Modeling Hierarchy using Symbolic Regression

Ilknur Icke
Morphology, Evolution and Cognition Lab.
Department of Computer Science
University of Vermont
Burlington, VT 05401
ilknur.icke@uvm.edu

Joshua C. Bongard
Morphology, Evolution and Cognition Lab.
Department of Computer Science
University of Vermont
Burlington, VT 05401
jbongard@uvm.edu

***Abstract*—Symbolic Regression is an attractive modeling approach because it can capture and present, mathematically, relationships between variables of interest. However, given n variables to model, symbolic regression returns a flat list of n equations. As the number of state variables to be modeled scales, interpretation of such a list becomes difficult. Here we present a symbolic regression method that detects and captures hidden hierarchy in a given system. The method returns the equations in a hierarchical dependency graph, which increases the interpretability of the results. We demonstrate that two variations of this hierarchical modeling approach outperform non-hierarchical symbolic regression on a synthetic data suite.**

***Index Terms*—hierarchy, dependency graph, data mining**

I. Introduction

Hierarchical relationships abound in natural and man-made systems. Hierarchy is thought to be a fundamental characteristic of many complex systems such as biological organisms [1], ecological systems [2], the Internet, and traffic networks [3] and, arguably, social organizations [4]. The human visual system is known to be organized hierarchically [5], where the lower level components process the sensory stimuli and the higher levels process the output of the lower level components. Many artificial neural network architectures used for pattern recognition tasks were also designed based on this principle [6]. More recently, deep belief networks [7] attempt to discover the hierarchical structure hidden in large data sets by learning several layers of hierarchically organized features. These natural/artificial hierarchical systems have been evolved/trained to be able to respond to a wide variety of stimuli. In this scheme, each component is specialized in processing a subset of the inputs coming from the lower level.

Our goal is to be able to automatically reverse engineer hierarchical systems in order to understand which inputs each component is processing and uncover the nature of the process (i.e how each component is computing its output from its inputs). In this paper, we focus on systems where the inputs of the individual components are not overlapping (Fig. 1) such as the non-overlapping perceptron or biological neural networks with non-overlapping receptive fields [8], [9].

Here we show that if traditional symbolic regression is applied (in which each variable is modeled separately but allowed to be described as a function of every other variable) little progress is made on increasingly large yet hierarchical

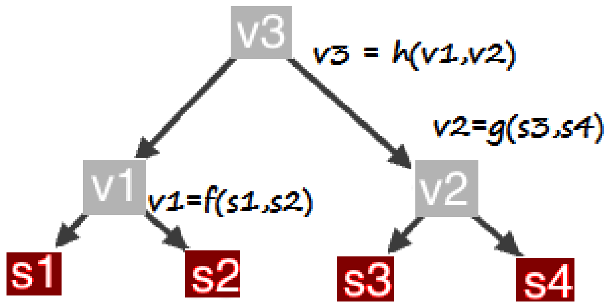


Fig. 1: Dependency graph for a hierarchical system with non-overlapping inputs. The leaf nodes are the stimuli (controlled) and internal nodes are the state variables whose behaviors are observed. The direction of the arrows indicate dependency that is opposite to the information flow in the system.

target systems. This is because as the number of variables increases, more independent runs must be performed (one for each variable), and each run is more difficult because there are more variables to make use of. In contrast we present two variations of symbolic regression adapted for hierarchical systems: the variables that are directly influenced by variables at the lowest level of the hierarchy are identified and modeled first, followed by the variables at the next-highest level but which are restricted to using the variables on the layer below them. We find that hierarchical approach significantly outperforms the traditional symbolic regression paradigm on a number of synthetic datasets that vary in difficulty.

This paper is organized as follows: section II presents the problem in terms of a motivating example and section III discusses the related work. Our proposed approaches for automatic identification of hierarchy are presented in section IV. Experimental results are presented in section V. Finally, conclusions and future work are discussed in section VI.

II. Identifying Hierarchy

Many systems are made up of multiple components that interact with each other by receiving inputs from and/or transmitting outputs to other components. In some cases, these components are hierarchically organized where the information flows in a bottom up manner. In a hierarchical organization, the output of each component depends on the inputs it receives from the components at the lower levels.

Fig. 1 shows a very simple hierarchical system that receives 4 stimuli (s_1, s_2, s_3, s_4). The stimuli are then processed by the two components (v_1 and v_2) and their outputs are passed on to the top component v_3. In this system, the stimuli are provided by the environment (or controlled by an experimenter) and the behavior of v_1, v_2 and v_3 are observed. It is important to note that, only the stimuli are known and the responses of v_1, v_2 and v_3 to these external inputs are observed, but the actual connectivity between the components are not known. The goal is to identify which components are connected and how they are connected.

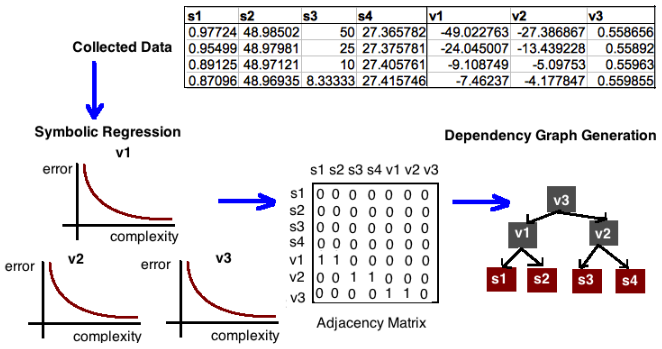
In order to understand the nature of the relationships between the inputs and outputs of each component (the functions f, g and h in Fig. 1), one would employ a regression approach. In this regard, being a free-form modeling technique, symbolic regression can be seen as a more flexible approach as opposed to the mainstream linear regression techniques.

The algorithms we present in this paper are built around a genetic programming based implementation of symbolic regression (SR) in order to model the relationships between the components of an hierarchical system. In symbolic regression, the most predictive variables will appear in the evolved expressions eliminating the non-informative variables. In the context of this paper, if a variable is predictive of another, we say that the predicted variable depends on it. The nature of this dependency can be further examined by looking at the evolved expression relating the predicted variable to its predictors.

III. Related Work

A genetic programming based method for modeling the ODEs for gene regulatory networks was presented in [10]. The algorithm independently evolves expressions to explain each state variable on the observed time series data and does not explicitly model hierarchy. This work was followed by several other papers that produced flat lists of ODEs when exposed to multivariate datasets [11], [12]. A linear genetic programming based reverse engineering algorithm for neuronal networks was presented in [13]. Starting from the observed data, the algorithm tries to infer the structure and parameters of the system. Each state variable is evolved with respect to the neuroscience domain knowledge that is built into the algorithm which limits the use of the algorithm beyond that specific problem domain.

The idea of building variable interaction networks from multivariate data was explored in several papers. An algorithm that extracts a linear dependency structure from multivariate data without explicitly modeling hierarchy was reported in [14]. The limitation of the algorithm is that it uses linear regression in order to construct a linear dependency tree or forest of the variables. Linear models are easier to build and they are intuitive, however there is no guarantee that the phenomenon under study is governed by linear relationships. In [15], multiple genetic programming based symbolic regression runs are executed for each variable separately and a variable interaction network is built by identifying the most relevant variables for a given target variable in terms of a



s1	s2	s3	s4	v1	v2	v3
0.97724	48.98502	50	27.365782	-49.022763	-27.386867	0.558656
0.95499	48.97981	25	27.375781	-24.045007	-13.439228	0.55892
0.89125	48.97121	10	27.405761	-9.108749	-5.09753	0.55963
0.87096	48.96935	8.33333	27.415746	-7.46237	-4.177847	0.559855

$$\begin{array}{c|ccccccc} & s1 & s2 & s3 & s4 & v1 & v2 & v3 \\ \hline s1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ s2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ s3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ s4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ v1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ v2 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ v3 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}$$

Fig. 2: The workflow of the NSR algorithm

measure of relative frequencies of variable appearances in the expressions modeling that target variable. The algorithm does not assume any specific network topology such as hierarchy.

IV. Symbolic Regression Approach to Model Hierarchy in Multivariate Data

In this section, we present three approaches to automatically extract the hierarchical relationships in multivariate data. The Naive SR algorithm models each state (non-stimuli) variable separately using symbolic regression. After the run is completed, the hierarchy is extracted by examining the expressions and identifying which variables depend on each other. The other two approaches are iterative that aim to enforce hierarchy during the search for the optimal symbolic expressions.

A. The Naive SR

In identifying the relationships between multiple variables, a straight-forward approach is to model each variable separately in terms of all other variables using symbolic regression. In doing this, one would expect that the best models would reveal the most informative (highly predictive) variables for each modeled variable. After the symbolic regression phase is done, constructing the hierarchy is just a matter of post-processing. Because each variable is modeled independently, the algorithm does not impose any constraints on the connectivity. The workflow of the Naive SR is presented in Fig. 2 and the steps of the method are outlined in algorithm 1.

After each non-stimulus variable is modeled separately on the training dataset using symbolic regression (the evolve phase), the post-processing phase begins. At this stage, the set of all non-dominated models for each variable are evaluated on the validation dataset. Then, the best model for each variable is chosen (line 8 of algorithm 1) as the model with the lowest error on the validation dataset. The ties are broken in favor of the simplest model. Following the selection of the best models, the variables appearing in these models are identified as the predictors for each respective modeled variable (line 10). An adjacency matrix is then built (line 13) based on these identified predicted variable-predictor mappings. Finally, the algorithm returns the adjacency matrix representing the connectivity between the variables along with the set of best models evolved for each non-stimulus variable.

Algorithm 1: Naive SR (NSR) Algorithm

```
Input: V={$v_1$,$v_2$, ..., $v_N$}
Output: Dependency graph/forest G and a set of
        expressions E={$v_i$=$f_i(\mathbf{v} - \{\mathbf{v_i}\})$}
1  while time budget not exceeded do
2      foreach $v_i$ do
3          Evolve $v_i$ = $\mathbf{f}_i$(V- {$\mathbf{v}_i$}) on training set
4      end
5  end
6  E={}, D={ }
7  foreach $v_i$ do
8      $g_i$=FindBestOnParetoFront($\mathbf{f}_i$) on validation set
9      E=E ∪ {$g_i$}
10     $\mathbf{d}_i$ =ExtractPredictorSet($g_i$)
11     D = D ∪ {$\mathbf{d}_i$}
12 end
13 G=BuildGraph(V,D)
```

B. *Hierarchical SR version 1 (h1)*

In this method, we enforce hierarchical extraction of the dependencies in an iterative manner. At each iteration, dependencies for one non-stimulus variable are discovered. The algorithm starts with only the stimuli as the set of available independent variables. After first iteration, the variable (v_i) that is best explained by a subset of these inputs is determined. Then, all predictors for variable (v_i) are removed from the set of available independent variables in accordance with our constraint that inputs can not overlap. Next, v_i is added to the list of independent variables. The algorithm stops after each (non-stimulus) variable has been modeled using symbolic regression. The method is outlined in algorithm 2.

As opposed to the naive algorithm, the HSR version 1 works in epochs. At the end of each epoch, one variable that is modeled the best is selected and eliminated from the set of dependent variables. This selection is done by identifying one best model across all models for all variables (line 12 of algorithm 2). For N dependent variables, there are N non-dominated sets at the end of each epoch. A combined non-dominated set is then generated in terms of validation set error and the expression complexity (Fig. 3). Throughout this paper, expression complexity is computed as the number of nodes in the expression tree.

Once the best model on the combined non-dominated set is selected (the model with the lowest validation data error), the corresponding dependent variable and its predictors are identified. Since these predictors can only be used to model the identified dependent variable due to our non-overlapping inputs assumption, they are removed from the list of possible inputs for modeling other variables. The identified dependent variable is added to the list of possible predictors for other variables that are waiting to be modeled. This process actively enforces the extraction of hierarchical relationships and generates a set of easily interpretable expressions instead of a flat list of unstructured expressions.

Algorithm 2: Hierarchical SR (HSR) Algorithm v.1

```
Input: V={$v_1$,$v_2$, ..., $v_N$}
Output: Dependency graph G and a set of expressions
        E={$v_i$=$f_i(\mathbf{v} - \{\mathbf{v_i}\})$}
1  E={}
2  I={$s_1$,$s_2$,...$s_N$}
3  C=V, D={ }
4  NumEpochs=#Dependent Variables
5  timeBudget = totalTimeBudget/NumEpochs
6  while not all dependent variables are modeled do
7      while time budget not exceeded do
8          foreach $v_i$ do
9              Evolve $v_i$ = $\mathbf{f}_i$(V- {$\mathbf{v}_i$}) on training set
10         end
11     end
12     $g_i$=FindBestModel($\mathbf{f}_i$) on validation set
13     E=E ∪ {$g_i$}
14     $\mathbf{d}_i$ =ExtractPredictorSet($g_i$)
15     D = D ∪ {$\mathbf{d}_i$}
16     C = C - {$v_i$}
17     I= I - {$\mathbf{d}_i$ }
18     I =I ∪ {$v_i$}
19 end
20 G=BuildGraph(V,D)
```

Reconsidering the motivating example in Fig. 1, it is easy to see that the top-level component v_3 can also be modeled as $v_3 = h'(s_1, s_2, s_3, s_4)$. However, such an expression will be dominated by the less complex but equally fit expressions for v_1 and v_2 upon combining all non-dominated sets as in Fig. 3.

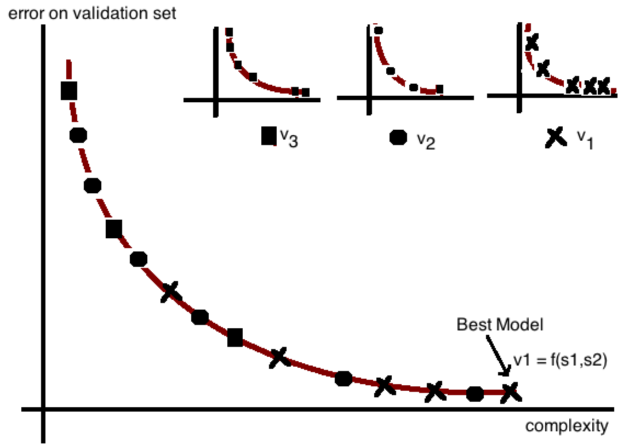


Fig. 3: Selection of the best model on the combined non-dominated set in HSR version 1

C. *Hierarchical SR version 2 (h2)*

The first version of the hierarchical SR algorithm pools all evolved model in an epoch and makes the selection based on the best model on the validation data set. In the second version of the hierarchical algorithm, we modify this selection strategy. Instead of identifying one best model across all variables and then extracting the predictors, we try rather the opposite approach. We first identify the best non-dominated set and then find the set of most frequently occurring predictors across all non-dominated models for the corresponding modeled variable. An additional evolve step is performed in order to find the best expression based on the identified predictor variables only.

The reasoning behind this strategy is that making use of the statistics about how the current set of dependent variables are used across the Pareto front, rather than just how those variables are used in the model with lowest error, might improve its performance compared to h1. The method is outlined in algorithm 3. Similar to the h1 algorithm, the h2 algorithm also runs in epochs. The only difference is in the selection of the best model and the predictor variables at the end of each epoch.

Algorithm 3: Hierarchical SR (HSR) Algorithm v.2

```
Input: V={$v_1$,$v_2$, ..., $v_N$}
Output: Dependency graph G and a set of expressions
        E={$v_i$=$f_i(\mathbf{v} - \{\mathbf{v_i}\})$}
1  E={}
2  I={$s_1$,$s_2$,...$s_N$}
3  C=V, D={ }
4  NumEpochs=#Dependent Variables
5  timeBudget = totalTimeBudget/NumEpochs
6  while not all dependent variables are modeled do
7      while time budget not exceeded do
8          foreach $v_i$ do
9              $v_i$ =Evolve( $\mathbf{f}_i$(V- {$\mathbf{v}_i$}) ) on training set
10         end
11     end
12     $b_i$=FindBestModeledVariable($\mathbf{v}_i$) on validation set
13     $\mathbf{d}_i$ =ExtractPredictorSet($b_i$)
14     $b_i$= Evolve( $\mathbf{f}_i$({$\mathbf{d}_i$}) ) on training set
15     $g_i$=FindBestModel($\mathbf{f}_i$) on validation set
16     E=E ∪ {$g_i$}
17     D = D ∪ {$\mathbf{d}_i$}
18     C = C - {$b_i$}
19     I= I - {$\mathbf{d}_i$ }
20     I =I ∪ {$b_i$}
21 end
22 G=BuildGraph(V,D)
```

The selection process is summarized in Fig. 4. For each modeled variable, all non-dominated models are evaluated on the validation dataset and new non-dominated sets based on the validation dataset error versus expression complexity are built (line 12 of algorithm 3). For each non-dominated set (ns), the fitness is computed as the weighted sum of the error on validation dataset:

$$fitness(ns) = \sum_{i=1}^{M} error * complexity$$

The non-dominated set with lowest weighted error is selected for further processing. Ties are broken in favor of the smallest total expression complexity. Again, reconsidering the motivating example in Fig. 1, where v_3 can also be modeled as $v_3 = h'(s_1, s_2, s_3, s_4)$, the weighted error fitness will penalize v_3 because of the higher total complexity of the expressions.

The next step considers the models that are in the non-dominated set for the selected variable only. A histogram of unique occurrences of each predictor is generated and the set of most frequently occurring predictors are selected by identifying the cut-off point on the histogram. The final model is then generated via symbolic regression using only the selected predictors as the terminal symbols (lines 14-15).

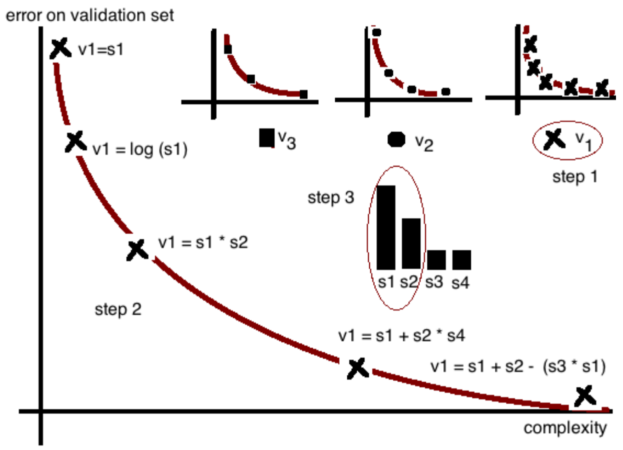


Fig. 4: Selection of the best variable in HSR version 2 (line 12 of algorithm 3). Selection of the best non-dominated set (step 1), building the predictor-frequency histogram (step 2), selecting the most frequently occurring predictors (step 3).

V. Experimental Results

In this section, we first discuss the conceptual differences between the naive way of modeling hierarchical relationships in multivariate data versus actively enforcing hierarchical modeling on an example dataset. Then, we outline our experimental procedures in generating a large benchmark synthetic data suite with varying levels of difficulty and comparing the three algorithms. As it is stated in [16], challenging benchmark datasets are needed for genetic programming research. Ideally, applying algorithms to real-world data is preferable. However, especially in the case of testing new algorithms, we believe that it is very important to have control over the data generation process so that analysis of the strengths and weaknesses of the algorithms can be easier. Finally, we present and discuss our findings on these synthetic benchmark datasets.

A. An example 16-Input Binary Tree System

A simple 16-input synthetic system has been generated using the following expressions representing the relationships between the variables:

$$
\begin{aligned}
&layer1: \\
&\quad v_1 = s_1 + s_2 \\
&\quad v_2 = s_3 - s_4 \\
&\quad v_3 = s_5 + s_6 \\
&\quad v_4 = s_7 + s_8 \\
&\quad v_5 = s_9 + s_{10} \\
&\quad v_6 = s_{11} + s_{12} \\
&\quad v_7 = s_{13} + s_{14} \\
&\quad v_8 = s_{15} + s_{16} \\
&layer2: \\
&\quad v_9 = v_1 / v_2 \\
&\quad v_{10} = v_3 * v_4 \\
&\quad v_{v11} = v_5 - v_6 \\
&\quad v_{v12} = v_7 + v_8 \\
&layer3: \\
&\quad v_{13} = v_9 / v_{10} \\
&\quad v_{v14} = v_{11} - v_{12} \\
&layer4: \\
&\quad v_{15} = v_{14} + v_{13}
\end{aligned}
$$

The resulting dependency graph is shown in Fig. 5. The binary tree (arity=2) consists of 4 layers, 16 stimuli and 15 internal nodes which are the variables to be modeled. The total number of edges in the tree is $arity * internal\ nodes = 30$.

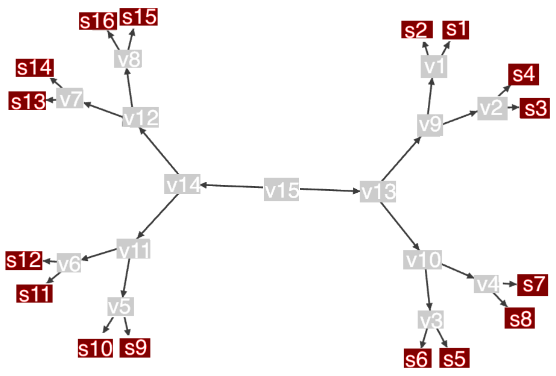
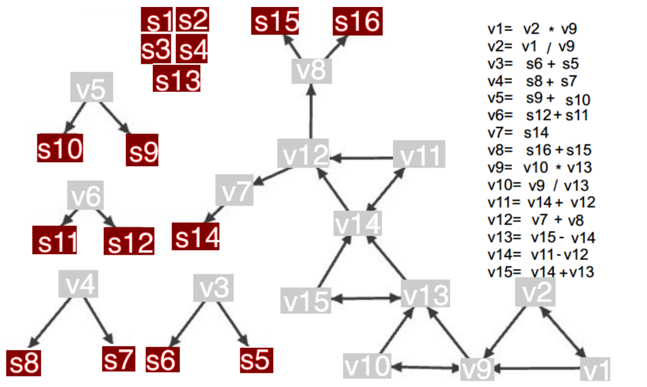


Fig. 5: True dependency graph for the synthetic 16-input binary tree system

Fig. 6 shows the best dependency graph generated by the naive SR algorithm in terms of the error on the test dataset. The test error is calculated as the average error across all modeled variables. Despite the low error, the constructed dependency graph is very dissimilar to the original graph shown in Fig. 5. A closer look at the constructed graph and the generated models for the variables reveals many redundant and cyclic dependencies as well as a number of ignored stimuli. This example shows that even though the given dataset is perfectly hierarchical, the NSR algorithm might fail to capture this hierarchy. Therefore, solely minimizing the prediction error without any constraints on the connectivity might be deceptive for the purposes of modeling hierarchical systems. On the other hand, when multiple runs are performed, it was possible for the algorithm to find the correct dependency graph structure in some of the runs along with the lowest possible test dataset error. However, for this algorithm, a low test error does not always mean that the hierarchy in the underlying system is captured.



v1= v2 * v9
v2= v1 / v9
v3= s6 + s5
v4= s8 + s7
v5= s9 + s10
v6= s12 + s11
v7= s14
v8= s16 + s15
v9= v10 * v13
v10= v9 / v13
v11= v14 + v12
v12= v7 + v8
v13= v15 - v14
v14= v11 - v12
v15= v14 + v13

Fig. 6: Dependency graph with lowest test set error (rmse: 0.025) generated by NSR

Fig. 7 shows the dependency graph with the highest test set error generated using the HSR version 1 (h1) algorithm. In terms of the prediction accuracy, this system is worse than the one discovered using the naive approach (Fig. 6). However, the constructed graph almost perfectly captures the true system. The high prediction error was caused by just one misplaced edge (s_{15} is erroneously tied to v_6 instead of v_8). Therefore, for h1 and h2 algorithms, a high test set error does not always mean failure in terms of how close the hierarchy is captured.

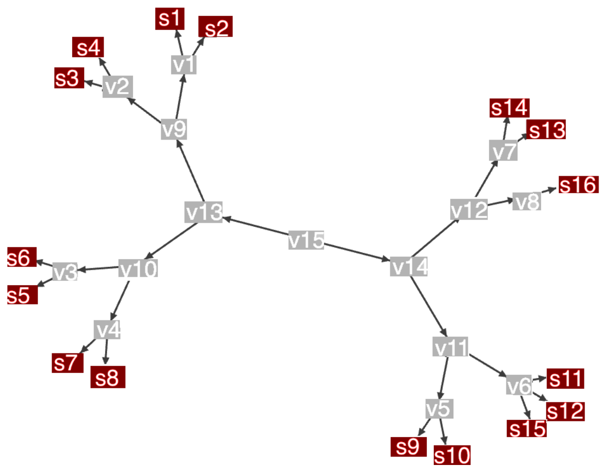


Fig. 7: Dependency graph with highest test set error (rmse: 1.662) generated by HSR version 1

B. Synthetic Benchmark Problems

In order to study the behavior of the algorithms across multiple datasets with varying difficulty, we generated 30 synthetic datasets for each combination of arity=2,3,4,5 and layers=3,4,5,6. A total of 480 datasets were generated with random stimuli as inputs to the systems and randomly generated expressions to represent the functions of the state variables. For the sake of simplicity, the expressions included only {+,-,* and protected / } operators without any constant values. We also kept the degree of nonlinearity constant across all datasets by enforcing that only binary nonlinear interactions were allowed in the randomly generated expressions for the state variables. For instance, for an arity-5 system, the hidden expression for a state variable can be $v_1 = s_1 + s_2 - s_4 * s_5 - s_3$, but not $v_1 = s_1 * s_2 / s_4 * s_5 - s_3$. By design, the difficulty of the dataset increases as the data dimensionality increases (between 4 - 3125 inputs).

TABLE I: Number of inputs, variables to be modeled and number of edges for the generated synthetic benchmark trees

		Layers			
		3	4	5	6
Arity	2	4, 3, 6	8, 7, 14	16, 15, 30	32, 31 ,62
	3	9, 4, 12	27, 13, 39	81, 40, 120	243, 121, 363
	4	16, 5, 20	64, 21, 84	256, 85, 340	1024, 341, 1364
	5	25, 6, 30	125, 31, 155	625, 156,789	3125, 781, 3905

Each dataset was divided into training, validation and test partitions as follows: for each n-arity tree system, all expressions for the state variables (internal nodes of the tree) were evaluated for 5000 randomly generated stimuli creating a 5000x($(arity^{layers} - 1)/(arity - 1)$) dataset. For every 4 rows, the first two rows were included in the training set, while the third and fourth rows are included in the validation and test sets respectively. The training,validation and testing partitions consisted of 2500, 1250 and 1250 rows each.

C. Results on Benchmark Problems

We ran each algorithm 30 times on all 480 datasets for a run time budget of 10 minutes per run for a total of 43200 runs on a cluster computing environment. All algorithms were implemented in C++. The baseline symbolic regression implementation that is used by all three algorithms utilizes the standard tree based representation with sub-tree crossover and mutation operators. Fixed values for population size (500), crossover probability (0.9) and mutation probability (0.1) were used across all experiments. Root mean squared error (rmse) versus expression complexity trade-off along with the age of the individuals were used as the multi-objective fitness function. Similar to the AFPO algorithm [17], a new random individual is added to the population at the beginning of each generation. For n state variables to be modeled, the population included n sub-populations, each of which were set up to evolve expressions for one state variable.

We report the results for each tree structure separately in Fig. 8. For each arity-layer pair, the results are pooled over 30 different trees and 30 runs for each tree resulting in a total of 900 runs per algorithm. In each case, the results are presented from three perspectives: the percentage of the edges that were correctly discovered, the prediction error of the generated tree on the test set, the distribution of test set error versus the percentage of the edges correctly discovered.

The statistical significance of the results are reported as follows: for the percentage of correct edges, we compare each pair of algorithms using the left-tailed Wilcoxon rank sum test with Bonferroni correction and unequal variances assumption. In those cases where the h1 and h2 algorithms are significantly better than the naive algorithm, and when the h2 algorithm is significantly better than the h1 algorithm, the significance is presented using $*$ sign ($***$:$\alpha = 0.001$, $**$:$\alpha = 0.01$, $*$:$\alpha = 0.05$). A similar comparison is performed on the test error results, the only difference being the Wilcoxon rank sum test to be a right-tailed test since lower test error indicates better performance in this case.

The top row shows the results for 2-arity tree systems with varying numbers of layers. In terms of capturing the hierarchy, h1 and h2 almost always discover the correct connectivity matrix and consistently outperform the naive algorithm. As far as the test error is concerned, h1 and h2 clearly outperform the naive algorithm only when the ratio of the total nodes to the stimuli gets too large for the naive algorithm to deal with. This happens when the height of the tree increases to 5. The heatmaps show that the naive algorithm mostly finds low-error models at the expense of missing many edges.

As the arity of the trees increase, the trees get broader, as a result, the number of leaves (stimuli) increases. Since this will increase the number of possible predictors, the search becomes more and more difficult for all three algorithms. Accordingly, the h1 and h2 algorithms start to lose their advantage in faithfully modeling the hierarchy as the arity increases. Another source of difficulty for all algorithms is the increase in tree height. As the trees get taller, the number of leaves (stimuli) also increases. Except for 2-arity systems (top row in Fig. 8), h1 and h2 failed to complete within the assigned run time budget in the case of the tallest trees.

D. Discussion

Our results on the synthetic benchmark datasets clearly show that for binary input problems (arity 2), the hierarchical SR algorithms consistently outperform the naive SR as the problem difficulty increases (more tree layers). However, the lack of scaling to much higher arity problems is due to a number of reasons. High data dimensionality is a big challenge for symbolic regression. In additional experiments (not shown here), increasing the time budget from 10 minutes to 1 hour did not significantly increase the performance of h1 and h2. Therefore, efficient feature selection schemes for high dimensional data within symbolic regression is definitely an area for improvement.

Specifically, the results indicate that as the trees get broader and taller, the initial h1 and h2 algorithms do not, in their current form, continue to outperform the naive approach. This is due to the fact that in these cases, the number of epochs significantly increases, which will in turn decrease the

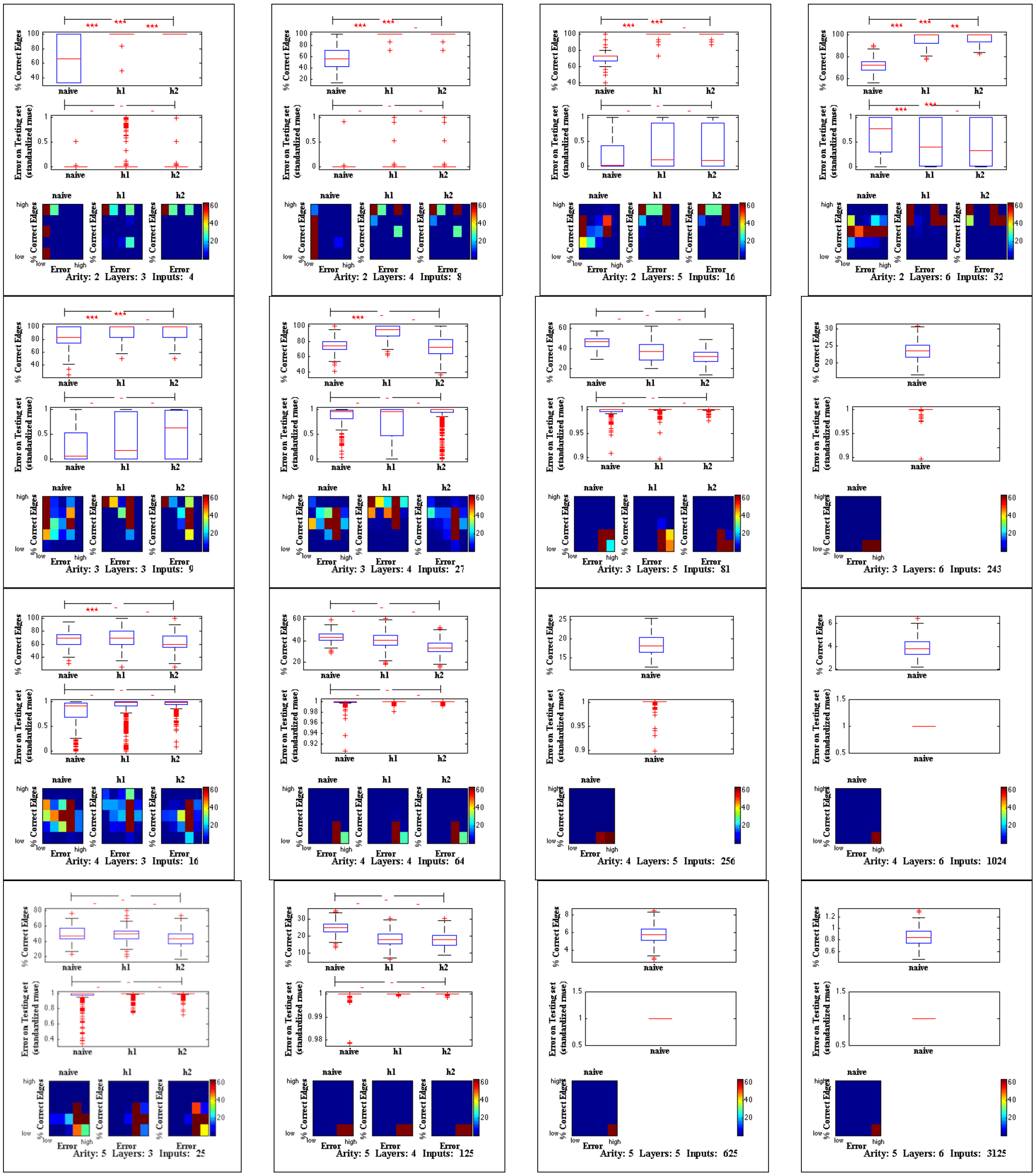


Fig. 8: Comparison of the algorithms across varying data dimensionality and hierarchical organization given 10 minutes runtime budget. Problem difficulty increases from left-right and top-bottom. For each problem type, the plots show the percentage of the edges that were correctly discovered (top), the prediction error of the generated tree on the test set (middle), the distribution of test set error versus the percentage of the edges correctly discovered (bottom). The naive approach mostly fails to recover the hierarchical network topology. The hierarchical approach outperforms the naive approach for easier problems (small arity and/or short trees). All three algorithms perform poorly for more difficult problems (larger arity and taller trees).

amount of time for each individual epoch. This increases the risk that h1 or h2 will select the wrong predictors. Indeed h2 was initially devised to reduce the risk of selecting the wrong dependent variables during an epoch. It was thought that providing h2 with statistics about how the current set of dependent variables are used across the Pareto front, rather than just how those variables are used in the model with lowest error, would improve its performance compared to h1. However this was not found to be the case: instead, h1 significantly outperformed the naive algorithm on seven of the tree structures (Fig. 8) while h2 only significantly outperformed the naive algorithm on five of them.

We hypothesize that h2 may be further improved by incorporating diversity maintaining measures that increase the size of the membership on the Pareto front. This should improve the reliability of the statistics computed across the front and thus improve the probability of selecting the correct set of dependent variables. We also plan to explore alternative methods for selecting the cutoff point in the predictor-frequency histogram. Additionally, in h2, predictor-frequency histograms constructed from previous epochs are discarded; in future work we will investigate ways to re-use information from previous histograms to produce more accurate histograms in the current epoch. Also, for both h1 and h2, selecting more than one accurately modeled variable in each epoch will speed up the algorithms by reducing the number of epochs.

Finally, we note that our initial implementations did not utilize any parallel and distributed techniques. Our algorithms have the potential to scale to real-world problems upon utilizing GPUs [18] and/or cloud computing [19].

VI. Conclusion

Extracting and visualizing the relationships in a hierarchical system as a dependency graph improves the intelligibility of the overall model, compared to the flat list of equations produced by traditional symbolic regression. Our results clearly show that in order to find hierarchy, one needs to explicitly search for it rather than waiting for the hierarchical models to emerge in an unconstrained search such as in the naive SR. Moreover, it was found that explicitly seeking hierarchy in a data set leads to more accurate models compared to traditional symbolic regression. These algorithms were tested against a large number of synthetic datasets with increasing difficulty in terms of data dimensionality and hierarchical organization. Even though the intuition suggests that the HSR version 2 (h2) algorithm would be more robust since it considers multiple models in making the selection for the best modeled variable, our experimental results on 480 synthetic datasets showed no clear advantage over the HSR version 1 (h1) which makes the selections based on one best model at each stage.

The focus of our current work is to further explore more efficient ways for the selection process at each stage and to extend the algorithm to model more general systems that exhibit mixtures of hierarchy and network connectivity. Ultimately, our goal is to apply our algorithms to real-world problems such as functional brain connectivity and gene expression networks.

Acknowledgment

This research was supported by a DARPA grant #FA8650-11-1-7155. The authors also acknowledge the Vermont Advanced Computing Core which is supported by NASA (NNX 06AC88G), at the University of Vermont for providing High Performance Computing resources that have contributed to the research results reported within this paper.

References

[1] R. J. Sommer, "Homology and the hierarchy of biological systems." *Bioessays*, vol. 30, no. 7, pp. 653–8, 2008.

[2] H. Hirata and R. E. Ulanowicz, "Information theoretical analysis of the aggregation and hierarchical structure of ecological networks," *Journal of Theoretical Biology*, vol. 116, no. 3, pp. 321 – 341, 1985.

[3] E. Ravasz and A.-L. Barabási, "Hierarchical organization in complex networks," *Phys. Rev. E*, vol. 67, p. 026112, Feb 2003.

[4] S. Buldyrev, N. Dokholyan, S. Erramilli, M. Hong, J. Kim, G. Malescio, and H. Stanley, "Hierarchy in social organization," *Physica A: Statistical Mechanics and its Applications*, vol. 330, no. 3-4, pp. 653 – 659, 2003.

[5] S. J. Kiebel, J. Daunizeau, and K. J. Friston, "Perception and hierarchical dynamics," *Frontiers in Neuroinformatics*, vol. 3, no. 20, 2009.

[6] M. Kaiser, C. C. Hilgetag, and R. Kötter, "Hierarchy and dynamics of neural networks," *Frontiers in Neuroinformatics*, vol. 4, no. 112, 2010.

[7] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th Annual International Conference on Machine Learning,ICML '09*, 2009, pp. 609–616.

[8] T. R. Hancock, M. Golea, and M. Marchand, "Learning nonoverlapping perceptron networks from examples and membership queries," *Mach. Learn.*, vol. 16, no. 3, pp. 161–183, Sep. 1994.

[9] M. Schmitt, "On the sample complexity for nonoverlapping neural networks," *Mach. Learn.*, vol. 37, no. 2, pp. 131–141, Nov. 1999.

[10] E. Sakamoto and H. Iba, "Inferring a system of differential equations for a gene regulatory network by using genetic programming," in *Proc. Congress on Evolutionary Computation*. IEEE Press, 2001, pp. 720–726.

[11] J. Bongard and H. Lipson, "Automated reverse engineering of nonlinear dynamical systems," *PNAS, Proceedings of the National Academy of Sciences of the United States of America*, vol. 104, no. 24, pp. 9943–9948, 12 Jun. 2007.

[12] B. A. McKinney, J. E. Crowe, H. U. Voss, P. S. Crooke, N. Barney, and J. H. Moore, "Hybrid grammar-based approach to nonlinear dynamical system identification from biological time series," *Phys. Rev. E*, vol. 73, p. 021912, Feb 2006.

[13] A. G. Floares, "A reverse engineering algorithm for neural networks, applied to the subthalamopallidal network of basal ganglia," *Neural Networks*, vol. 21, no. 2-3, pp. 379–386, Mar. 2008.

[14] J. Tikka and J. Hollmén., "Learning linear dependency trees from multivariate time-series data," *In Proceedings of the Workshop on Temporal Data Mining: Algorithms, Theory and Applications (in conjunction with The Fourth IEEE International Conference on Data Mining),Brighton, UK*, November 2004.

[15] G. Kronberger, S. Fink, M. Kommenda, and M. Affenzeller, "Macroeconomic time series modeling and interaction networks," in *Proceedings of the 2011 international conference on Applications of evolutionary computation - Volume Part II, EvoApplications'11*, 2011, pp. 101–110.

[16] J. McDermott, D. R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, R. Harper, K. De Jong, and U.-M. O'Reilly, "Genetic programming needs better benchmarks," in *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference, GECCO '12*, 2012, pp. 791–798.

[17] M. D. Schmidt and H. Lipson, "Age-fitness pareto optimization," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation,GECCO '10*, 2010, pp. 543–544.

[18] "Genetic programming on general purpose graphics processing units," http://www.gpgpgpu.com/.

[19] D. Sherry, K. Veeramachaneni, J. McDermott, and U.-M. O'Reilly, "Flex-gp: genetic programming on the cloud," in *Proceedings of the 2012 European conference on Applications of Evolutionary Computation, EvoApplications'12*, 2012, pp. 477–486.