Predicting optimal superposition trees in symbolic regression

R. G. Neychev,[1,*] I. A. Shibaev,[1,**] and V. V. Strijov[1,2,***]

[1]Moscow Institute of Physics and Technology, 9 Institutskiy per., Dolgoprudny, 141701, Russia
[2]Dorodnitsyn Computing Centre FRC CSC RAS, 40 Vavilova., 119333, Russia

Received December 02, 2020

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. A graph node corresponds to some primitive function. An edge sets a superposition of two functions. The weight of an edge sets a probability of the superposition. To generate an optimal model one has to reconstruct its structure from an adjacency matrix. This matrix contains superposition probabilities. The presented solution is based on search of minimum spanning tree in weighted colored graph. Solutions to symbolic regression problems tend to be easier to interpret and more effective. The symbolic regression problem is solved via restoration of the superposition tree from the predicted adjacency matrix. The restoration procedure is sensitive to noise in the predicted matrix. This paper presents a novel approach based on Prize-Collecting Steiner Tree algorithm. The proposed algorithm is compared with existing approaches to superposition trees restoration from noised adjacency matrix. Synthetic data is used in the experimental setup.

2010 Mathematical Subject Classification: 15A09, 15A63, 62J05, 62H30, 62H86

Keywords and phrases: forecasting model, symbolic regression, superposition, minimum spanning tree, adjacency matrix

1. INTRODUCTION

Symbolic regression is a well known approach to regression problems. It aims the optimization of the selected criteria with superposition of functions from some basis set. Such approach delivers simple yet accurate solutions. The derived solutions are much easier to interpret compared to deep neural networks or tree ensembles, which is a necessary property in many domains, especially in medicine or fraud-detection.

Symbolic regression problems usually require complex computations. While different approaches are used to find the optimal subset of the basis functions set, genetic algorithms [1] are one of the most popular ones. The symbolic regression problem is a popular illustration of the genetic programming [2]. More specific approaches are used if additional constraints are present (e.g. linear combinations of basis functions [3, 4] only). The symbolic regression might be considered as structural optimization problem, widely known in Deep Learning. Recent achievements show the promising outlook of the genetic algorithms [5, 6].

Different approach to solving the symbolic regression problems is based on matrix representation [7] of the computation graph of the target function. This leads to the necessity of superposition matrix prediction. However, fuzzy classification methods do not consider the constraints of the arity of used functions and the tree orientation. Solutions based on the classification methods predictions of the adjacency matrix are rather unstable solutions, the computational graph contains many low-probability edges.

This paper approaches the symbolic regression problem restoring the superposition matrix from the predicted edge probabilities. The solution incorporates several steps. First, the original problem

* E-mail: neychev@phystech.edu
** E-mail: shibaev.kesha@gmail.com
*** E-mail: strijov@phystech.edu

is reduced to the k-minimum spanning tree (k-MST) problem. This problem is NP-hard [8], so only approximate solutions are applicable [9]. The k-MST problem can be approached as Prize-Collecting Steiner Tree (PCST) problem [10] due to the equivalency of the relaxed formulations of the Linear Programming problem statements.

Many approximate solutions are present for the k-MST problem [9, 11, 12]. The approach presented in [12] is based on the classic PCST problem solution with $(2-\varepsilon)$ factor, which is described in [13]. In case of known root node the undirected tree can be restored to the directed one. Hence the algorithm restoring undirected tree from the graph in PCST problem might be used. Then the superposition matrix is restored from the directed tree.

This paper proposes a novel approach to solving the symbolic regression problem by restoring the superposition tree. The proposed solution is based on the relaxed version of k-MST problem, which transforms to PCST problem with constant prizes same for all vertices. The fast algorithm for PSCT problem is described in [14], source code available in [15]. The proposed approach is based on $(2-\varepsilon)$ approximating algorithm for PSCT problem. This approach is compared with variety of other solutions based on DFS and BFS and Prim's algorithm using the synthetic data.

2. PROBLEM STATEMENT

There given a collection of datasets $\mathcal{A} = \{\mathbf{A}_1, \ldots, \mathbf{A}_N\}$. Every dataset $\mathbf{A}_i = (\mathbf{X}_i, \mathbf{y}_i)$ the object matrix $\mathbf{X}_i$ and the target values vector $\mathbf{y}_i$ of length n_i. Denote a function f_i such that $f_i(\mathbf{X}_i) = y_i$. This function is a superposition of basic functions $g_1, \ldots, g_l$. Let's call f_i a generative superposition for the dataset $\mathbf{A}_i$.

All the datasets in the collection $\mathcal{A}$ are homogeneous w.r.t. the family of generative superpositions $\mathcal{F}$, where $\mathcal{F} = \{f : f = \sup(g_1, \ldots, g_l)\}$, $f_i \in \mathcal{F} \forall i$.

The optimal superposition f^* for every fixed pair $\mathbf{A} = (\mathbf{X}, \mathbf{y})$ is derived from the following optimization problem:

$$f^* = \arg\min_{f \in \mathcal{F}} S(f|\mathbf{X}, \mathbf{y}), \tag{1}$$

where S stays for the loss function. In the following paper Squared Deviation is used for this purpose:

$$S(f|\mathbf{X}, \mathbf{y}) = \|f(\mathbf{X}) - \mathbf{y}\|_2^2.$$

2.1. The proposed solution

Denote the mapping $h : \mathcal{A} \to \mathcal{F}$ which delivers the solution for the problem (1). The optimization problem takes the following interim form:

$$h^* = \arg\min_{h \in H} \sum_{i=1}^{N} \|h(\mathbf{A}_i)(\mathbf{X}_i) - \mathbf{y}_i\|_2^2,$$

where H is some parametric set of potential mappings. Due to the original problem formulation, every dataset $\mathbf{A}_i$ is paired with the generative superposition f_i, such that $\|f_i(\mathbf{X}_i) - \mathbf{y}_i\|_2 = 0$ (while its original form may be unknown). The optimization problem then takes the final form:

$$h^* = \arg\min_{h \in H} \sum_{i=1}^{N} \| \left(h(\mathbf{A}_i) - f_i\right)(\mathbf{X}_i)\|_2^2, \tag{2}$$

so the optimal mapping $h(A_i)$ delivers the optimal approximation of the superposition f_i in closed form.

Every superposition f_i can be represented as a graph where the basic functions are placed in the nodes. The similar representation is proposed in [7]. This approach is illustrated with the following example.

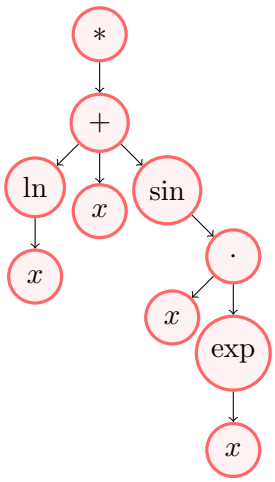

Figure 1. Computational graph

ar	$f(.)$	$*$	$+$	ln	sin	$\cdot$	exp	x
1	$*$	0	1	0	0	0	0	0
3	$+$	0	0	1	1	0	0	1
1	ln	0	0	0	0	0	0	1
1	sin	0	0	0	0	1	0	0
2	$\cdot$	0	0	0	0	0	1	1
1	exp	0	0	0	0	0	0	1

Table 1. Adjacency matrix

ar	$f(.)$	$*$	$+$	ln	sin	$\cdot$	exp	x
1	$*$	0.2	0.7	0.5	0.4	0.5	0.3	0.2
3	$+$	0.3	0.2	1.	0.8	0.6	0.3	0.7
1	ln	0.3	0.2	0	0.	0.1	0.5	0.5
1	sin	0.1	0.4	0	0.5	0.9	0.2	0.5
2	$\cdot$	0.3	0.	0.3	0.5	0.	0.8	0.6
1	exp	0.3	0.3	0.4	0.1	0.5	0.4	0.4

Table 2. Probability of adjacency matrix

Denote function $f(x) = \ln(x) + x + \sin(x \cdot e^x)$. Its graph form is shown on figure 1 and corresponding adjacency matrix is provided in table 1. The root node is represented with the wildcard $*$ symbol. Node x is repeated only for the visual sake.

To reconstruct the original function f one need only the graph visualization (or the corresponding adjacency matrix) and the mapping between the nodes indices and basic functions.

Hereinafter all the superpositions are constructed with the following procedure:

- The set basic functions (with fixed arity) g is defined and extended with the special function $* = *(x) : *(x) = x$ for the root node.
- The graph is constructed with the basic functions as the nodes. The directed tree is defined with this graph, where the root node is represented with $*$.

2.1.1. Symbolic Regression Problem Statement in matrix form Problem (2) can be reformulated in the matrix form:

$$h^* = \arg\min_{h \in H} \sum_{i=1}^{N} \| \left(F(M(h(\mathbf{A}_i))) - F(M(f_i))\right)(\mathbf{X}_i)\|_2^2,$$

where M maps function f to matrix form m and F maps matrix form m to function f.

Small deviations in the superposition matrix cause significant changes in the restored function. The following formulation focuses on finding the correct superposition matrix:

$$h^* = \arg\max_{h \in H} \sum_{i=1}^{N} [M(h(\mathbf{A}_i)) = M(f_i)].$$

The final mapping $h : \mathcal{A} \to \mathcal{F}$ incorporates several steps: $h(A_i) = F(R(P(A_i)))$.

- The mapping $P : \mathbf{X} \times \mathbf{y} \to \mathbf{M}$ (P stands for "predict") maps the original data points combined with labels $(\mathbf{X}, \mathbf{y})$ to approximate superposition matrices $M = P(\mathbf{A})$. All elements of the matrix have the following property: $M_{ij} \in [0, 1]$.
- The mapping $R : M \to M$ (R stands for "restore") provides the correct (in terms of arity) superposition matrix.
- The mapping F is already described.

Hence, the final solution can be derived from the optimization problem in the following form:

$$h^* = \arg\max_{R,P} \sum_{i=1}^{N} [R(P(\mathbf{A}_i)) = M(f_i)]. \tag{3}$$

2.2. Restoring the superposition tree

Definition 1. Superposition tree restoration problem Denote the directed weighted graph $G = (V, E)$ with colored vertices v_i and special vertex r. Every edge $e_i \in E$ e_i is assigned with a weight $w(e_i) = c_i \in [0, 1]$, every vertex $v_i \in V$ is assigned with a color $t(v_i) = t_i \in \mathbb{N}$.

The goal is to construct a minimum-weight directed tree with root node r covering at least k vertices in this graph so that the half-step of the vertex's output v_i (the number of edges coming from it) after covering is less than or equal to t_i (root vertex r has $t_r = 1$).

This constraints can be translated to the form of a linear programming problem with integer constraints:

$$
\begin{aligned}
\underset{\substack{x_e, z_S \\ e\in E, S\subseteq V\setminus\{r\}}}{\text{minimize}} \quad & \sum_{e\in E} c_e x_e \\
\text{s.t.} \quad & \sum_{\substack{e\in\delta(S):\\ e=(*,v_i),\ v_i\in\delta(S)}} x_e + \sum_{T:T\supseteq S} z_T \geqslant 1, && \forall S \subseteq V\setminus\{r\}, \\
& \sum_{e\in E:\ e=(*,v)} x_e \leqslant 1, && \forall v \in V, \qquad (4) \\
& \sum_{e\in E:\ e=(v,*)} x_e \leqslant t_i, && \forall v \in V, \\
& \sum_{S\subseteq V\setminus\{r\}} |S| z_S \leqslant n - k, \\
& x_e \in \{0, 1\}, && \forall e \in E, \\
& z_S \in \{0, 1\}, && S \subseteq V\setminus\{r\},
\end{aligned}
$$

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 final superposition. If edge source vertex is irrelevant it is denoted as $e = (*, v)$; if terminal vertex is irrelevant edge is denoted as $e = (v, *)$.

The constraints can be treated as following:

- First constraint defines the structure of the final solution as the tree with root r.
- Second constraint fixes the orientation of the final tree (so every vertex has the only incoming edge).
- Third constraint fixes the arity of the used basic functions (or the number of edges which has the certain vertex as their source).
- 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 can take more strict form "number of vertices should be exactly k". However, the softer constrain allows to find possible connections with other optimization problems. The exact form of the constraints in 4 has the same goal.

3. PCST ALGORITHM FOR SUPERPOSITION MATRIX RESTORATION

3.1. k-MST and PCST algorithms

Definition 2. (k-MST, k-minimum spanning tree)

Denote weighted graph $G = (V, E)$ with root vertex r and edge weights $w(e_i) = c_i \geqslant 0,\ e_i \in E$.

Construct a minimum-weight directed tree with root vertex r covering at least k vertices in this graph.

The same problem can be formulated for the directed graphs, so the final tree with root r should be directed. The linear programming problem for the directed k-MST takes the following form:

$$\begin{aligned}
\underset{\substack{x_e, z_S \\ e\in E, S\subseteq V\setminus\{r\}}}{\text{minimize}} \quad & \sum_{e\in E} c_e x_e \\
\text{s.t.} \quad & \sum_{\substack{e\in\delta(S):\\ e=(*,v_i),\ v_i\in\delta(S)}} x_e + \sum_{T:T\supseteq S} z_T \geqslant 1, && \forall S\subseteq V\setminus\{r\}, \\
& \sum_{e\in E:\ e=(*,v)} x_e \leqslant 1, && \forall v\in V, \qquad (5)\\
& \sum_{S\subseteq V\setminus\{r\}} |S| z_S \leqslant n-k, \\
& x_e\in\{0,1\}, && \forall e\in E, \\
& z_S\in\{0,1\}, && S\subseteq V\setminus\{r\}.
\end{aligned}$$

In such form the k-MST problem is almost equivalent to the original superposition tree restoration problem (4). The only difference is the absence of the third constraint on the arity of basic functions (which is equivalent to the constrain on number of edges sourcing in certain vertex).

Definition 3. (PCST, Prize-Collecting Steiner Tree) Denote weighted graph $G=(V,E)$ with root vertex r, and edge weits $w(e_i)=c_i\geqslant 0,\ e_i\in E$, 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 following 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 $\pi(S)=\sum_{v\in S}\pi(v)$.

As well as in the k-MST case, this problem can be generalized to the case of directed graphs. In such form it is named A-PCST (Asymmetric-PCST). The linear programming problem for the A-PCST takes the following form:

$$\begin{aligned}
\underset{\substack{x_e, z_S \\ e\in E, S\subseteq V\setminus\{r\}}}{\text{minimize}} \quad & \sum_{e\in E} c_e x_e + \sum_{S\subseteq V\setminus\{r\}} \pi(S) z_S \\
\text{s.t.} \quad & \sum_{\substack{e\in\delta(S):\\ e=(*,v_i),\ v_i\in\delta(S)}} x_e + \sum_{T:T\supseteq S} z_T \geqslant 1, && \forall S\subseteq V\setminus\{r\}, \\
& \sum_{e\in E:\ e=(*,v)} x_e \leqslant 1, && \forall v\in V, \qquad (6)\\
& x_e\in\{0,1\}, && \forall e\in E, \\
& z_S\in\{0,1\}, && S\subseteq V\setminus\{r\}.
\end{aligned}$$

According to the Karush-Kuhn-Tucker conditions and [16] the third constraint from (5) can be included into the optimized functional. In such case k-MST and A-PCST problems have equivalent constraints and only differ in the optimized functional.

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

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

In case of equivalent prize values $\pi(v) = \lambda$ the only difference is the constant $\lambda(n-k)$. So the optimization problems take the following forms:

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

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

The constant λ stands for non-negative Langrange multiplier in the k-MST problem and for vertex prize in the A-PCST.

So, the original problem of restoring the superposition tree can be reduced to the problem k-MST on a directed graph by releasing the constraints on arities. Then it can be reduced to A-PCST if all prized are the same. Several effective algorithms are present for solving the PCST problem (but not the A-PCST). A possible workaround is releasing the constraints on the graph orientation so the PCST algorithms work and restoring the tree orientation later.

3.2. $(2-\varepsilon)$-approximation algorithm for constrained forest problems

General overview of techniques for constrained forest problems is provided in [13]. Selected results are relevant for this research.

Denote weighted undirected graph $G = (V, E)$, where all weights $w(e_i) = c_i \geqslant 0,\ \forall e_i \in E$. Denote mapping $f : 2^V \to \{0, 1\}$. The following linear programming problem with integer constraints can be stated:

$$\begin{aligned} \underset{x_e:\ e\in E}{\text{minimize}} \quad & \sum_{e\in E} c_e x_e & & \\ \text{s.t.} \quad & x(\delta(S)) \geqslant f(S), & \emptyset \neq S \subset V, & \qquad (7) \\ & x_e \in \{0,1\}, & \forall e \in E, & \end{aligned}$$

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

Assume the mapping f possess the following properties:

(*i*) $f(V) = 0$

(*ii*) [Symmetry] $f(S) = f(V\setminus S)$

(*iii*) [Disjunctivity] $\forall A, B \subset V:\ A \cap B = \emptyset,\ f(A) = f(B) = 0 \to f(A \cup B) = 0$

In this setup f specifies the number of edges which starts in the vertices set S. E.g. for the minimum ideal matching problem $f(S) = 1$ if and only if $|S| \mod 2 = 1$ (since such a set is impossible to divide into pairs).

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

Proof. The Symmetry property leads to $f(V \backslash S) = 0$. Since $V \backslash S \cap B = \emptyset$, the disjunctivity property leads to $f((V \backslash S) \cup B) = 0$. According to the symmetry property, the equation $f\big(V \backslash ((V \backslash S) \cup B)\big) = f(S \backslash B) = 0$ is correct. □

Problems with such description are called optimal forest search problems with correct constraints.

Such problem statement with appropriate mapping 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 belongs to the NP-complete class, so only approximate and heuristic algorithms are applicable.

Definition 4. (α-approximating algorithm) A heuristic polynomial algorithm that delivers a solution for some optimization problem is called α-approximating 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 (7) should be relaxed.

$$\begin{aligned} \underset{x_e:\ e \in E}{\text{minimize}} \quad & \sum_{e \in E} c_e x_e \\ \text{s.t.} \quad & \sum_{e \in \delta(S)} x_e \geqslant f(S), && \emptyset \neq S \subset V, \\ & x_e > 0, && \forall e \in E, \end{aligned} \tag{8}$$

The dual problem will take the following form:

$$\begin{aligned} \underset{y_S:\ \emptyset \neq S \subset V}{\text{maximize}} \quad & \sum_{S \subset V} f(S) y_S \\ \text{s.t.} \quad & \sum_{S:\ e \in \delta(S)} y_S \leqslant c_e, && \forall e \in E, \\ & y_S > 0, && \emptyset \neq S \subset V, \end{aligned} \tag{9}$$

regarding the complementary slackness conditions: $y_S \cdot \big(\sum\limits_{e \in \delta(S)} x_e - f(S) \big) = 0,\ \forall S \subset V$.

Denote the set of vertices $A = \{v \in V : f(\{v\}) = 1\}$. An adaptive greedy $\left(2 - \frac{2}{|A|}\right)$ - approximating algorithm for problems of the form (7) is proposed next. Brief informal description of the algorithms is provided below.

The algorithm consists of two stages. On the first stage it greedily combines clusters of vertices (initially every vertex belongs to its own cluster) increasing the dual variables y_S. If the next edge e reaches equality in the constraints in (9), this edge is added to the set S and the connected clusters will be merged. This stage is similar to Kraskal minimal spanning tree algorithm (moreover, it provides equivalent results with specific f). Opposed to the Kraskal algorithms, the effective weight of the next edge in minimized, not the weight itself. This change makes the algorithm capable to adapt to specific data.

In the second stage some edges are removed from the final set S. If the edge deletion does not violate the constraints, this edge is to be removed.

The pseudo-code for the described algorithm is provided next. Z_{DRLP} stands for Dual-Relaxed-LP.

Algorithm 1: $(2-\varepsilon)$-approximation algorithm for problem (7)

Data: Weighted undirected graph $G=(V,E)$ with non-negative edges' weights $c_i \geqslant 0$; mapping f
Result: Forest F'; optimized in problem functional (7) value Z_{DRLP}
Stage 1, Merging
begin
  $F \longleftarrow \emptyset$
  $Z_{\text{DRLP}} \longleftarrow 0$
  $\mathcal{C} \longleftarrow \{\{v\} : v \in V\}$
  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
        $d(v) \longleftarrow d(v) + \varepsilon(e^*) \cdot f(C)$
    $Z_{\text{DRLP}} \longleftarrow Z_{\text{DRLP}} + \varepsilon(e^*) \sum\limits_{C\in\mathcal{C}} f(C)$
    $\mathcal{C} \longleftarrow \mathcal{C}\backslash\{C_p\}\backslash\{C_q\} \cup \{C_p \cup C_q\}$ (e^* connects components (clusters) 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 — connected component

The initial value of $F \longleftarrow \emptyset$ in 1 is equivalent to the assumption $x_e = 0 \quad \forall e \in E$. According to the slackness conditions $y_S = 0, \ \forall \emptyset \neq S \subset V$.

At any step of the algorithm, cluster $\mathcal{C}$ can be divided into 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 (9) as follows

$$d(i) = \sum_{S: i\in S} y_S.$$

This statement can be easily proved by induction.

Let's take a closer look at the 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 $\sum\limits_{S:\ e\in\delta(S)} y_S \leqslant c_e$. In terms of $d(v)$, this condition takes the form:

$$\sum_{S:\ e\in\delta(S)} y_S = d(v_1) + d(v_2),\ e = (v_1, v_2),$$

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 by ε leads to the following equation

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

which leads to the formula used in line 10 of the algorithm 1 a. In the case when the next edge is included into 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 addition of new active components are removed on the second stage of the algorithm.

The following lemma defines useful properties of connected components in F'.

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

Proof. 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 $f(N_i) = 0\ \forall i$. According to disjunctive property $f\big(\bigcup\limits_{i=1}^{|\delta(S)|} N_i\big) = 0$. Hence, according to Lemma 1 $f(N) = f\big(V\backslash \bigcup\limits_{i=1}^{|\delta(S)|} N_i\big) = 0$. □

The following theorem states that the solution derived by the described algorithm is meeting the constraints of the original linear programming problem. Theorems' proofs are available in the supplementary materials.

Theorem 1. The edge set F' derived by algorithm 1 meets all the constraints of the original problem (7).

The following theorem describes the properties of Algorithm 1.

Theorem 2. Algorithm 1 is an α-approximate algorithm for problem (7) with $\alpha = 2 - \frac{2}{|A|}$ where $A = \{v \in V : f(\{v\}) = 1\}$.

Despite the provided theoretical basis, the is no appropriate function f to state the PCST problem as referenced in 7. But the Algorithm 1 can be modified to work in these conditions.

3.3. Upgraded problem statement for PCST problem

The relaxed form of the linear programming problem PCST can be described as follows (equivalent to the A-PCST problem statement):

$$\begin{aligned}
\underset{\substack{x_e, s_v \\ e\in E, v\in V\backslash\{r\}}}{\text{minimize}} \quad & \sum_{e\in E} c_e x_e + \sum_{v\in V\backslash\{r\}} (1 - s_v)\pi_v \\
\text{s.t.} \quad & \sum_{e\in\delta(S)} x_e \geqslant s_v, && \forall S \subseteq V\backslash\{r\},\ v \in S, \qquad (10)\\
& x_e \geqslant 0, && \forall e \in E, \\
& s_v \geqslant 0, && \forall v \in V\backslash\{r\}.
\end{aligned}$$

This problem statement is different from the original one (6), but it makes possible to align the k-MST problem with the current one. The s_v can be treated as indicators that vertex v is included in the tree.

The dual problem takes form:

$$\begin{aligned}
\underset{y_S:\ S\subset V\backslash\{r\}}{\text{maximize}} \quad & \sum_{S\in V\backslash\{r\}} y_S \\
\text{s.t.} \quad & \sum_{S: e\in\delta(S)} y_S \leqslant c_e, && \forall e \in E, \qquad (11)\\
& \sum_{S\subseteq T} y_S \leqslant \sum_{v\in T} \pi_v, && \forall T \subset V\backslash\{r\}, \\
& y_S \geqslant 0, && \forall S \subset V\backslash\{r\}.
\end{aligned}$$

The algorithm to solve this problem 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 the both groups of constraints).

The pseudo-code for this algorithm is provided next.

The λ function can be recognized as indicator of the fact, that the new component is active (and is similar to thr f function in the Algorithm 1. The approximation properties of the upgraded algorithm can be found in [13]).

Theorem 3. Algorithm 2 is and α-approximate algorithm for PCST problem with $\alpha = 2 - \frac{2}{n-1}$, where n stays for the number of vertices in the graph G.

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 $\varepsilon_2(C) = \sum_{i \in C} \pi_i - w(C)$
  - $\varepsilon = \min(\varepsilon_1(e^*), \varepsilon_2(C^*))$
  - 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 following constraints:

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

---

4. COMPUTATIONAL EXPERIMENT

4.1. Description of the used algorithms

4.1.1. DFS Greedy tree depth-first traverse (traversing the edges with highest weights is equivalent to selecting the most probable path). The traverse stops when the number of edges starting within the vertex equals the arity of the function.

4.1.2. BFS Greedy tree breadth-first traverse (traversing the edges with highest weights is equivalent to selecting the most probable path). The traverse stops when the number of edges starting within the vertex equals the arity of the function.

4.1.3. Prim's algorithm This algorithm searches for a minimum spanning tree for a graph with additional constraints on the arity of the used functions. This constraints are to find the minimum weght edge. After the vertex addition, all edges targeting this vertex are excluded to preserve direction of the tree. If the number of edges starting in some vertex exceeds the corresponding arity, all edges starting in this vertex are excluded from the set of possible edges.

The main advantage of this algorithm is coming from its independence of the traverse procedure. In case of small noise (like stated above) this algorithm is able to restore the superposition tree without errors. The main disadvantage is the failure of the algorithm if one of the variables is used several times.

Algorithm 3: Superposition tree restoration with Prime'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 $i = from$

4.1.4. Algorithms based on PCST First the matrix should be transformed to the undirected form. Last column from matrix M is dropped resulting in matrix M'. The PCST algorithm takes the adjacency matrix $1 - (M' + M'^T)/2$ with prize value 0.5 for every vertex.

The subtraction of 1 in this case is the element-wise operation.

The prize value is equal to 0.5 because with smaller values the tree will be truncated (due to the noise 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 can be restored with one of the described algorithms. In addition, the results of the PCST based approach can be used as prior for other algorithms, e.g.

$$M' \longleftarrow (M'_{pcst} + M')/2,$$

so the PCST results are used as "recommendations" for other algorithms.

Algorithm 4: Superposition tree restoration 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}
  Restore the tree from M_{pcst} with some traverse procedure from the root vertex to derive M_{res}

Additional analysis has been performed on algorithms in case when the PCST problem input was non-symmetric matrix M'. From some point of view such approach is close to the Prim's algorithm. The whole list of 11 compared algorithms is provided below.

- DFS
- BFS
- Prim's algorithm
- k-MST via PCST
- k-MST + DFS
- k-MST + BFS
- k-MST + Prim's algorithm
- k-MST via PCST, directed
- k-MST + DFS, directed
- k-MST + BFS, directed
- $k - MST$ + Prim's algorithm, directed

Last four approaches are the cases where PCST problem solution is using the directed graph.

4.2. Data description and generation procedure

Data generation was performed with the following assumptions:

- The arities of the function are generated by Binomial distribution (so there are many functions with small arity).
- All the functions are taking only one input.

The main goal is to restore the correct superposition tree. All cases with partial restoration are treated as errors.

$$\mathrm{Acc}(\mathrm{R},\mathrm{N},\mathcal{M}) = \frac{1}{|\mathcal{M}|}\sum_{M\in\mathcal{M}}\left[R(N(M)) = M\right],$$

where N is the noise function and R is the restoration algorithm.

- 50 sets of arity values (with length from 5 to 20)
- 20 function for every set
- 5 noised variants of every function
- Uniformly distributed noise
- Linear calibration in segment $[0, 1]$

Remark. Denote α as the maximum noise value. The noised matrix $N(M)$ is generated as

$$N(M) = Cal\big(M + U(M.\mathrm{size}, [-\alpha, \alpha])\big),$$

where $U(M.size, [-\alpha, \alpha])$ returns the matrix with the same shape where every element is independent variable from the Uniform distribution on the $[-\alpha, \alpha]$ segment. $Cal(M)$ stands for the linear min max calibration in the segment $[0, 1]$ such that

$$Cal(M) = \frac{M - \min(M)}{\max(M) - \min(M)}.$$

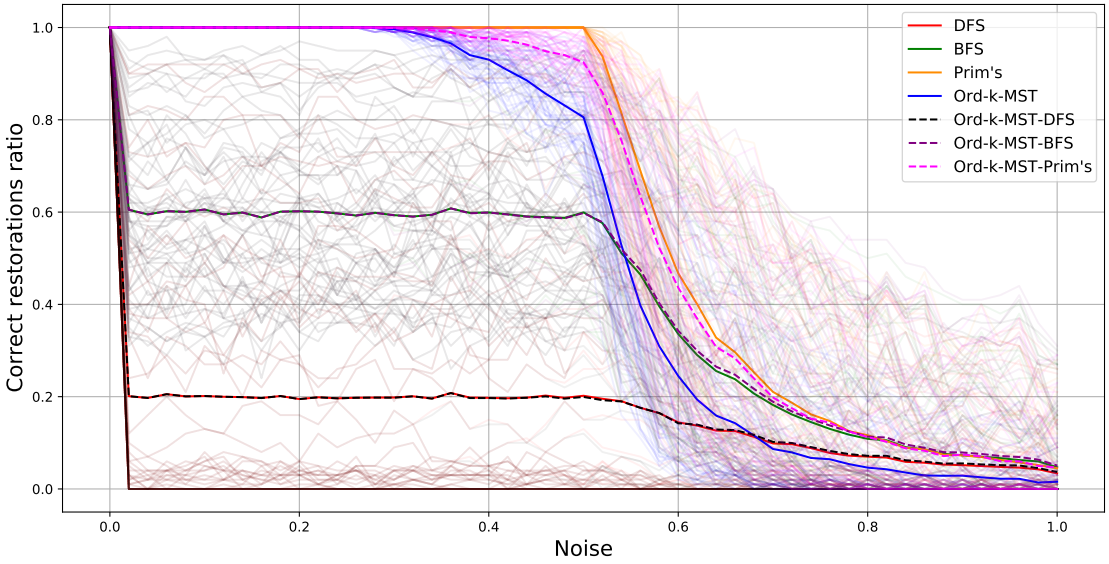

Figure 2. Ratio of correct restoration with different noise values. Arity $\in [5; 20]$, k-MST based algorithms use symmetric adjacency matrix.

Minimum and maximum values are global for the whole matrix, all arithmetical operations are element wise.

4.3. Results

Results for undirected case of functions with arities 5 to 20 are illustrated on the Picture 2.

The Fig. 3 focuses on the algorithms performance with noise close to 0.5 threshold. The best results are delivered by the Prim's algorithm. The second best solution is based on the BFS. Table 3 accompanies the Fig. 3.

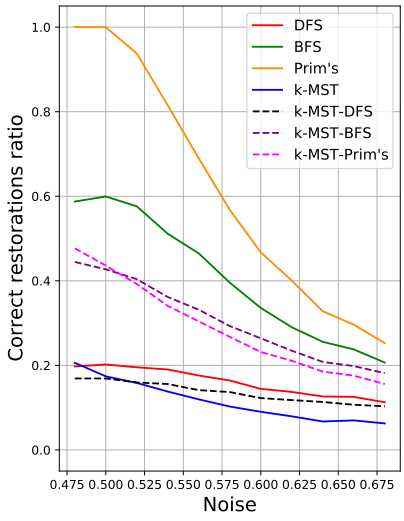

Figure 3. Quality of the algorithms on the border; small arities; Uniform noise; unordered input

Noise	.50	.52	.54	.56	.58
DFS	.2	.2	.19	.18	.16
BFS	.6	.58	.51	.46	.4
Prim's algorithm	1.0	.94	.81	.69	.57
k-MST	.17	.16	.14	.12	.1
k-MST-DFS	.17	.16	.16	.14	.14
k-MST-BFS	.43	.4	.36	.33	.29
k-MST-Prim's	.44	.39	.34	.33	.27

Table 3. Quality with noise ~ 0.5

The ordered versions of the k-MST based algorithms are illustrated on Fig. 5. Prim's algorithm and k-MST show much closer restoration ratios. Algorithms based on k-MST deliver significantly better results. Algorithms based on k-MST and using BFS, DFS or Prim's algorithm to restore the original matrix show achieve results close to the sole results of BFS, DFS and Prim's algorithm accordingly.

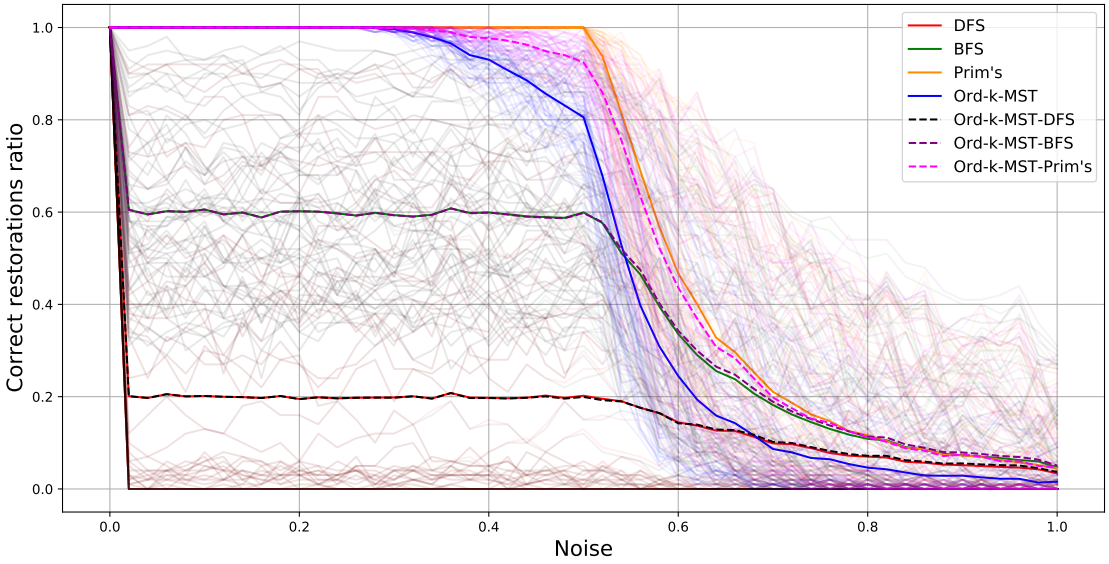

Figure 5. Quality of the algorithms on the border; arities $\in [5; 20$; Uniform noise; ordered input

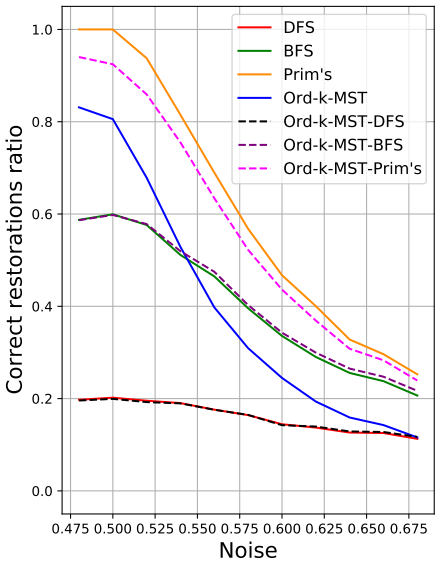

Figure 4. Ratio of correct restoration with different noise values. Arity $\in [5; 20]$, k-MST based algorithms use original adjacency matrix.

Noise	.50	.52	.54	.56	.58
DFS	.2	.2	.19	.18	.16
BFS	.6	.58	.51	.46	.4
Prim's algorithm	1.0	.94	.81	.69	.57
Ord-k-MST	.81	.68	.53	.4	.31
Ord-k-MST-DFS	.2	.19	.19	.18	.16
Ord-k-MST-BFS	.6	.58	.52	.47	.4
Ord-k-MST-Prim's	.92	.86	.76	.63	.52

Table 4. Ratio of correct restorations with noise ~ 0.5

Fig. 4 and Table 4 illustrate the algorithms performance with noise close to 0.5.

Table 6 illustrates the time complexity of the described algorithms. The measurements were performed using $51 \cdot 50 \cdot 20 \cdot 5 \approx 250000$ matrix restorations.

5. CONCLUSION

This paper compares different approaches to symbolic regression problem based on restoration of the superposition matrix. The approach based on Prim's algorithm delivers the most accurate results and is the most resistant to small noise in data.

The proposed algorithm delivers accurate results, but is more prone to noise in the superposition matrix.

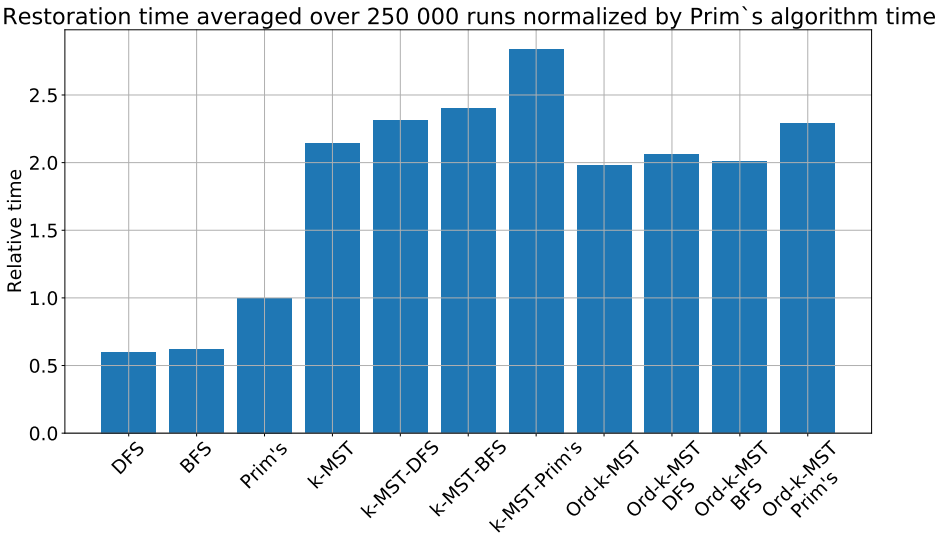

Figure 6. Time comparison. All scores are normalized by Prim's algorithm elapsed time.

Approaches based on BFS and DFS are unable to restore the original superposition if noise is present. PCST algorithm with BFS used for superposition matrix restoration shows mediocre results.

Acknowledgments. The reported study was funded by RFBR according to the research projects 20-37-90050, 19-07-01155 and NTI project 13/1251/2018.

REFERENCES

1. L. D. Davis, Handbook Of Genetic Algorithms (Van Nostrand Reinhold; 1st Edition, 1991).
2. J. R. Koza “"Genetic programming as a means for programming computers by natural selection",”Statistics and computing 4 (2) 87–112 (1994).
3. D. P. Searson, D. E. Leahy, M. J. Willis, “"GPTIPS: an open source genetic program- ming toolbox for multigene symbolic regression",”Proceedings of the International multiconference of engineers and computer scientists, 1, 77–80 (2010).
4. D. P. Searson, “GPTIPS 2: an open-source software platform for symbolic data mining,”Handbook of genetic programming applications, 551–573 (2016).
5. K. O. Stanley, R. Miikkulainen, “Evolving neural networks through augmenting topologies,”Evolutionary computation 10 (2), 99–127 (2002).
6. A. Gaier, D. Ha, “Weight Agnostic Neural Networks,”Advances in Neural Information Processing Systems 32, 5364–5378 (2019).
7. A. M. Bochkarev, I. L. Sofronov, V. V. Strijov, “Generation of expertly-interpreted models for prediction of core permeability,”Systems and Means of Informatics 27 (3), 74–87 (2017).
8. D. Lozovanu, A. Zelikovsky, “Minimal and bounded tree problems,”ezele Congre- sului XVIII al Academiei Romano-Americane, 25–26 (1993).
9. R. Ravi, R. Sundaram, M. V. Marathe et al., “Spanning trees — short or small,”SIAM Journal on Discrete Mathematics 9 (2), 178–200 (2015).
10. F. A. Chudak, T. Roughgarden, D. P. Williamson, “Approximate k-MSTs and k- Steiner trees via the primal-dual method and Lagrangean relaxation,”Mathematical Programming, 100 (2) 411–421 (2004).
11. B. Awerbuch, Y. Azar, A. Blum, S. Vempala, “New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen,”SIAM Journal on computing, 28 (1), 254–262 (1998).
12. S. Arora, G. A. Karakostas, “2 + ε approximation algorithm for the k-MST problem,”Mathematical Programming, 107 (3), 491–504, (2006).
13. M. X. Goemans, D. P. Williamson, “A general approximation technique for constrained forest problems,”SIAM Journal on Computing, 24 (2), 296–317 (1995).
14. C. Hegde, P. Indyk, L. Schmidt, “A fast, adaptive variant of the Goemans-Williamson scheme for the prize-collecting Steiner tree problem,”Workshop of the 11th DIMACS Implementation Challenge, (2014).
15. `https://github.com/fraenkel-lab/pcst_fast`
16. C. Ras, K. Swanepoel, D. A. Thomas, “Approximate Euclidean Steiner Trees,”Journal of Optimization Theory and Applications, 172 (3), 845–873 (2017).

6. SUPPLEMENTARY MATERIALS

Theorem 1

Proof. Assume there is empty set $\emptyset \neq S \subset V$, such that $\sum\limits_{e \in \delta(S)} x_e < f(S)$. 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 $\sum\limits_{e \in \delta(S)} x_e = 0$). According to Lemma 2 and disjunctive property, $f(S) = f\big(\bigcup\limits_j C_{i_j}\big) = 0$, which is contrary to the original assumption $f(S) = 1$. □

Theorem 2

Proof. Recall the following inequality:

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

- The first part is correct according to the Z^*_{LP} definition and the fact that F' meets the constraints (according to Theorem 1).
- The last part is correct because for the optimal solution Z^*_{LP} of Problem 7 the following inequality holds (due to the below constraints of the dual problem): $Z_{\text{DRLP}} = \sum_{S \subset V} y_S \leqslant Z^*_{RLP} \leqslant Z^*_{LP}$.
- 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:

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

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

$$\sum_{S \in \mathcal{C}_i} d(S) \geqslant 2|\mathcal{C}_i|, \tag{12}$$

which implies

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

which is correct because the number of clusters does not increase through time, or equivalently $|A| \geqslant |\mathcal{C}_a|$.

The following lemma will prove the inequality (12).

Lemma 3. Denote graph H where every vertex is corresponding 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.

Proof. Recall that lone inactive vertices are ignored due to the zero power of every inactive vertex, so they all can be subtracted in the inequality $\sum\limits_{S\in\mathcal{C}} d(S) \leqslant 2(|\mathcal{C}_a| + |\mathcal{C}_i| - 1)$.

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 $f(C_i) = 0 \forall i \neq v$ (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 power of every inactive vertex is greater or equal to two (or equal to zero, but such vertices do not affect the target inequality). □

According to the Lemma 3 proves the inequality (12), so the second part of the original statement is also correct. □