorithm are performed. Selection and replacement are performed with size four single tournament selection; four genes are chosen and the two lowest error models reproduce and replace the two higher error models.

B. Function Stacks

The algorithm uses function stacks of length 12 using the operations and terminals given in Table I. The function stack is an array of nodes, each of which contains an ephemeral constant in the range $-5 \leq c \leq 5$, an operation (binary or unary), and two arguments which may be the variable x, the ephemeral constant in the node, or the index of a node with a higher index. To evaluate the function stack, the zeroth node is evaluated. If the zeroth node points to other nodes, they are evaluated and return their value, and this proceeds recursively until all required nodes are evaluated. The nodes used are marked.

In order to perform partitioning regression, when k models are needed, any function stack that did not evaluate its kth smallest nodes is awarded a "very bad" fitness of 1E6. This forces function stacks that do not have meaningful output in the top k nodes to be replaced by evolution. An example of a function stack appears in Figure 3. It obtains a fitness of 3.51344e-06 on a data set sampled from two lines, representing a correct partitioning model of the data.

```
 0) sub       L1        L4
 1) add       L6    5.0033
 4) add       L11        L6
 6) sub       L8        L7
 7) wav       L11        L8 wt 0.5003
 8) mlt   4.9974         X1
11) add  -1.9944         X1
```

Fig. 3. An example of a 12-node function stack. Only nodes called by model nodes, nodes 0 and 1, are displayed.

The evolutionary algorithm used to evolve function stacks uses a population of 1000 function stacks with 12 nodes. Reproduction uses size four tournament selection, as described for the simple evolutionary regression operators. Reproduction uses two point crossover of the function stack and 1-3 mutations with the number of mutations selected uniformly at random. Each mutation picks a node uniformly at random and modifies one of its operation, ephemeral constant, or its two arguments. Whichever object is modified is replaced with a value selected uniformly at random. For each experiment, the algorithm is run 100 times.

The data sets used for the experiments with function stacks are as follows.

1) The same data set as the two line test with simple evolutionary regression.
2) A data set of 150 points, balanced with 50 point sampled from each of $y_1 = 2x+1$, $y_2 = x-1$, and $y_3 = -x+5$.
3) A set of 100 points, 50 each from the line $y_1 = x + 1$ and the parabola $y_2 = x^2 - 5$.
4) A data set of 90 points with 30 points each sampled from the lines $y_1 = x - 3$ for $-4 \leq x < -2$, $y_2 = x$ for $-2 \leq x < 2$, and $y_2 = x + 3$ for $2 \leq x < 6$.
5) A data set of 90 points with 30 points each sampled from the lines $y_1 = 2x + 3$ for $-4 \leq x < -2$, $y_2 = x - 2$ for $-2 \leq x < 2$, and $y_2 = -x$ for $2 \leq x < 6$.

The latter two sets were to test the ability to partition data that is already partitioned with break points, as in [16]. Interestingly, this made the problem somewhat harder.

IV. Results and Discussion

In the discussion of results, we say that a model *captures* a point if, among the models under consideration, it has the lowest error value on that point. This notion is most useful when more models are used than turn out to be needed; it supports the implementation, discussed in Section V of extinction operators.

Figure 4 shows the best results for the experiments using simple evolutionary regression. The distribution of fitness

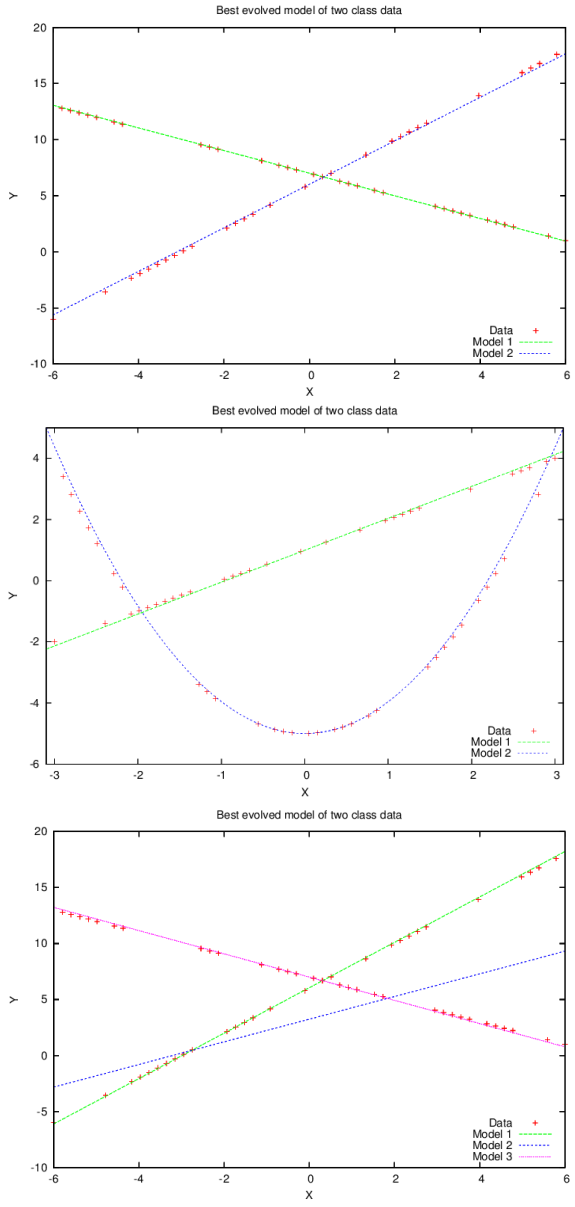

Fig. 4. Best multi-models found with the simple evolutionary regressors for the three experiments performed. Models are shown as lines, data sets as crosses.

values is shown in Figure 5. BSE fitness is an error measure and shows that good models were located, but not nearly exact ones (with fitness so small that it suggests the error is computational round-off error). We will see nearly exact models in the experiments with function stacks. The simple evolutionary regressor does correctly partition the data. After the data is partitioned, classical methods like least squares fit may be used to obtain excellent models, at least for these simple data where the model is known.

The experiment where three linear models were used for two lines also had an interesting outcome. The result shown in the last panel of Figure 4 show that two of the lines correctly partition the data while one of the lines simply accurately

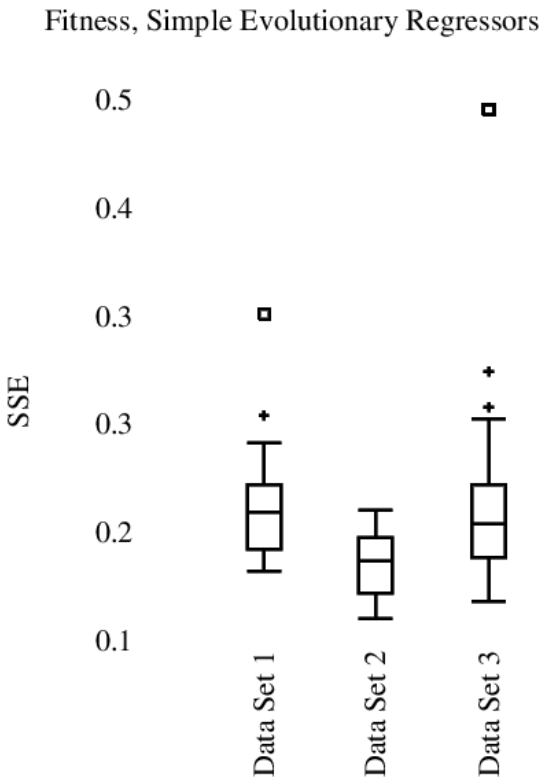

Fig. 5. Distribution of BSE fitness values for the simple evolutionary regression experiments.

models one point. This means that the error contribution from the third model was only its error at that point - that is the only point where it would have lowest error. In [5], regression with multiple models using a type of evolutionary algorithm called the *multiple-worlds model* was proposed. A key part of this model is *extinction* where, when a model ceases to capture a significant fraction of the data, it is removed, reducing the total number of models. It would be easy to implement extinction in the algorithm developed in this study, permitting evolution to chose the number of models below an initial allotment.

In Figure 5, data sets 1 and 3 are the same but approached with two and three linear models. The high outlier is one where two lines both badly approximated the data sampled from one of the original underlying lines used to generate the data; this shows that having the wrong number of models makes the problem harder - but not so mach harder that solution does not occur. This example provides additional motivation for implementing and extinction operator.

A. *Function Stack Results*

None of the solutions found with simple regression had fitness values better than 0.1, a good fit, but not a direct discovery of the generating formulas for the data. All of the experiments with function-stack based regression managed to discover data models with error below $1e-5$; hand verification showed that at least some of these were function stacks that had reproduced the models used to generate the data. Figures

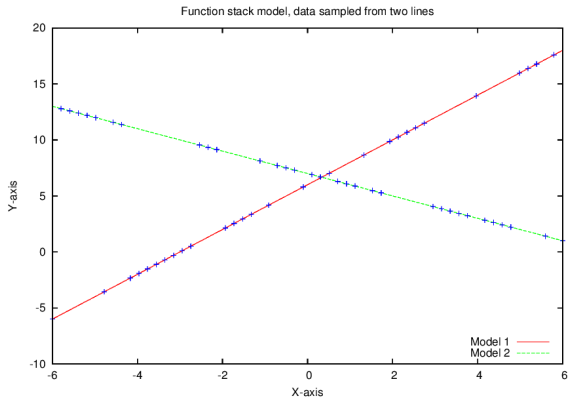

Fig. 6. Results using function stacks for partitioning regression on the first data set with two lines.

6, 8, 9, and 10 show plots of highly accurate models for each data set - as well as other, interesting models. Table II tabulated the number of such very accurate models discovered for each data set.

TABLE II
NUMBER OF HIGHLY ACCURATE MODELS DISCOVERED FOR EACH DATA SET WHEN FUNCTIONS STACKS ARE PERFORMING THE MODELING.

Data Set	Low error Models
1	24
2	3
3	43
4	13
5	35

The information in Table II shows that the problems vary in difficulty and, that the difficulty ranking is not the one that a human being would naturally assign to the data sets. The fourth and fifth data sets, with obvious, visible data partitions in the x-coordinate are, in some philosophical sense, the easiest. In spite of this, the one based on three parallel lines is the second hardest from the perspective of locating highly accurate models. In addition, the middle panel of Figure 10 shows a highly accurate solution that uses only two models to capture all the data, producing an inaccurate partition.

The distribution of fitness values for the five experiments with function stacks are shown in Figure 7. This figure does not contain subtle information; it does show that the five data sets are very different as optimization problems, reinforcing the conclusions reached by looking at Table II. This strongly suggests that the fitness landscapes may be very different which, in turn, opens up new direction for research.

The only data set containing a non-linear data source, data set three with the parabola, was the one for which it was easiest to locate highly accurate models. Examining Figure 9 we see that in addition to highly accurate models that discovered the formulas used to generate the data, we get a partition into two sets that does not respect the data generation method. At the points where the parabola intersects the line there is a change

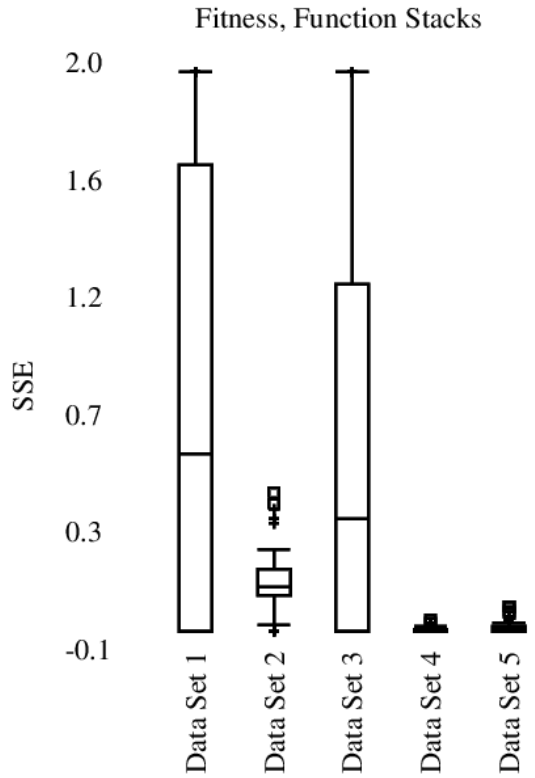

Fig. 7. Shown are box plots of the 100 fitness values for each experiment performed with function stacks.

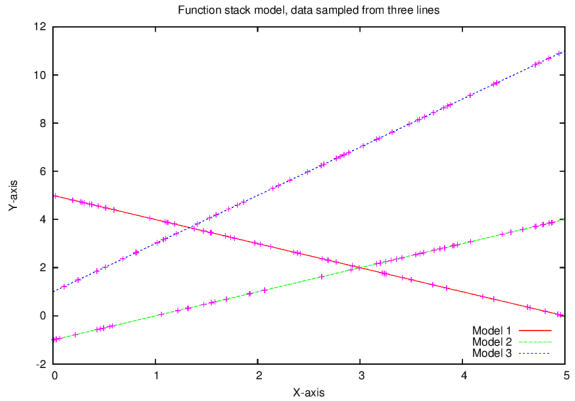

Fig. 8. Results using function stacks for partitioning regression on the second data set with three lines.

from the linear to the parabolic data source at both ends. This is not and incorrect partition, and it is in fact reassuring in that it shows that the system can do more than discover the embedded data model.

Two of the data sets, the second and fifth, used three lines with distinct slopes. The first, where these lines form a triangle in the center of the data arena, was the hardest to model, yielding only three highly accurate models. The other, similar except that the triangle formed by the lines was not even entirely contained in the data area, was the second easiest to model. The question of what makes a data set hard of easy to

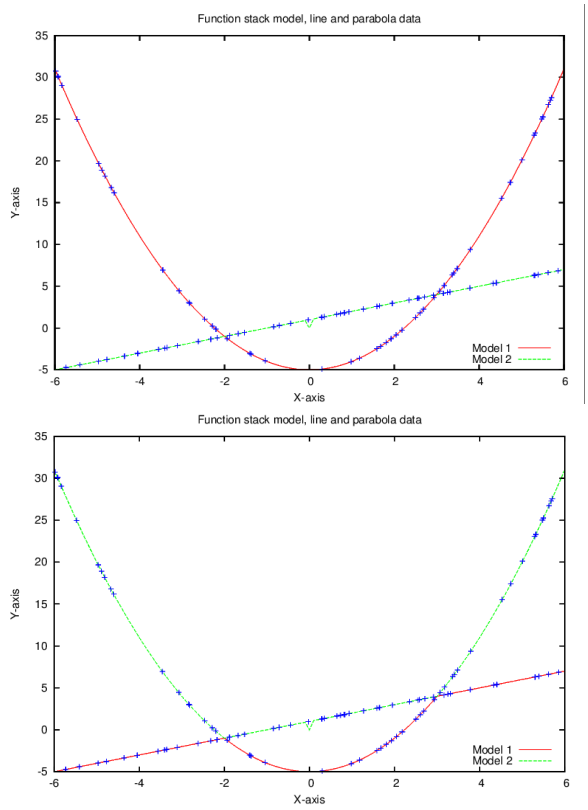

Fig. 9. Results using function stacks for partitioning regression on the third data set with a line and a parabola. The upper panel shows a solution that correctly partitions the data relative to the technique used to generate it, the lower panel shows a solution that partition the data in an interesting alternate fashion.

model is one that deserves revisiting.

The incorrect model in the middle panel of Figure 10 is very interesting. Model one, which captures two of the data sources used to generate the data, is model 2 (which captures nothing) plus a linear trend. This suggests that the decision to pull the data models from nodes 0, 1, and two of the function stack could perhaps be profitably revisited.

V. Conclusions and Next Steps

This study provides proof-of-concept for the idea of performing partitioning regression both model and partition data into reasonable classes. The partitioning of the data is a form of clustering, and clustering is always an exploratory activity. The ability of the system to discover multiple different ways to partition, exemplified by the lower panel of Figure 9 shows that the system presented in this paper does perform in an exploratory fashion.

One thing that is an issue is that the system sometimes manages to cobble together one model that cobbles together a function that yields high accuracy on two of the embedded data categories. In the middle panel Figure 10 this was done with an arctangent function added to a linear trend that put both its asymptotes through two parallel data classes. Figure 11 shows a way this can happen when the lines are not parallel: a split rule line is used to capture two of the three data categories.

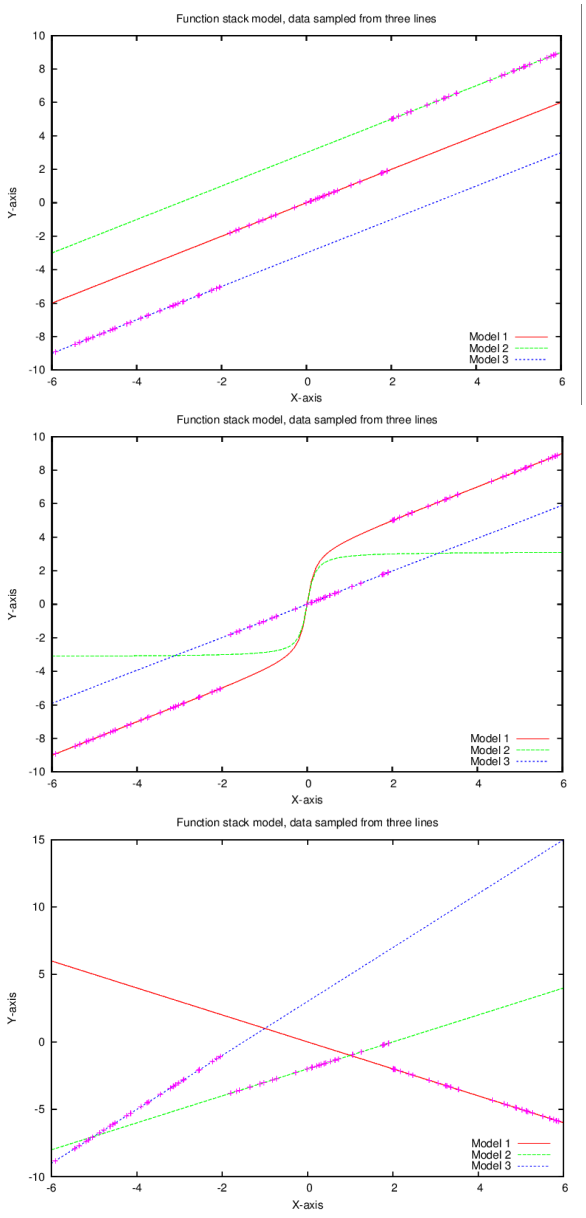

Fig. 10. Results using function stacks for partitioning regression on the forth and fifth data sets with three lines that are easily segmented along the x-axis. The first and third panels show correctly partitions the data relative to the technique used to generate it. The middle panel shows an incorrect, but interesting, solution with two partitions.

The *Tartarus* virtual robotics task [17] asks a bulldozer in a 6x6 grid world to push six boxes into the corner and against the walls. A master's project (never published) many years ago modified the task to use two bulldozers and tried to evolve cooperative behavior. This project failed in an interesting and informative way: one of the two robots would head for a corner. If it caught a box on the way, great, but either way is would simply go to the corner and turn back and forth. The second robot would then try and do all the work. We called this the "student project team" model of cooperation.

WModern teaching has caught up with the problem implied

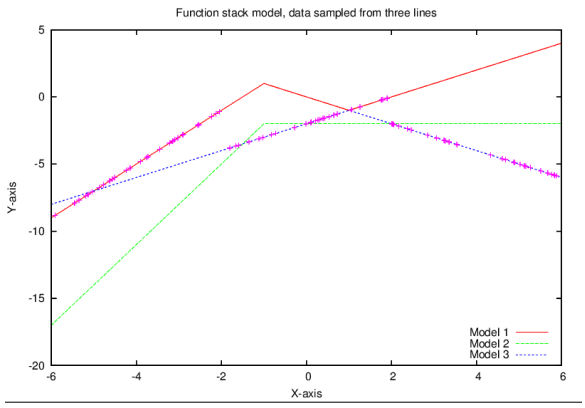

Fig. 11. Shown is a results using function stacks for partitioning regression on the fifth data set that models the date with high accuracy using only two models.

by the name for the local optima of robot behavior mentioned in the last paragraph. this seems not to be so for evolved models: for the three parallel line data set the capture of two of the embedded data models by one evolved model was more common than correct solutions. This, in turn, suggests that something like "student project team" optima may exist in the function-stack based partitioning regression system. In the discussion of the results for three parallel line data sources, it was noted that the one-model-captures-two effect might be, in part, due to the adjacency in the function stack of the nodes tapped to provide the models. This in turn suggests that experimenting with the placement of the function stack nodes used as models, or permitting them to evolve is a good idea.

It is important to note that these once-captures-two optima are not "wrong" per se. Many of them obtain excellent BSE fitness. What they fail to do is capture the partitions embedded when the data were generated. This effect may be much less noticeable in noisy situations, e.g. applications to real world measured data.

A. Partitioning Regression as Clustering

One of the great advantages of symbolic regression over standard regression is that standard regression require that a model be selected, while symbolic regression uses genetic programming to simultaneously selected and fit a model. The analysis of the system presented in this paper has, thus far, concentrated on the ability of the system to recover implanted partitions of the data. Since applications data will not have implanted partitions and may well support multiple clustering principles that inform the researcher about the data in different ways, a genetic programming based clustering methods is a way to being the advantages of not pre-selecting models to clustering.

In [8] a technique called *multi-clustering* is presented. It creates ensemble clusterings of many different instances of k-means clustering [12], [11] to create a clustering that is both more defensible than a single instance of k-means and also less influenced by the distance model used to perform the k-means computations. Partitioning regression exhibits many of the same model-independence properties as k-means. It can also be used, in place of k-means as a foundation for multiclustering.

B. Analysis with Residuals

The *residual* is the difference between a model and the data. The BSE fitness function sums the minimum of the squared residuals for each data point. Suppose that we are performing partitioning regression with m models on a data set. Then, for each point, we get m residuals. If we transform the data into m-vectors of residuals, even a very basic algorithm like k-means should be able to discover the clusters. The fact that members of a data partition will have a strong agreement in one coordinate means that clusterings of the residual vectors have a good chance of recovering the cluster structure. The residual vector can be used to wash out the geometry of the models, leaving data that are easier to work with. The models used in residual analysis need not be drawn from the same function stack - an analysis using *agent case embeddings* [1] could be used to select a diverse collection of models from those evolved for performing this type of analysis.

C. Tuning the Algorithm

The experiments performed in this study used the default values for evolving function stacks (as single model regressors) set during earlier research. While these parameters have been reported, they have not been tuned. The reason for this is that the proof-of-concept for partitioning regression was accomplished with the default values, even though the task had been changed. A parameter tuning study is an early priority for future research.

In addition to the standard parameters for evolutionary computation systems and function stacks, such as population size, mutation rate, crossover rate, and number of nodes in the function stack, some novel parameters need to be examined. These include placement of the output nodes for models in function stack - there is some evidence in the current study that "adjacent" may not be the best choice - as well as checking the impact of using too many models, initially, with an *extinction* operator that removes from consideration models that are capturing few points.

D. Possible Innovations

A natural innovation is to augment the function stack representation to designate its own model outputs. This would require a very small expansion of the data structure while permitting discovery of better placements for model output that the adjacent placement used in the current study. When two models share large parts of their mathematical formula, adjacent placement is potentially an effective choice - when there is little "code sharing" between different models, more distant placement of the output nodes seems more sensible. This argues strongly for permitting evolution to place the models when the character of the data is not known *a priori*.

In [2], [4], [3], the multiple worlds model for partitioning regression [5] operates using a version of extinction. In

essence, a model that is capturing little or none of the data is removed from consideration. Since some computation is need for each model to find which model has the best error on a point, there is a savings in eliminating models. This benefit is in addition to using extinction to suggest number of models for a given data set. When juxtaposed with the idea of evolutionary placement of model output nodes, extinction becomes even more valuable. If the good placements are a subset of an initial, too large, set of model outputs then the discovery of good placements by evolution becomes easier.

The data used in this study was clean (no noise) with planted partitions of the data. If noise is added, some preliminary work shows that correct partitioning still occurs but, necessarily, the ability to discover recovery of the models used to plant the data partitions by looking for tiny error values vanishes. Another natural direction for future research is to characterize the reaction of the system to the addition of noise, a critical factor for applications.

Finally, this study used univariate data. This is because the results are easy to analyze and visualize. The evolutionary system used is already able to be run on higher dimensional data. This too needs to be part of any additional work with this system.

REFERENCES

[1] D. Ashlock and C. Lee. Agent-case embeddings for the analysis of evolved systems. *IEEE Transaction on Evolutionary Computation*, 17(2):227–240, 2013.

[2] J. A. Brown. Multiple worlds evolution for motif discovery. In *Proceedings of the 2012 IEEE Symposium on Bioinformatics and Computational Biology*, page 9299, 2012.

[3] J. A. Brown. Examination of graphs in multiple agent genetic networks for iterated prisoner's dilemma. In *Proceedings of the IEEE Conference on Computational Intelligence in Games*, pages 314–321, 2013.

[4] J. A. Brown. More multiple worlds evolution for motif discovery. In *Proceedings of the 2013 IEEE Symposium on Bioinformatics and Computational Biology*, pages 168–175, 2013.

[5] J. A. Brown. *Regression and Classification from Extinction*. PhD thesis, University of Guelph, 2014.

[6] N. Q. Uy *et. al.* Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119, 2010.

[7] Maarten Keijzer. *Genetic Programming: 6th European Conference, EuroGP 2003 Essex, UK, April 14–16, 2003 Proceedings*, chapter Improving Symbolic Regression with Interval Arithmetic and Linear Scaling, pages 70–82. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[8] E.Y. Kim, S.Y. Kim, D. Ashlock, and D. Nam. Multi-k: accurate classification of microarray subtypes using ensemble k-means clustering. *BMC Bioinformatics*, 260(10):1–12, 2009.

[9] John R. Koza. *Genetic Programming*. The MIT Press, Cambridge, MA, 1992.

[10] John R. Koza. *Genetic Programming II*. The MIT Press, Cambridge, MA, 1994.

[11] S. P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[12] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. Fifth Berkeley Symp. on Math. Statist. and Prob.*, volume 1, pages 281–297, Berkeley, CA, 1967. University of California Press.

[13] J. F. Miller and S. L. Smith. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, 10(2):167–174, 2006.

[14] Julian F. Miller and Peter Thomson. *Genetic Programming: European Conference, EuroGP 2000, Edinburgh, Scotland, UK, April 15-16, 2000. Proceedings*, chapter Cartesian Genetic Programming, pages 121–132. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[15] D. P. Searson, D. E. Leahy, and M. J. Wells. Gptips:an open source genetic programming toolbox for multigene symbolic regression. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, pages 1–4, 2010.

[16] Xiong Shengwu, Wang Weiwu, and Li Feng. A new genetic programming approach in symbolic regression. In *Tools with Artificial Intelligence, 2003. Proceedings. 15th IEEE International Conference on*, pages 161–165, Nov 2003.

[17] Astro Teller. The evolution of mental models. In Kenneth Kinnear, editor, *Advances in Genetic Programming*, chapter 9. The MIT Press, 1994.