Optimal spanning tree reconstruction in symbolic regression

January 2022

R. G. Neychev, I. A. Shibaev, V. V. Strijov

Abstract

This paper investigates the problem of regression model generation. A model is a superposition of primitive functions. The model structure is described by a weighted colored graph. Each graph vertex corresponds to some primitive function. An edge assigns a superposition of two functions. The weight of an edge equals the probability of superposition. To generate an optimal model one has to reconstruct its structure from its graph adjacency matrix. The proposed algorithm reconstructs the minimum spanning tree from the weighted colored graph. This paper presents a novel solution based on the prize-collecting Steiner tree algorithm. This algorithm is compared with its alternatives.

Keywords: symbolic regression; linear programming; superposition; minimum spanning tree; adjacency matrix

1 Introduction

Symbolic regression is a method to construct a non-linear model to fit data. A superposition of primitive functions defines the model structure. The set of primitive functions is fixed for a particular application problem. An optimization algorithm generates alternative model structures to select an optimal model. This paper proposes to define the model structure by a probabilistic graph. A spanning tree in the graph defines some superposition. To select an optimal model, a minimum spanning tree must be reconstructed from the graph.

The genetic programming methods [1] find an optimal subset in the primitive set, but require complex computations. The paper [2] describe methods that use additional constraints, like linear combinations of primitive functions. Symbolic regression is a model structure optimization method. Recent achievements are shown in [3].

Various methods to solve the symbolic regression problem are based on the matrix representation of the model structure [4]. However, these methods do not include constraints

on the number of arguments of the primitive functions and on the graph structure, which delivers admissible superposition. This paper solves the symbolic regression problem. It requires reconstructing an admissible superposition from the predicted adjacency matrix with edge probabilities. The k-minimum spanning tree (k-MST) reconstruction problem is stated. This problem is NP-hard, so only approximate solutions are applicable [5]. The k-MST is equivalent to the prize-collecting Steiner tree (PCST) problem [6] due to its equivalence of the relaxed formulation of the linear programming problem statement. The papers [5, 7, 8] present approximate solutions for the k-MST problem.

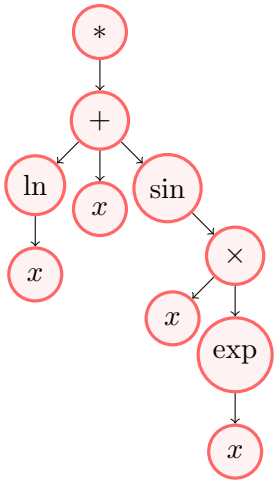

Figure 1: The regression model structure is a directed graph

The proposed solution is based on the relaxed version of k-MST problem, which transforms into the PCST problem with constant prizes, the same for all vertices. The fast algorithm for PSCT is described in [9]. An alternative solution is based on $(2-\varepsilon)$ approximation algorithm for PSCT problem. It is compared with the other algorithms, including the tree depth-first traverse, tree breadth-first traverse, Prim's algorithms.

2 The regression model selection problem

One has to select a regression model φ from a set of alternative models. The model fits a sample set $D = \{(x_i, y_i)\}$ and minimises the error

$$\hat{\varphi}(D) = \arg\min_{\varphi} \sum_{i=1}^{m} \big(\varphi(x_i) - y_i\big)^2. \tag{1}$$

The model is a superposition of the primitive functions from some given set. Fig. 1 shows an example. A structure of the model φ, a superposition corresponds to some graph $G = (V, E)$ where the primitive functions are placed in the vertices V. The root vertex is denoted by $*$. The model is $\varphi(D) = \ln(x) + x + \sin\big(x \times \exp(x)\big)$. Its structure as the

graph adjacency matrix is provided in Table 1. The primitive functions are listed in the first row. The elements of the matrix are probabilities of the edges E of the tree. The bold typesetting highlights the edges of the reconstructed tree M, which form the superposition φ. To reconstruct the model structure φ as superposition that is defined by the tree M one needs only the graph representation G and the primitive functions.

Table 1: Probabilities in the adjacency matrix generate the directed graph

arity	prim.	$*$	$+$	ln	sin	$\times$	exp	x
1	$*$	0.2	**0.7**	0.5	0.4	0.5	0.3	0.2
3	$+$	0.3	0.2	**1.0**	**0.8**	0.6	0.3	**0.7**
1	ln	0.3	0.2	0.0	0.0	0.1	0.5	**0.5**
1	sin	0.1	0.4	0.0	0.5	**0.9**	0.2	0.5
2	$\times$	0.3	0.0	0.3	0.5	0.0	**0.8**	**0.6**
1	exp	0.3	0.3	0.4	0.1	0.5	0.4	**0.4**

State the problem of the model structure reconstruction. There given a collection of sample sets $\{D_k\}$. Each sample set D_k corresponds to its model to fit. This model has the structure M_k. So there is a collection of pairs: a sample set and its model structure, $\{(D_k, M_k)\}$. Denote by P a map, which predicts the probabilities of nodes in the graph G using a sample set D. To define a model $\varphi(D)$ one has to reconstruct the model structure M from the graph G. Denote by R this tree reconstruction algorithm. The regression model $\hat{\varphi}(D)$, which solves the problem (1), is defined by $\hat{M} = R\big(P(D)\big)$. Since the tree M plays the main role, this paper sets the quality criterion of the tree reconstruction algorithm as follows:

$$\min_{M_k \in G} \frac{1}{K} \sum_{k=1}^{K} [\hat{M}_k = M_k)].$$

The reconstructed tree must be equal to the given tree to guarantee the selected regression model fits its sample set.

3 The superposition tree reconstruction problem

The key problem of this work is to propose and analyze the tree reconstruction algorithms. Each reconstructed tree defines a superposition from the previous section. There given a directed weighted graph $G = (V, E)$ with the colored vertices v_i and the root vertex r. Every vertex $v_i \in V$ has its color $t(v_i) = t_i$. Every edge $e_i \in E$ has its weight $w(e_i) = c_i \in [0, 1]$.

The goal is to reconstruct a minimum-weight directed tree with the root r. It must cover at least k vertices in the given graph G. And a number of tail edges, out-coming from a vertex v_i of the tree, must be less than or equal to t_i. The root r has one edge, $t_r = 1$.

Formulate this statement in the form of a linear programming problem with integer

constraints:

$$
\begin{aligned}
\underset{\substack{x_e, z_S \\ e\in E, S\subseteq V\setminus\{r\}}}{\text{minimize}} \quad & \sum_{e\in E} c_e x_e \\
\text{s.t.} \quad & \sum_{\substack{e\in\delta(S): \\ e=(*,v_i),\ v_i\in\delta(S)}} x_e + \sum_{T:T\supseteq S} z_T \geqslant 1, && S\subseteq V\setminus\{r\}, \\
& \sum_{e\in E:\ e=(*,v)} x_e \leqslant 1, && v\in V, \qquad (2) \\
& \sum_{e\in E:\ e=(v,*)} x_e \leqslant t_i, && v\in V, \\
& \sum_{S\subseteq V\setminus\{r\}} |S| z_S \leqslant n-k, \\
& x_e \in \{0,1\}, && e\in E, \\
& z_S \in \{0,1\}, && S\subseteq V\setminus\{r\},
\end{aligned}
$$

where $x_e = 1$ if edge e is included into the final superposition and $x_e = 0$ otherwise, $z_S = 1$ for all vertices excluded from the final superposition. Denote by $e = (*, v)$ a directed edge with the tail v. Denote by $e = (v, *)$ a directed edge with the head v.

Every constraint in (2) has its specific purpose. The first constraint defines the structure of the solution graph as a tree with the root r. The second constraint defines the orientation of the tree: every vertex has no more than one incoming edge. The third constraint defines the arity of the used primitive functions, so the number of edges with a certain vertex as their source is fixed. The fourth constraint states that the final tree has at least k vertices. If all weights are non-negative, the fourth constraint on the minimal number of vertices takes a more strict form: the number of vertices should be exactly k. However, the weaker constraint allows finding possible connections with other optimization problems. The exact form of the constraints in (2) has the same goal.

4 The k-MST and PCST algorithms for tree reconstruction

Definition 1 *k-minimum spanning tree (k-MST). There given a weighted graph $G = (V, E)$ with root r and edge weights $w(e_i) = c_i \geqslant 0,\ e_i \in E$. Construct a minimum-weight directed tree with root r, which covers at least k vertices in G.*

If the same problem is formulated for the directed graphs, the final tree with root r should be directed. The linear programming problem for the directed k-MST excludes the third

condition in (2) and takes the form:

$$\begin{array}{lll}
\underset{\substack{x_e, z_S \\ e\in E, S\subseteq V\setminus\{r\}}}{\text{minimize}} & \displaystyle\sum_{e\in E} c_e x_e & \\
\text{s.t.} & \displaystyle\sum_{\substack{e\in\delta(S):\\ e=(*,v_i),\ v_i\in\delta(S)}} x_e + \sum_{T:T\supseteq S} z_T \geqslant 1, & S\subseteq V\setminus\{r\}, \\
& \displaystyle\sum_{e\in E:\ e=(*,v)} x_e \leqslant 1, & v\in V, \\
& \displaystyle\sum_{S\subseteq V\setminus\{r\}} |S| z_S \leqslant n-k, & \\
& x_e\in\{0,1\}, & e\in E, \\
& z_S\in\{0,1\}, & S\subseteq V\setminus\{r\}.
\end{array} \tag{3}$$

In such form the k-MST problem is different from the original superposition tree reconstruction problem (2) by the absence of the third constraint on the arity of primitive functions. It is equivalent to the constraint on number of edges out-coming from a vertex.

Definition 2 *Prize-collecting Steiner tree (PCST). There given a weighted graph* $G = (V, E)$ *with root* r *and edge weights* $w(e_i) = c_i \geqslant 0,\ e_i \in E$, *where every vertex* $v_i \in V$ *is assigned with a* prize $\pi(v_i) = \pi_i \geqslant 0$. *Construct a tree* T *with root* r, *which minimizes the functional:*

$$\sum_{e\in E} c_e x_e + \sum_{S\subseteq V\setminus\{r\}} \pi(S) z_S,$$

where $x_e \in \{0,1\},\ x_e = 1$ *if* $e \in E$ *is included in the three* T, $z_S \in \{0,1\},\ z_S = 1$ *for all vertices excluded from tree* $T\ S = V\setminus V(T)$ *and* $\pi(S) = \sum_{v\in S}\pi(v)$.

In case of directed graphs this problem generalizes to the asymmetric A-PCST problem. The linear programming problem for A-PCST takes the form:

$$\begin{array}{lll}
\underset{\substack{x_e, z_S \\ e\in E, S\subseteq V\setminus\{r\}}}{\text{minimize}} & \displaystyle\sum_{e\in E} c_e x_e + \sum_{S\subseteq V\setminus\{r\}} \pi(S) z_S & \\
\text{s.t.} & \displaystyle\sum_{\substack{e\in\delta(S):\\ e=(*,v_i),\ v_i\in\delta(S)}} x_e + \sum_{T:T\supseteq S} z_T \geqslant 1, & S\subseteq V\setminus\{r\}, \\
& \displaystyle\sum_{e\in E:\ e=(*,v)} x_e \leqslant 1, & v\in V, \\
& x_e\in\{0,1\}, & e\in E, \\
& z_S\in\{0,1\}, & S\subseteq V\setminus\{r\}.
\end{array} \tag{4}$$

If the last constraint from (2) is included into the optimized functional, the k-MST and A-PCST problems have equivalent constraints and only differ in the optimized functional. Such transformation is possible according to the Karush-Kuhn-Tucker conditions

and [10]. If the prize values are equivalent $\pi(v) = \lambda$ the only difference is the constant term $\lambda(n-k)$. So the optimization problems k-MST and A-PCST take the forms:

$$\underset{\substack{x_e, z_S \\ e\in E, S\subseteq V\setminus\{r\}}}{\text{minimize}} \quad \sum_{e\in E} c_e x_e + \lambda \left(\sum_{S\subseteq V\setminus\{r\}} |S| z_S - (n-k) \right), \qquad (k\text{-}MST)$$

$$\underset{\substack{x_e, z_S \\ e\in E, S\subseteq V\setminus\{r\}}}{\text{minimize}} \quad \sum_{e\in E} c_e x_e + \sum_{S\subseteq V\setminus\{r\}} \pi(S) z_S. \qquad (A\text{-}PCST)$$

The constant λ stands for non-negative Lagrange multiplier in the k-MST problem and for vertex prize in the A-PCST. There are several algorithms to solve the PCST problem, but not A-PCST. A possible workaround is to release the constraints on the graph orientation so the PCST algorithm could reconstruct the tree orientation later. If the constraints on arities are not present, the original problem of reconstructing the superposition tree is reduced to the k-MST problem on a directed graph. If all prizes are equivalent, this problem is reduced to A-PCST.

5 The $(2-\varepsilon)$-approximation algorithm for constrained forest problem

An overview of techniques for constrained forest problems is provided in [11]. This research selects relevant results. There given a weighted undirected graph $G = (V, E)$. All its weights $w(e_i) = c_i \geqslant 0,\ e_i \in E$. There given some function $f : 2^V \to \{0, 1\}$. State the linear programming problem with integer constraints:

$$\begin{aligned} \underset{x_e:\ e\in E}{\text{minimize}} \quad & \sum_{e\in E} c_e x_e \\ \text{s.t.} \quad & x\big(\delta(S)\big) \geqslant f(S), && S \subset V,\ S \neq \emptyset, \\ & x_e \in \{0, 1\}, && e \in E, \end{aligned} \qquad (5)$$

where $x\big(\delta(S)\big) = \sum\limits_{e\in\delta(S)} x_e$; $x_e = 1$ if edge e is included into the final set. The function $\delta(S)$ stands for all edges from E such that only one of the connected vertices is included in S.

Assume the map f, which satisfies

$$f(V) = 0, \underbrace{f(S) = f(V\setminus S)}_{\text{symmetry}}, \underbrace{A, B \subset V :\ A \cap B = \emptyset,\ f(A) = f(B) = 0 \to f(A \cup B) = 0}_{\text{disjunctivity}}.$$

In these conditions are satisfied, f specifies the number of edges, which starts in the set of vertices S. For example, for the minimum matching problem $f(S) = 1$ if and only if $|S| \mod 2 = 1$.

Lemma 1 *Let $B \subseteq S \subset V$. Then $f(S) = 0$ and $f(B) = 0$ leads to $f(S \backslash B) = 0$.*

A problem with such description is the *optimal forest search problem with correct constraints*. Such problem statement (5) with appropriate map f fits many well-known weighted graph problems, e.g. minimum backbone search, *st*-path, the Steiner problem on the minimum tree. The last problem is NP-complete, so apply an approximate algorithm.

Definition 3 *(α-approximation algorithm) A heuristic polynomial algorithm that delivers a solution for some optimization problem is called α-approximation if it guarantees a constraint-satisfying solution to this optimization problem with a factor less or equal to α, so the solution is different from the optimal one no more that by α times in terms of the optimized functional.*

To propose an appropriate approximate algorithm, the integer constraints in (5) should be relaxed:

$$\begin{aligned} \underset{x_e:\ e\in E}{\text{minimize}} \quad & \sum_{e\in E} c_e x_e \\ \text{s.t.} \quad & \sum_{e\in\delta(S)} x_e \geqslant f(S), && S \subset V,\ S \neq \emptyset, \\ & x_e > 0, && e \in E, \end{aligned} \tag{6}$$

The dual problem takes the form:

$$\begin{aligned} \underset{y_S:\ S\subset V,\, S\neq\emptyset}{\text{maximize}} \quad & \sum_{S\subset V} f(S) y_S \\ \text{s.t.} \quad & \sum_{S:\ e\in\delta(S)} y_S \leqslant c_e, && e \in E, \\ & y_S > 0, && S \subset V,\ S \neq \emptyset, \end{aligned} \tag{7}$$

regarding a complementary slackness condition: $y_S \cdot \left(\sum\limits_{e\in\delta(S)} x_e - f(S) \right) = 0, \quad S \subset V.$

Denote the set of vertices $A = \{v \in V : f(\{v\}) = 1\}$. Propose an adaptive greedy $\left(2 - \frac{2}{|A|}\right)$ — approximation algorithm for problems of the form (5). The algorithm consists of two stages. On the first stage, it greedily combines clusters of vertices, increasing the dual variables y_S. Initially, every vertex belongs to its own cluster. If the next edge e reaches equality in the constraints in (7), this edge is added to the set S and the connected clusters will be merged. This stage is similar to Kruskal minimum spanning tree algorithm. (Opposed to the Kruskal algorithm, the effective weight of the next edge is minimized, not the weight itself. This change makes the algorithm capable of adapting to specific data.) In the second stage, some edges are removed from the final set S. If the edge pruning does not violate the constraints, this edge is to be removed.

The pseudo-code for the described algorithm is provided in the Appendix of this paper. The index Z_{DRLP} in Algorithm 1 stands for dual-relaxed linear programming. The initial value of $F := \emptyset$ in 1 is equivalent to the assumption $x_e = 0 \quad e \in E$. According to the slackness conditions $y_S = 0,\ S \subset V,\ S \neq \emptyset$.

At any step of the algorithm, cluster $\mathcal{C}$ contains two components $\mathcal{C} = \mathcal{C}_i \cup \mathcal{C}_a$, where $C \in \mathcal{C}_a$ if $f(C) = 1$ and $C \in \mathcal{C}_i$ otherwise. Let's call $\mathcal{C}_a$ an active component. The variables $d(v)$ in this algorithm are related to the variables y_S from (7) as $d(i) = \sum\limits_{S: i \in S} y_S$.

Analyze two different components $C_q,\ C_p,\ C_q \cap C_p = \emptyset$ on some iteration of the first stage of the algorithm. All y_S should be evenly by some ε without violating the constraints

$$\sum_{S:\ e \in \delta(S)} y_S \leqslant c_e.$$

In terms of $d(v)$, this condition takes the form $\sum\limits_{S:\ e \in \delta(S)} y_S = d(v_1) + d(v_2),\ e = (v_1, v_2)$, so $y_S = 0$ for any S, such that $v_1, v_2 \in S$ because the components only grow on the first stage. Increasing some of the components by ε leads to the equation

$$d(v_1) + d(v_2) + \varepsilon \cdot \big(f(C_q) + f(C_p)\big) \leqslant c_e,\ e = (v_1, v_2),$$

which leads to the formula used in line 10 of the Algorithm 1. When the next edge is included in the component, the sum $\sum\limits_{S:\ e \in \delta(S)} y_S$ will not increase, so the constraints are satisfied.

Edges that can be removed from F without the addition of new active components are removed in the second stage of the algorithm. The following lemma defines properties of connected components in F'.

Lemma 2 *For every connected component N from F' the equation holds: $f(N) = 0$.*

The following theorem states that the solution derived by the described algorithm meets the constraints of the original linear programming problem.

Theorem 1 *The edge set F' derived by Algorithm 1 meets all the constraints of the original problem* (5)*.*

Meeting the constraints leads to the following inequality (because the dual problem is a constraint from below) :

$$Z_{\text{DRLP}} = \sum_{S \subset V} y_S \leqslant Z^*_{RLP} \leqslant Z^*_{LP},$$

where Z^*_{LP} is the optimal solution of (5). The following theorem describes the properties of Algorithm 1.

Lemma 3 *Denote graph H where every vertex corresponds to one of the connected components $C \in \mathcal{C}$ on the fixed step of the algorithm. Edge (v_1, v_2) is present if there exists an edge $\hat{e}$ of the original graph included in F': $\hat{e} \in F'$, so the graph H is a forest. There are no leaf vertices within H such that correspond to inactive vertices in the original graph.*

Theorem 2 *Algorithm 1 is an α-approximate algorithm for problem* (5) *with* $\alpha = 2 - \frac{2}{|A|}$ *where* $A = \{v \in V : f(\{v\}) = 1\}$.

Despite this theoretical basis, there is no appropriate function f to state the PCST problem as referenced in (5). To apply these conditions, the Algorithm 1 needs several modifications.

6 The upgraded problem statement for PCST

As in the A-PCST case, the relaxed form of the linear programming problem PCST takes form:

$$\begin{aligned}
\underset{\substack{x_e, s_v \\ e \in E, v \in V\setminus\{r\}}}{\text{minimize}} \quad & \sum_{e \in E} c_e x_e + \sum_{v \in V\setminus\{r\}} (1 - s_v)\pi_v \\
\text{s.t.} \quad & \sum_{e \in \delta(S)} x_e \geqslant s_v, && S \subseteq V\setminus\{r\},\ v \in S, \\
& x_e \geqslant 0, && e \in E, \\
& s_v \geqslant 0, && v \in V\setminus\{r\}.
\end{aligned} \tag{8}$$

This problem statement is different from the original one (4), but it is possible to align the k-MST problem with it. Indicators s_v show that vertex v is included in the tree.

The dual problem takes the form:

$$\begin{aligned}
\underset{y_S:\ S \subset V\setminus\{r\}}{\text{maximize}} \quad & \sum_{S \in V\setminus\{r\}} y_S \\
\text{s.t.} \quad & \sum_{S : e \in \delta(S)} y_S \leqslant c_e, && e \in E, \\
& \sum_{S \subseteq T} y_S \leqslant \sum_{v \in T} \pi_v, && T \subset V\setminus\{r\}, \\
& y_S \geqslant 0, && S \subset V\setminus\{r\}.
\end{aligned} \tag{9}$$

Algorithm 2 solves this problem. It is similar to Algorithm 1. The dual variables should be updated at an even rate with additional constraints. Then ε will take the minimum of two values, according to both groups of constraints. (The function λ stays for the indicator that the new component is active, similar to the f function in the Algorithm 1.) The broader analysis of the approximation properties of the upgraded algorithm is provided in [11]. Algorithm 2 is an α-approximate algorithm for PCST problem with $\alpha = 2 - \frac{2}{n-1}$, where n is the number of vertices in the graph G.

7 Computational experiment

The main goal of the experiment is to reconstruct the correct superposition tree. The algorithms used for reconstruction are listed below.

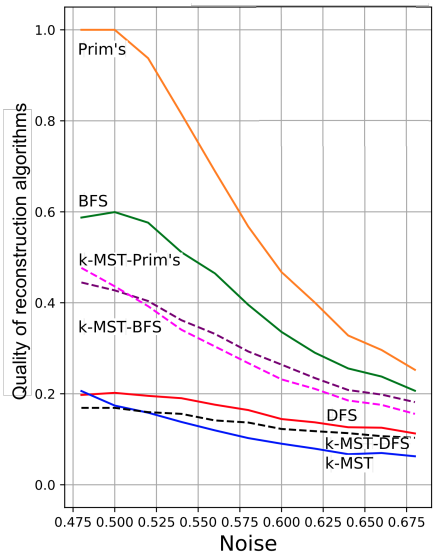

Figure 2: Quality of the reconstruction algorithms with primitive functions of small arities and unordered inputs

DFS, BFS. Greedy tree depth-first traverse and Greedy tree breadth-first traverse algorithms. Traversing the edges with the highest weights is equivalent to selecting the most probable path. The traverse algorithm stops when the number of tail edges out-coming from some vertex equals the arity of the corresponding function.

Prim's algorithm. This algorithm reconstructs the minimum spanning tree for a graph with additional constraints on the primitive functions' arity. These constraints assign the minimum weight edge. After a vertex is added, all tail edges of this vertex are excluded to preserve the direction of the tree. If the number of edges starting in some vertex exceeds the corresponding arity, the rest of the edges are excluded from the set of possible edges in this vertex. The algorithm is independent of the traverse procedure. In case of small noise in the adjacency matrix, this algorithm reconstructs the superposition tree without errors.

Table 2: Quality of reconstruction algorithms with uniform noise near to 0.5

Algorithm, noise	0.50	0.52	0.54	0.56	0.58
DFS	0.20	0.20	0.19	0.18	0.16
BFS	0.60	0.58	0.51	0.46	0.40
Prim's	1.00	0.94	0.81	0.69	0.57
k-MST	0.17	0.16	0.14	0.12	0.10
k-MST-DFS	0.17	0.16	0.16	0.14	0.14
k-MST-BFS	0.43	0.40	0.36	0.33	0.29
k-MST-Prim's	0.44	0.39	0.34	0.33	0.27

Algorithms based on PCST. The adjacency matrix M must be transformed to the undirected form. Use the square matrix M' without the last column. PCST takes the adjacency matrix $1 - \frac{1}{2}(M' + M'^{\mathsf{T}})$ with prize value 0.5 for every vertex. The prize value is equal to 0.5 because with smaller values, the tree will be truncated: if the noise is equal to 0.5 some vertices might be pruned by error. In case of greater prize values, the PCST tree might include unnecessary vertices. The tree is reconstructed with one of the described algorithms. The results of PCST can be used as prior for other approaches, $M' := \frac{1}{2}(M'_{\text{PCST}} + M')$, so the PCST results update M'.

The data generation procedure has the following assumptions: the arities of the function are generated by the Binomial distribution, so there are many functions with small arity, and all primitive functions have only one input. Any case with partial reconstruction causes an error. The quality of reconstruction algorithms is $\frac{1}{K}\sum_{k=1}^{K}[R\big(\bar{N}(M_k)\big) = M_k]$,where R is the reconstruction algorithm and $\bar{N} = \big(N - \min(N)\big)/\big(\max(N) - \min(N)\big)$ is the normalized noise matrix. The matrix N is generated as $N(M) = M + U(-\alpha, \alpha)$. The random generator returns a matrix with the same shape as M where every element is an independent variable from the uniform distribution in the segment $[-\alpha, \alpha]$.

Here is the list of seven compared algorithms: DFS, BFS, Prim's algorithm, k-MST via PCST, k-MST + DFS, k-MST + BFS, k-MST + Prim's algorithm. Fig. 2 shows the error of the reconstruction algorithms with the noise close to 0.5 threshold. Prim's algorithm delivers the best results. The second best solution is based on BFS. Table 2 accompanies Fig. 2 and shows the reconstruction quality of seven algorithms for the border noise values 0.50–0.58.

8 Conclusion

This paper proposes and compares different algorithms of superposition reconstruction for the symbolic regression problem. Prim's algorithm delivers the most accurate results and is resistant to small data noise. The proposed algorithm delivers accurate results but is more prone to noise in the superposition matrix. The algorithms based on BFS and DFS cannot reconstruct the original superposition in noisy cases. PCST with BFS used for superposition matrix reconstruction shows baseline results.

References

[1] Koza J. R. 1994. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing* 4:87–112.

[2] Searson, D. P., D. E. Leahy, and M. J. Willis. 2010. Gptips: an open source genetic programming toolbox for multigene symbolic regression. *Proceedings of the International multiconference of engineers and computer scientists* 1:77–80.

[3] Stanley, K. O. and R. Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10(2):99–127.

[4] Bochkarev, A. M., I. L. Sofronov, and V.V. Strijov. 2017. Generation of expertly-interpreted models for prediction of core permeability. *Sistemy i Sredstva Informatiki — Systems and Means of Informatics* 27(3):74–87.

[5] Ravi, R., R. Sundaram, M. V. Marathe, D. J. Rosenkrantz, and S. S. Ravi. 1996. Spanning trees — short or small. *SIAM Journal on Discrete Mathematics* 9(2):178–200.

[6] Chudak, F. A., T. Roughgarden, and D. P. Williamson. 2004. Approximate k-MSTS and k-Steiner trees via the primal-dual method and Lagrangean relaxation. *Mathematical Programming* 100(2):411–421.

[7] Awerbuch, B., Y. Azar, A. Blum, and S. Vempala. 1998. New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM Journal on computing* 28(1):254–262.

[8] Arora, S. and G. Karakostas. 2006. A $2 + \varepsilon$ approximation algorithm for the k-MST problem. *Mathematical Programming* 107(3):491–504.

[9] Hegde, C., P. Indyk, and L. Schmidt. 2014. A fast, adaptive variant of the Goemans-Williamson scheme for the prize-collecting steiner tree problem. *Workshop of the 11th DIMACS Implementation Challenge* 2:3.

[10] Ras, C., K. Swanepoel, and D. A. Thomas. 2017. Approximate Euclidean Steiner trees. *Journal of Optimization Theory and Applications* 172(3):845–873.

[11] Goemans, M. X. and D. P. Williamson. 1995. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317.

Appendix 1: Tree reconstruction algorithms

Algorithm 1: $(2-\varepsilon)$-approximation algorithm for problem (5)

Data: Weighted undirected graph $G=(V,E)$ with non-negative weights $c_i \geqslant 0$; map f

Result: Forest F'; optimized in problem functional (5) value Z_{DRLP}

Stage 1, Merging

begin

$F \longleftarrow \emptyset$

$Z_{\mathrm{DRLP}} \longleftarrow 0$

$\mathcal{C} \longleftarrow \{\{v\} : v \in V\}$

foreach $v \in V$ **do**

$d(v) \longleftarrow 0$

while $\exists C \in \mathcal{C} : f(C) = 1$ **do**

$e^* = \underset{\substack{e=(i,j):\\ i\in C_p\in\mathcal{C},\ j\in C_q\in\mathcal{C}, C_p\neq C_q}}{\arg\min} \varepsilon(e)$ where $\varepsilon(e) = \frac{c_e - d(i) - d(j)}{f(C_p)+f(C_q)}$

$F \longleftarrow F \cup e^*$

foreach $C \in \mathcal{C}$ **do**

foreach $v \in C$ **do**

$d(v) \longleftarrow d(v) + \varepsilon(e^*) \cdot f(C)$

$Z_{\mathrm{DRLP}} \longleftarrow Z_{\mathrm{DRLP}} + \varepsilon(e^*) \sum_{C\in\mathcal{C}} f(C)$

$\mathcal{C} \longleftarrow \mathcal{C} \backslash \{C_p\} \backslash \{C_q\} \cup \{C_p \cup C_q\}$ (e^* connects components C_q and C_p)

Stage 2, pruning

$F' \longleftarrow \{e \in F : \exists N \in (V, F\backslash\{e\}), f(N) = 1\}$, where N is the connected component

Algorithm 2: $(2-\varepsilon)$-approximate algorithm for PCST problem

Data: Weighted undirected graph $G = (V, E)$ with non-negative edges' weights $c_i \geqslant 0$, prizes $\pi_i \geqslant 0$ and root r

Result: Tree F' including vertex r

Stage 1, Merging

begin

$F \longleftarrow \emptyset$

$Z_{\text{DRLP}} \longleftarrow 0$

$\mathcal{C} \longleftarrow \{\{v\} : v \in V\}$

foreach $v \in V$ **do**

Remove markup from v

$d(v) \longleftarrow 0$

$w(\{v\}) \longleftarrow 0$

if $v = r$ **then** $\lambda(\{v\}) \longleftarrow 0$

else $\lambda(\{v\}) \longleftarrow 1$

while $\exists C \in \mathcal{C} : \lambda(C) = 1$ **do**

$e^* = \underset{\substack{e=(i,j):\\ i \in C_p \in \mathcal{C},\ j \in C_q \in \mathcal{C}, C_p \neq C_q}}{\arg\min} \varepsilon_1(e)$ where $\varepsilon_1(e) = \frac{c_e - d(i) - d(j)}{\lambda(C_p) + \lambda(C_q)}$

$C^* = \underset{C : C \in \mathcal{C},\ \lambda(C) = 1}{\arg\min} \varepsilon_2(C)$ where $\varepsilon_2(C) = \sum_{i \in C} \pi_i - w(C)$

$\varepsilon = \min\big(\varepsilon_1(e^*), \varepsilon_2(C^*)\big)$

foreach $C \in \mathcal{C}$ **do**

$w(C) \longleftarrow w(C) + \varepsilon \cdot \lambda(C)$

foreach $v \in C$ **do**

$d(v) \longleftarrow d(v) + \varepsilon \cdot \lambda(C)$

if $\varepsilon_1(e^*) > \varepsilon_2(C^*)$ **then**

$\lambda(C^*) \longleftarrow 0$ Mark all unmarked vertices from C^* with C^*.

else

$F \longleftarrow F \cup e^*$

$\mathcal{C} \longleftarrow \mathcal{C} \backslash \{C_p\} \backslash \{C_q\} \cup \{C_p \cup C_q\}$ (e^* connects components C_q and C_p)

$w(C_p \cup C_q) \longleftarrow w(C_p) + w(C_q)$

if $r \in C_p \cup C_q$ **then** $\lambda(C_p \cup C_q) \longleftarrow 0$

else $\lambda(C_p \cup C_q) \longleftarrow 1$

Stage 2, pruning

F' is derived from F by dropping the maximum number of edges meeting the constraints:

1. All unmarked vertices are connected with root r.
2. If vertex marked with C is connected with root r, all other vertices marked with C should be connected with root r as well.

Algorithm 3: Superposition tree reconstruction with Prim's algorithm

Data: Noised superposition matrix$M \in \mathbb{R}_{+}^{n\times(n+1)}$, list l with $n-1$ arity values for used functions

Result: Superposition matrix M_{res} with correct arities

begin

- $l \longleftarrow [1] + l$ (add 1 to the list)
- $M' \longleftarrow$ zero matrix of shape $n \times (n+1)$
- $used \longleftarrow \{0\}$
- $edges \longleftarrow \emptyset$
- **foreach** $j \in range(0, n)$ **do**
  - **if** $j \notin used$ **then**
    - $edges \longleftarrow edges \cup (0, j, M[0][j])$ (from, to, weight)
- **while** $edges \neq \emptyset$ **do**
  - Find tuple $(from, to, w)$ maximizing the edge weight w over all $edges$
  - **foreach** $j \in used$ **do**
    - $M[to][j] = 0$
  - **foreach** $j \in range(0, n)$ **do**
    - **if** $j \notin used$ **then**
      - $edges \longleftarrow edges \cup (to, j, M[to][j])$ (from, to, weight)
  - **if** $to \neq n$ **then**
    - $edges \longleftarrow edges \backslash (from, to, w)$
    - $l[from] \longleftarrow l[from] - 1$
  - Remove from $edges$ all tuples (i, j, w) with $j = to$
  - **if** $l[to] = 0$ **then**
    - Remove from $edges$ all tuples (i, j, w) with $i = from$

Algorithm 4: Superposition tree reconstruction with algorithm for the PCST problem

Data: Noised superposition matrix$M \in \mathbb{R}_{+}^{n\times(n+1)}$, list l with $n-1$ arity values for used functions

Result: Superposition matrix M_{res} with correct arities

begin

Drop the last column from matrix M to derive matrix M'

$M'_{new} = 1 - \frac{M'+M'^{T}}{2}$

$M'_{pcst} = PCST(M'_{new}, 0.5)$

Add zero column to the M'_{pcst} on the right to derive M_{pcst}

Reconstruct the tree from M_{pcst} with some traverse procedure from the root vertex to derive M_{res}

Appendix 2: Proofs to the lemmas and theorems

Proof 1 (to lemma 1) *The Symmetry property leads to* $f(V\backslash S) = 0$. *Since* $V\backslash S \cap B = \emptyset$, *the disjunctivity property leads to* $f\big((V\backslash S) \cup B\big) = 0$. *According to the symmetry property, the equation holds:*

$$f\Big(V\backslash\big((V\backslash S) \cup B\big)\Big) = f(S\backslash B) = 0.$$

Proof 2 (to lemma 2) *Recall that* F' *is constructed from* F *via pruning. Hence there is a connected component* $C \in F$ *such that* $N \subseteq C$. *The algorithm has stopped, so* $f(C) = 0$. *All the edges* $\delta(N)$ *which started from* N *before pruning and were pruned. Then there is no component* $\hat{N}$ *such that*$f(\hat{N}) = 1$ *present in* $(V, E\backslash\{e\})$, $e \in \delta(N)$. *Denote the* C *components derived via edge pruning from* $\delta(N)$ *as* $N, N_1, \ldots, N_{|\delta(N)|}$), *then* $f(N_i) = 0$. *According to disjunctive property* $f\big(\bigcup_{i=1}^{|\delta(S)|} N_i\big) = 0$. *Hence, according to Lemma 1* $f(N) = f\big(V\backslash\bigcup_{i=1}^{|\delta(S)|} N_i\big) = 0$.

Proof 3 (to theorem 1) *Assume there is empty set* $S \subset V$, $S \neq \emptyset$, *such that* $\sum e \in \delta(S)x_e < f(S)$. *Then* $f(S) = 1$. *For every connected component* $C_1, \ldots, C_m$ *from* (V, F') *one of the following equations stand:* $C_i \subseteq S$ *or* $C_i \cap S = \emptyset$ *according to* $\sum_{e\in\delta(S)} x_e = 0$. *According to Lemma 2 and disjunctive property,* $f(S) = f\big(\bigcup_j C_{i_j}\big) = 0$, *which is contrary to the original assumption* $f(S) = 1$.

Proof 4 (to theorem 2) *Recall the inequality:*

$$Z^*_{LP} \leqslant \sum_{e\in F'} c_e \leqslant \big(2 - \frac{2}{|A|}\big) Z_{DRLP} \leqslant \big(2 - \frac{2}{|A|}\big) Z^*_{LP}.$$

1. *The first part holds according to the* Z^*_{LP} *definition and the fact that* F' *meets the constraints according to Theorem 1.*

2. *The last part holds because for the optimal solution* Z^*_{LP} *of the problem 5 the following inequality holds due to the below constraints of the dual problem:* $Z_{DRLP} = \sum_{S\subset V} y_S \leqslant Z^*_{RLP} \leqslant Z^*_{LP}$.

3. *The middle part should be proven explicitly.*

After the Algorithm 1 stops $c_e = \sum y_S$, *so the following equation stands:*

$$\sum_{e\in F'} c_e = \sum_{e\in F'} \sum_{S: e\in\delta(S)} y_S = \sum_{S\subset V} y_S \cdot |F' \cap \delta(S)|.$$

So the following inequality should will be proved by induction:

$$\sum_{e\in F'} c_e = \sum_{S\subset V} y_S \cdot |F' \cap \delta(S)| \leqslant \big(2 - \frac{2}{|A|}\big) Z_{DRLP} = \big(2 - \frac{2}{|A|}\big) \sum_{S\subset V} y_S.$$

Basis case: *On the first step of the algorithm* $y_S = 0$.

Inductive step: *assume the induction hypothesis that for a particular step* k*, the inequality holds. On the* $(k+1)$ *step the left hand side is increased by* $\varepsilon \sum\limits_{S \in \mathcal{C}_a} |F' \cap \delta(S)|$*, where* $\mathcal{C}_a$ *stays for all active components and* $f(C) = 1$*. The right hand side will be increased by* $\varepsilon\big(2 - \frac{2}{|A|}\big) \cdot |\mathcal{C}_a|$*. Let's focus on the following inequality:*

$$\sum_{S \in \mathcal{C}_a} |F' \cap \delta(S)| \leqslant \big(2 - \frac{2}{|A|}\big) \cdot |\mathcal{C}_a|.$$

Denote the number of edges starting in S *as* $d(S) = |F' \cap \delta(S)|$*. So*

$$\sum_{S \in \mathcal{C}_a} d(S) = \sum_{S \in \mathcal{C}} d(S) - \sum_{S \in \mathcal{C}_i} d(S) \leqslant 2(|\mathcal{C}_a| + |\mathcal{C}_i| - 1) - \sum_{S \in \mathcal{C}_i} d(S),$$

where $\mathcal{C} = \mathcal{C}_i \cup \mathcal{C}_a$*. The last inequality holds because* F' *defines a forest in the original graph. The last step is to prove that*

$$\sum_{S \in \mathcal{C}_i} d(S) \geqslant 2|\mathcal{C}_i|, \tag{10}$$

which implies

$$\sum_{S \in \mathcal{C}_a} d(S) \leqslant 2(|\mathcal{C}_a| + |\mathcal{C}_i| - 1) - 2|\mathcal{C}_i| = 2\big(1 - \frac{1}{|\mathcal{C}_a|}\big) \cdot |\mathcal{C}_a| \leqslant 2\big(1 - \frac{1}{|A|}\big) \cdot |\mathcal{C}_a|.$$

It holds because the number of clusters does not increase through time, or equivalently $|A| \geqslant |\mathcal{C}_a|$*. According to the Lemma 3 proves the inequality* (10)*, so the second part of the original statement is also correct.*

Proof 5 (to lemma 3) *The following lemma is needed to prove inequality* (10)*. Recall those lone inactive vertices are ignored due to the zero power of every inactive vertex, so they all can be subtracted in the inequality* $\sum_{S \in \mathcal{C}} d(S) \leqslant 2(|\mathcal{C}_a| + |\mathcal{C}_i| - 1)$*.*

Assume there is a leaf vertex $v \in V(H)$ *connected with edge* e*. This vertex corresponds to the inactive set* $C_v \in \mathcal{C}$*. This set* C_v *is included in one of the connected components* $N \in F$*, where* F *is the set of vertices before pruning. The fact that* $v \in V(H)$ *is a leaf implies that all edges connecting* C_v *with other vertices from* N *but the edge* e *were excluded during pruning.*

If the edge e *is excluded from the component* N*, which is a tree itself, the component splits into* N_1 *and* N_2*. Without loss of generality, assume* $C_v \subseteq N_1$*. The edge* e *was not pruned, so* $f(N_1) = 1$ *or* $f(N_2) = 1$*. Due to* $f(N) = 0$*, the only possible variant is* $f(N_1) = f(N_2) = 1$*. Other cases are contradictory to Lemma 1.*

Denote components $(C_v, C_1, \ldots, C_m)$ *derived from* N_1 *if edges from* F' *are used. For every component but* C_v $f(C_i) = 0 i \neq v$ *because the edges were pruned. But* $f(C_v) = 0$ *according to the original assumption. Hence* $f(N_1) = f\big(\bigcup\limits_{i=1}^{m} C_i \cup C_v\big) = 0$ *by symmetry,*

which is contradictory to $f(N_1) = 1$. So there are no leaf vertices in H, which correspond to inactive vertices in the original graph. Hence the power of every inactive vertex is greater or equal to two or zero. In the latter case, the vertices do not affect the target inequality.