Министерство науки и высшего образования
Российской Федерации
«Московский физико-технический институт
(государственный университет)»
Физтех-школа прикладной математики и информатики
Факультет управления и прикладной математики
Кафедра «Интеллектуальные системы»

Шибаев Иннокентий Андреевич

Прогнозирование оптимальных суперпозиций в задачах регрессии

03.03.01 — Прикладные математика и физика

Выпускная квалификационная работа бакалавра

Научный руководитель:
д.ф.-м.н. Стрижов Вадим Викторович

Москва
2018

АННОТАЦИЯ

Данная работа посвящена методам решения задачи символьной регрессии на основе алгоритмов поиска деревьев минимального веса во взвешенных графах с покрашенными вершинами. Символьная регрессия используется для порождения интерпретируемых моделей. Задача символьной регрессии ставится в форме задачи восстановления матриц смежности графов. Проблемой является шум, возникающий при прогнозировании матриц смежности, и не позволяющий интерпретировать их как матрицы смежности графов суперпозиций. Целью исследования является разработка и сравнение качества работы различных методов восстановления графов суперпозиций функций из их зашумленного матричного описания.

Исследуется алгоритм, получаемый после сведения исходной задачи к задаче собирающего призы дерева Штейнера ($PCST$). Для этого задача была переформулирована в терминах k-минимального остовного дерева (минимальное остовное дерево на минимум k-вершинах). Задача k-MST в постановке задачи целочисленного программирования после релаксации совпадает с постановкой задачи $PCST$ при равных призах за вершины. В результате был получен алгоритм восстановления графов суперпозиций на основе $(2-\varepsilon)$-аппроксимирующего алгоритма решения задачи $PCST$.

Полученный алгоритм сравнивается с другими алгоритмами восстановления деревьев на синтетических данных.

Содержание

1 Введение

Символьная регрессия — один из методов решения задач регрессии заключающийся в нахождении функции, наилучшим образом приближающей данные, путем перебора суперпозиций функций из некоторого набора базовых. Такие модели достаточно хорошо интерпретируются, что особенно важно, к примеру, в задачах меддицинской диагностики.

Как и любая переборная задача, символьная регрессия имеет высокую вычислительную сложность. Для решения задач переборного типа часто применяют генетические алгоритмы [1], а задача символьной регрессии является основным примером использования т.н. генетческого программирования [2]. Также существуют подходы на основе генетического программирования с ограничениями класса функций до линейных комбинаций базовых, к примеру [3], [4].

Другой подход к решению задачи символьной регрессии основан на использовании матричного представления [5] деревьев суперпозиции (деревьев вычисления). Тогда задача сводится к прогнозированию матриц суперпозиции алгоритмами нечеткой классификации (каждый элемент матрицы суть отдельный класс, принадлежность к которому надо проверить). Но тогда возникает проблема с тем что спрогнозированная матрица может не быть матрицей суперпозиции (с точки сохранения арностей функций, т.е. ограничений на степени полуисхода вершин, а также на ориентированность дерева). Если же использовать предсказание вероятностей то возникает большое количество шумовых ребер малого веса (малой вероятности).

В данной работе сравниваются методы восстановления матрицы суперпозиции по спрогнозированной матрице вероятностей ребер. Для этого задача в начале сводится к NP задаче построения дерева минимального веса с фиксированным корнем на минимум k вершинах (k-MST), поставленную в [6] (доказательство принадлежности к классу NP сложных) и [7] (алгоритмы для частных случаев, а также некоторые аппроксимирующие оптимальное решение алгоритмы). Как показано, к примеру, в [8] задача k-MST может бы сведена к задаче Prize-Collecting Steiner Tree ($PCST$) в смысле равенства их релаксированных постановок задач целочисленного ЛП.

Для задачи k-MST известно большое количество алгоритмов дающих различные аппроксимации (по функционалу) оптимального решения, к примеру [7], [9] и [10]. У последнего (как и у многих новых) в основе лежит классический алгоритм решения задачи $PCST$ с фактором $2 - \varepsilon$ описанный в статье [11].

Ориентированное дерево (зная корень) можно восстановить по неориентированному однозначно, то можно использовать алгоримт решения задачи $PCST$ в формулировке для неориентированных графов, а затем восстановить искомую матрицу суперпозиции по результатам его работы. В данной работе используется реализация алгоритма [12] построенная на основе [13] (код приведен в [14]). Полученный алгоритм восстановления деревьев суперпозиции сравнивается с другими (на основе DFS, BFS а также на основе алгоритма Прима) на синтетических данных.

2 Постановка задачи

2.1 Постановка задачи символьной регрессии

Решается задача регрессии. Дан некоторый набор выборок $\mathcal{A} = \{\mathbf{A}_1, \ldots, \mathbf{A}_N\}$, $\mathbf{A}_i = (\mathbf{X}_i, \mathbf{y}_i)$ где $\mathbf{X}_i$ — признаковое описание n_i объектов i-й выборки. а $\mathbf{y}_i$ - ответы на них. Для каждой пары A_i также задана некоторая f_i — суперпозиция базовых функций являющаяся порождающей функцией этой выборки (т.е. $f_i(\mathbf{X}_i) = y_i$).

Также предполагается, что данные однородны в том смысле, что $f_i \in \mathcal{F}$ — семейство порождающих функций m переменных и $\mathcal{F} = \{f : f = sup(g_1, \ldots, g_l)\}$ где $sup(g_1, \ldots, g_l)$ — суперпозиция l заданных базовых функций $g_1, \ldots, g_l$.

Для пары $\mathbf{A} = (\mathbf{X}, \mathbf{y})$ требуется найти суперпозицию f^* наилучшим образом приближающую данную выборку

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

где S — заданная функция ошибки, w^* - оптимальный набор параметров для модели f при заданных $\mathbf{X}, \mathbf{y}$:

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

В качестве функции потерь S будем использовать разность квадратов регрессионных остатков:

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

2.2 Метод решения

Рассмотрим отображение $h : \mathcal{A} \to \mathcal{F}$ которое дает решение (1). Тогда можно переписать задачу в виде

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

где H - некоторое параметрическое множество допустимых отображений. Теперь воспользуемся тем, что для каждого $\mathbf{A}_i$ нам также дана функция f_i порождающая эту выборку: $|f_i(\mathbf{X}_i) - \mathbf{y}_i|_2 = 0$. Заменяя, получим

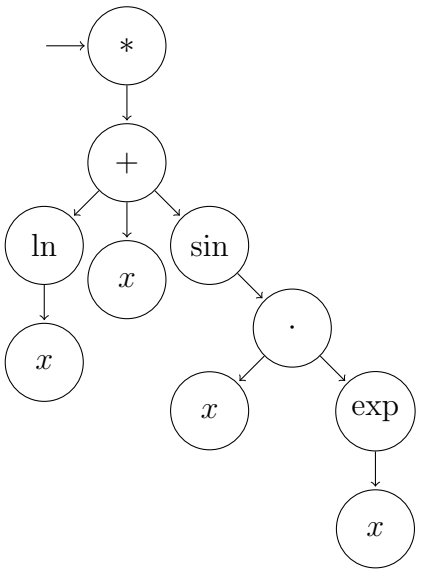
$$h^* = \arg\min_{h \in H} \sum_{i=1}^{N} \| \left(h(\mathbf{A}_i) - f_i\right)(\mathbf{X}_i)\|_2^2, \tag{2}$$

т.е. задача может быть сформулирована как поиск наилучшей в смысле функционала выше аппроксимации $h(A_i)$ суперпозиции f_i.

2.3 Формат описания суперпозиции

Для задания суперпозиции используется матричное представление ее графа вычисления, схожее с тем что используется в [5].

Пример 1. Рассмотрим функцию $f(x) = \ln(x) + x + \sin(x \cdot e^x)$. Для данной функции построим граф вычисления и выпишем его матрицу смежности (вершина x в данном случае присутствует в единственном экземпляре и продублирована в графе лишь для удобства записи)



$arity$	$f(.)$	$*$	$+$	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

При таком описании мы можем однозначно восстановить исходную суперпозицию зная ее матрицу смежности а также набор функций соответствующих вершинам (кроме последней вершины соответствующей переменной). ■

Далее в работе рассмтариваются суперпозиции задаваемые следующим образом:

- В начале задается набор порождающих функций, а так же набор арностей этих функций. К этому набору добавляется специальная функция $* = *(x) : *(x) = x$ — унарная функция задающая корень дерева.
- Затем рассматривается граф на вершинах соответствующих функциям из набора заданного на предыдущем шаге. На этом графе задается ориентированное дерево подвешенное за вершину $*$ (т.е. с корнем в этой вершине).
- Данное дерево должно удовлетворять следующему свойству: если в нем есть ребро e_{ij} из вершины i в вершину j то из вершины j должно выходить не менее a_j ребер где a_j — арность функции соответствующей j-й вершине. Если из вершины выходит меньше ребер, то оставшиеся считаются проведенными в вершину сооветствующую переменной.

Замечание. В данной постановке рассматривается лишь случай одной переменной. К нескольким переменным переход может быть сделан одним из двух способов:

- Можно просто расширить множество добавочных столбцов (по числу новых переменных). В таком варианте становится сложнее разбирать случаи, когда в функцию несколько раз входит одна и та же переменная ($f(x, y, x)$), но он не так сильно увеличивает сложность обучения.
- Другой способ заключается в том чтобы задавать то какие переменные использовать вместе с функциями, т.е. использовать функции вида $f(\mathbf{x}) = f(x_1, x_5)$, задавая то, какие именно переменные из набора $\mathbf{x}$ должны быть использованы именно в этой функции. Это может существенно расширить количество функций (на каждый поднабор переменных), но проще при реализации.

2.4 Постановка задачи символьной регрессии в матричной форме

В матричном представлении (2) переписывается в виде

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

где $M(f)$ возвращает матричное представление функции, а $F(m)$ возвращает функцию по матричному представлению.

Даже при малом отличии в матрицах суперпозиции функции восстановленные по этим матрицам могут отличаются очень сильно (в интегральном смысле что представлен выше), так что вместо предыдущей постановки будем рассматривать постановку корректного восстановления матричного представления

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

В данной работе рассматривалось представление $h(A_i) = F(R(P(A_i)))$ где отображение $P : \mathbf{X} \times \mathbf{y} \to \mathbf{M}$ — отображение из множества пар $(\mathbf{X}, \mathbf{y})$ во множество матриц такого же размера как матрицы $(M(f_i))$ рассматриваемых суперпозиций (P из-за $Predict$) и таких что для матрицы $M = P(\mathbf{A})$ любой ее элемент $M_{ij} \in [0, 1]$; второе отображение $R : M \to M$ дает корректную (в смысле соблюдения арностей) матрицу суперпозиции (R из-за $Restore$), а F определена выше.

Таким образом постановка корректного восстановления может быть записана в форме

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

2.5 Задача восстановления дерева суперпозиции

В данной работе исследуются алгоритмы R восстановления корректных матриц суперпозиции из матриц $P(\mathbf{A}_i)$. Задача восстановления матрицы суперпозиции ставится следующим образом:

Определение 1. *(Задача восстановления дерева суперпозиции)*

Дан ориентированный взвешенный граф $G = (V, E)$ с покрашенными вершинами и выделенной вершиной r, при этом веса $w(e_i) = c_i \in [0, 1]$, $e_i \in E$ и цвета вершин $t(v_i) = t_i \in \mathbb{N}$.

Построить ориентированное подвешенное за r дерево минимальноого веса накрывающее в этом графе как минимум k вершин так, чтобы полустепень исхода вершины v_i (число исходящих из нее ребер) после накрытия была меньше либо равна t_i (для корневой вершины r $t_r = 1$).

Эти же условия можно записать в форме задачи линейного программирования с

целочисленными ограничениями

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

где $x_e = 1$ если ребро e взято в ответ, $z_S = 1$ только для множества S состоящего из всех вершин что в ответ (в оптимальное дерево) не вошли. Обозначение $e = (*, v)$ значит что ребро заканчивается в вершине v, откуда оно идет нам не интересно. Аналогично для $e = (v, *)$.

Первое условие задает структуру искомого множества — дерево с корнем в r.

Второе говорит об ориентированности (есть не более одного входящего ребра для каждой вершины). Третье задает условия на арности функций.

Наконец четвертое условие говорит о том что в дереве должно быть не меньше k вершин.

Замечание. При неотрицательных весах условие «не менее k вершин» может быть заменено на «ровно k вершин». Однако нам нужна именно такая форма определения чтобы показать его связь с другими задачами. То же касается и в целом неестественности задания условий в задаче (4).

3 $PCST$-алгоритм восстановления матриц суперпозиции

3.1 Сведение к k-MST и $PCST$

Приведем формулировки задач k-MST и $PCST$:

Определение 2. *(k-MST, k-minimum spanning tree)*

Дан взвешенный граф $G = (V, E)$ с выделенной вершиной r, при этом веса $w(e_i) = c_i \geqslant 0,\ e_i \in E$.

Построить дерево минимального веса с корнем в r накрывающее в этом графе не менее k вершин.

Аналогичным образом можно ввести такую же задачу для ориентированных графов — тогда надо построить ориентированное дерево подвешенное за вершину r. Выпишем задачу линейного программирования для ориентированного варианта k-MST:

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

Легко видеть, что эта задача почти идентична задаче восстановления дерева суперпозиции (4), за исключением отсутствия третьего набора ограничений на число выходящих ребер (ограничений на арности функций).

Теперь сформулируем задачу $PCST$:

Определение 3. *($PCST$, Prize-Collecting Steiner Tree)*

Дан взвешенный граф $G = (V, E)$ с выделенной корневой вершиной r, при этом веса $w(e_i) = c_i \geqslant 0,\ e_i \in E$ и $\pi(v_i) = \pi_i \geqslant 0,\ v_i \in V$ — стоимости (призы) вершин.

Построить дерево T с корнем в r минимизирующее функционал

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

где $x_e \in \{0,1\},\ x_e = 1$ если ребро $e \in E$ взято в дерево T, $z_S \in \{0,1\},\ z_S = 1$ только для множества вершин $S = V\setminus V(T)$ (т.е. множества не взятых в дерево T вершин) и $\pi(S) = \sum_{v\in S} \pi(v)$.

Интуитивно: мы ищем дерево максимизирующее выигрыш (т.е. сумму призов взятых вершин минус сумму стоимостей ребер что нам потребовалось для этого провести). Как и для k-MST для этой задачи существует постановка для ориентированных графов. В таком виде эта задача имеет название A-$PCST$ ($Asymmetric$-$PCST$).

Выпишем задачу ЛП для $A\text{-}PCST$:

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

Теперь воспользуемся ККТ и следуя [15] перекинем третье условие из (5) в функционал. Тогда набор условий у первой и второй задачи станет одинаковым, отличатся будут лишь функционалы:

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

и при $\pi(v) = \lambda$ (равных призах за каждую вершину) получим, наконец, что функционалы отличаются лишь константной добавкой $-\lambda(n-k)$:

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

Отличие в том что λ в первом функционале это неотрицательный множитель Лагранжа, а во втором это константа, равная призу за каждую добавленную вершину.

Итак, исходная задача восстановления дерева суперпозиции отбрасыванием ограничений на арности может быть сведена к задаче $k\text{-}MST$ на ориентированном графе, а ее можно свести к $A\text{-}PCST$ с одинаковыми призами на ребрах. Существуют эффективные алгоритмы решения задачи $PCST$ (не $A\text{-}PCST$) о которых пойдет речь в следующей части. Предлагается отбросить ограничения ориентированности, и восстановить ориентацию в дереве после восстановления дерева в неориентированном графе.

Для этого от матрицы M (зашумленной матрицы суперпозиции), мы переходим к матрице M' (где «$'$» означает отбрасывание столбца соответствующего переменной),

а далее переходим к неориентированному графу рассматривая матрицу $\frac{M'+M'^T}{2}$. После восстановления в нем дерева (неориентированного) с корнем в r используя алгоритмы для решения задачи $PCST$ останется лишь восстановить ориентацию в этом дереве.

Утверждение. Пусть есть матрица M ориентированного графа суперпозиции и ее слабозашумленный вариант $N(M)$. Исходную матрицу можно восстановить по $N(M)$ в два шага, в начале восстановив само дерево по матрице $\frac{N(M)'+N(M)'^T}{2}$, а затем ориентировав его (это делается однозначно так как нам известен корень дерева).

Здесь «слабозашумленный» значит, что $\|M - N(M)\|_\infty < 0.5$ (тогда в матрице $\frac{N(M)'+N(M)'^T}{2}$ можно восстановить дерево просто правильно подобрав разделитель).

Здесь 0.5 берется из следующих рассуждений: рассмотрим незашумленную матрицу M. Тогда $\|\frac{M'+M'^T}{2}\| \leqslant 0.5$. Пусть $\|M - N(M)\|_\infty = 0.5$, тогда может найтись элемент матрицы $0 \to 0.5$ (т.е. в начале он был равен 0, а после зашумления стал равным 0.5) и такой же, симметричный ему, значит при восстановлении отделить реальные вершины от шумовых уже будет невозможно.

В следующем пункте будет рассмотрен общий подход к решению задач типа $PCST$.

3.2 $(2-\varepsilon)$-аппроксимационный алгоритм для задач поиска оптимальных лесов с ограничениями

В статье [11] рассматривается общий алгоритм для задач поиска оптимальных лесов с ограничениями. Приведем некоторые результаты оттуда.

Рассмотрим взвешенный неориентированный граф $G = (V, E)$. Пусть для весев ребер $w(e_i) = c_i \geqslant 0,\ \forall e_i \in E$. Пусть так же задана некоторая функция $f : 2^V \to \{0, 1\}$. Рассмотрим следующую постановку задачи целочисленного ЛП:

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

где $x(\delta(S)) = \sum\limits_{e\in\delta(S)} x_e$, и $x_e = 1$, как и в предыдущих задачах с целочисленными ограничениями, означает что ребро e взято в множество. $\delta(S)$ — это все ребра из E такие, что один и только один из их концов принадлежит S.

Пусть функция f обладает следующими свойствами:

(i) $f(V) = 0$

(ii) [Симметричность] $f(S) = f(V \backslash S)$

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

Замечание. f можно понимать как функцию, задающую то сколько ребер должно идти из множества наружу. К примеру для задачи минимального идеального паросочетания $f(S) = 1$ тогда и только тогда когда $|S| \mod 2 = 1$ (т.к. такое множество на пары разбить очевидно невозможно).

Лемма 1. *Пусть* $B \subseteq S \subset V$. *Если* $f(S) = 0$ *и* $f(B) = 0$ *то* $f(S \backslash B) = 0$.

Доказательство. По свойству симметричности $f(V \backslash S) = 0$. Так как $V \backslash S \cap B = \emptyset$ то по свойству дизъюнктивности получаем, что $f((V \backslash S) \cup B)$. Опять же применяем симметричность и получаем, что $f\left(V \backslash ((V \backslash S) \cup B)\right) = f(S \backslash B) = 0$ ■

Класс задач, что можно описать таким способом называется задачами поиска оптимального леса с корректными ограничениями.

Оказывается, что правильно подбирая функцию f можно свести к такой постановке большое количество других известных задач на взвешенных графах: поиск минимального остова, st-пути, задача Штейнера о минимальном дереве и другие. Последняя задача принадлежит классу NP-полных, так что далее речь пойдет об эврестических алгоритмах поиска решения. Нам понадобится следующее

Определение 4. *(α-аппроксимирующий алгоритм) Эвристический полиномиальный алгоритм, дающий решение некоторой оптимизационной задачи называется α-аппроксимирующим если он гарантирует нахождение удовлетворяющего ограничениям решения данной оптимизационной задачи с фактором не более α (т.е. решение отличается от оптимального не более чем в α раз по функционалу).*

Далее будет приведен адаптивный жадный $\left(2 - \frac{2}{|A|}\right)$-аппроксимирующий алгоритм для задач вида (7), где $A = \{v \in V : f(\{v\}) = 1\}$. Для того чтобы объяснить этот алгоритм в начале прорелаксируем ограничения целочисленности в задаче (7)

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

и выпишем двойственную к полученной задаче (8)

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

плюс условия дополняющей нежесткости вида $y_S \cdot \left(\sum\limits_{e \in \delta(S)} x_e - f(S) \right) = 0,\ \forall S \subset V.$

Алгоритм будет состоять из двух частей. В начале он будет жадно объединять кластеры (изначально — просто набор всех вершин), равномерно увеличивая двойственные переменные y_S. Когда для очередного ребра e будет достигаться равенство в ограничениях в (9) это ребро будет добавлятся (и соответствующие кластеры будут объединены). На этом шаге он схож с алгоритмом Краскала построения минимального остовного дерева, более того — при корректном задании f он будет давать точно такой же результат. При этом минимизироваться будет не вес очередного ребра, а его эффективный вес, в чем и проявится адаптивность алгоритма.

На втором же шаге из построенного множества ребер удаляются те, которые можно удалить не нарушив ограничения.

Теперь приведем псевдокод алгоритма (Z_{DRLP} из-за Dual-Relaxed-LP):

Algorithm 1: $(2-\varepsilon)$-аппроксимирующий алгоритм для задачи (7)

Data: Взвешенный неорентированный граф $G=(V,E)$ с неотрицательными весами ребер $c_i \geqslant 0$ и функция f

Result: Лес F' и значение Z_{DRLP} функционала в задаче (7)

1 Шаг 1, наращивание
2 **begin**
3 $F \longleftarrow \emptyset$
4 $Z_{DRLP} \longleftarrow 0$
5 $\mathcal{C} \longleftarrow \{\{v\} : v \in V\}$
6 **foreach** $v \in V$ **do**
7 $d(v) \longleftarrow 0$
8 **end**
9 **while** $\exists C \in \mathcal{C} : f(C) = 1$ **do**
10 $e^* = \underset{\substack{e=(i,j):\\ i\in C_p\in\mathcal{C},\ j\in C_q\in\mathcal{C},\\ C_p\neq C_q}}{\arg\min} \varepsilon(e)$ где $\varepsilon(e) = \frac{c_e - d(i) - d(j)}{f(C_p)+f(C_q)}$
11 $F \longleftarrow F \cup e^*$
12 **foreach** $C \in \mathcal{C}$ **do**
13 **foreach** $v \in C$ **do**
14 $d(v) \longleftarrow d(v) + \varepsilon(e^*) \cdot f(C)$
15 **end**
16 **end**
17 $Z_{DRLP} \longleftarrow Z_{DRLP} + \varepsilon(e^*) \sum\limits_{C\in\mathcal{C}} f(C)$
18 $\mathcal{C} \longleftarrow \mathcal{C}\backslash\{C_p\}\backslash\{C_q\} \cup \{C_p \cup C_q\}$ (e^* соединяет компоненты C_q и C_p)
19 **end**
20 **end**
21 Шаг 2, прореживание
22 $F' \longleftarrow \{e \in F :\ \exists N \in (V, F\backslash\{e\}), f(N) = 1\}$ где N — компонента связности

В строке 3 Алгоритма 1, когда мы задаем $F \longleftarrow \emptyset$ эквивалентно можно сказать что мы полагаем $x_e = 0$ для любого ребра (в начале не берем ни одного ребра во множества). Тогда из условий дополняющей нежесткости следует, что мы автоматически задаем $y_S = 0,\ \forall \emptyset \neq S \subset V$.

На любом шаге алгоритма можно разделить $\mathcal{C} = \mathcal{C}_i \cup \mathcal{C}_a$, где $C \in \mathcal{C}_a$ если $f(C) = 1$ и $C \in \mathcal{C}_i$ иначе. Компоненты из $\mathcal{C}_a$ называются активными компонентами.

Переменные $d(v)$ в данном алгоритме связаны с переменными y_S из (9) следующим образом (можно показать индукцией по основному циклу алгоритма):

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

Теперь рассмотрим две различные компоненты $C_q,\ C_p,\ C_q \cap C_p = \emptyset$ на какой-либо итерации первого шага алгоритма. Мы хотим увеличить все y_S равномерно на некоторое ε не нарушив ограничения $\sum\limits_{S:\ e\in\delta(S)} y_S \leqslant c_e$. Заметим, что в терминах $d(v)$

это условие переписывается в форме

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

т.к. для любого S такого, что $v_1, v_2 \in S$ выполнено $y_S = 0$ потому что компоненты на первом шаге лишь растут. Теперь мы увеличиваем некоторые на ε, и получаем

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

из чего и получается формула что используется в строке 10 Алгоритма 1. Если же очередное ребро попало в компоненту, то это значит что далее $\sum\limits_{S:\ e\in\delta(S)} y_S$ увеличиваться не будет, а значит далее условия будут выполнятся.

На втором шаге алгоритма удаляются те ребра, которые можно убрать из F не добавив активных компонент.

Докажем следующую лемму касающуюся свойств связных компонент в F'

Лемма 2. *Для любой компоненты связности N в F' выполняется $f(N) = 0$.*

Доказательство. F' получается прореживанием F. Значит существует компонента связности $C \in F$ такая, что $N \subseteq C$. При этом $f(C) = 0$ ввиду того что алгоритм остановился. Рассмотрим ребра $\delta(N)$ (ребра что выходили из N до прореживания). Их убрали, значит в $(V, E\backslash\{e\}),\ e \in \delta(N)$ нет компонент для которых значение f равно 1.

Это значит, что если мы рассмотрим то, в какие компоненты распадается C после удаления ребер из $\delta(N)$ (обозначим их $N, N_1, \ldots, N_{|\delta(N)|}$) то $f(N_i) = 0\ \forall i$. Тогда по свойству дизъюнктивности $f\left(\bigcup\limits_{i=1}^{|\delta(S)|} N_i\right) = 0$. Наконец, по Лемме 1 получаем, что $f(N) = f\left(V \setminus \bigcup\limits_{i=1}^{|\delta(S)|} N_i\right) = 0$ ■

Покажем, что данный алгоритм дает корректное (с точки зрения ограничений) решение исходной задачи целочисленного линейного программирования.

Теорема 1. *Набор ребер F' получаемый в Алгоритме 1 является допустимым (удовлетворяет ограничениям) в задаче (7).*

Доказательство. Предположим противное, пусть существует $\emptyset \neq S \subset V$, такое, что $\sum\limits_{e\in\delta(S)} x_e < f(S)$. Из этого сразу следует, что $f(S) = 1$. Рассмотрим все компоненты связности $C_1, \ldots, C_m$ из графа (V, F').

Заметим, что либо $C_i \subseteq S$ либо $C_i \cap S = \emptyset$ т.к. из S не выходит ни одного ребра (т.к. $\sum\limits_{e\in\delta(S)} x_e = 0$). Но тогда $f(S) = f\left(\bigcup\limits_j C_{i_j}\right) = 0$ — последний переход следует из Леммы 2 и дизъюнктивности f. Противоречие с предположением о том что $f(S) = 1$. ■

Заметим, что так как $Z_{DRLP} = \sum_{S\subset V} y_S$ и решение удовлетворяет ограничениям, а функционал двойственной является ограничением снизу то

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

где Z^*_{LP} — оптимальное решение задачи (7). Наконец, докажем теорему об аппроксимационных свойствах Алгоритма 1.

Теорема 2. *Алгоритм 1 является α-аппроксимирующим для задачи (7) с $\alpha = 2 - \frac{2}{|A|}$ где $A = \{v \in V : f(\{v\}) = 1\}$.*

Доказательство. Для того чтобы это доказать рассмотрим следующее неравенство:

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

Первый переход в нем верен в силу определения Z^*_{LP} и того что F' по Теореме 1 является допустимым в (7). Третье неравенство обсуждалось выше. Осталось обосновать второе.

Мы знаем, что

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

(первое следует из того что в конце работы алгоритма все ограничения становятся активными, т.е. $c_e = \sum y_S$, а второе это просто перестановка порядка суммирования). Соответственно надо доказать, что

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

Покажем это по индукции. В начале работы алгоритма $y_S = 0$, так что это неравенство выполнено. Пусть оно выполнено первых k итераций, рассмотрим $(k+1)$-ю. Левая часть неравенства увеличится на $\varepsilon \sum\limits_{S \in \mathcal{C}_a} |F' \cap \delta(S)|$, где $\mathcal{C}_a$ — множество активных (с $f(C) = 1$) кластеров на данной итерации. Правая получит прибавку $\varepsilon \left(2 - \frac{2}{|A|}\right) \cdot |\mathcal{C}_a|$. Докажем, что

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

тогда шаг индукции будет обоснован. Обозначим $d(S) = |F' \cap \delta(S)|$ — число выходящих из S ребер. Тогда

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

здесь $\mathcal{C} = \mathcal{C}_i \cup \mathcal{C}_a$, и неравенство верно т.к. F' задает лес. Если мы покажем, что

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

то из предыдущего неравенства можно будет получить

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

где последний переход верен в силу того что $|A| \geqslant |\mathcal{C}_a|$ (число активных кластеров не может расти в процессе работы алгоритма, т.к. алгоритм не объединяет неактивные кластеры). Это и подведет доказательство к концу.

Осталось показать что верно (10). Для этого рассмотрим конденсированный граф H, вершины v которого соответствуют компонентам $C \in \mathcal{C}$ ($\mathcal{C}$ — множество связных компонент на данной итерации алгоритма), а ребра (v_1, v_2) проведены если между соответствующими компонентами есть ребро из F'. В силу определения F' граф H является лесом. Тогда надо доказать, что среди деревьев леса H нет листьев, соответствующих неактивным вершинам (при этом мы не считаем одиночные неактивные вершины, так как их степень равна 0 и это можно учесть в переходе $\sum\limits_{S \in \mathcal{C}} d(S) \leqslant 2(|\mathcal{C}_a| + |\mathcal{C}_i| - 1)$ отняв внутри все вершины с нулевой степенью).

Предположим, что есть некоторая вершина $v \in V(H)$ являющаяся листом (соединенным со своим родителем ребром e) и соответствующая неактивному множеству $C_v \in \mathcal{C}$.

Это множество C_v принадлежит некоторой компоненте связности $N \in F$, где F — множество векторов до прореживания. Но тем не менее в H оказалось, что v является листом, значит все кроме одного ребра e что выходили из C_v в другие вершины N были убраны при прореживании.

Теперь рассмотрим что получится, если мы удалим ребро e из компоненты N. Т.к. эта компонента является деревом, то она распадется на две компоненты: N_1 и N_2. Без ограничения общности можем считать, что $C_v \subseteq N_1$. При прореживании это ребро не убирали, значит $f(N_1) = 1$ или $f(N_2) = 1$. Но $f(N) = 0$, так что если верно лишь одна из этих двух вариантов то получаем противоречие по Лемме 1 (пусть $f(N_1) = 0$, тогда и $f(N_2) = 0$, и наоборот). Значит $f(N_1) = f(N_2) = 1$.

Теперь рассмотрим компоненты $(C_v, C_1, \ldots, C_m)$, на которые разбивается N_1 если мы используем ребра F'. Каждая из этих компонент кроме C_v такова, что $f(C_i) = 0$ (т.к. ребра были удалены). Но и $f(C_v) = 0$ по нашему предположению. Значит $f(N_1) = f\left(\bigcup\limits_{i=1}^{m} C_i \cup C_v\right) = 0$ по симметричности, что противоречит тому что $f(N_1) = 1$. Полученное противоречие показывает, что нет листовых вершин в H соответствующих неактивным вершинам, а это значит что степень любой неактивной вершины не меньше 2 (либо равна 0, но они не влияют на неравенство как показано выше). ■

Осталось сказать пару слов по поводу реализации. При использовании структур данных вроде системы непересекающихся множеств, а так же хранении очереди ребер с приоритетом (в качестве ключа используя оставшееся до ограничения время) можно получить асимптотическую сложность этого алгоритма $O(\min(n^2 \log n, mn\alpha(m, n)))$ где $\alpha(m, n)$ — обратная функция Аккермана (можно считать $\alpha(m, n) < 5$ ввиду очень быстрого роста прямой функции), m — число ребер, n — число вершин [11]. Также можно гарантировать $O(gm \log n)$ если использовать деление ребер, как показано в [16] где g — точность с которой хранятся веса ребер.

Проблема заключается в том что задачу $PCST$ нельзя поставить в форме (7), для нее нет подходящей f. Но можно модифицировать алгоритм 1 так чтобы он работал и с такой постановкой, что и будет показано в следующем пункте.

3.3 Реализация для решения задачи $PCST$

Выпишем релаксированную постановку задачи ЛП для $PCST$ (ранее мы выписывали ее для A-$PCST$)

$$\begin{aligned}
\underset{\substack{x_e, s_v \\ e\in E, v\in V\setminus\{r\}}}{\text{minimize}} \quad & \sum_{e\in E} c_e x_e + \sum_{v\in V\setminus\{r\}} (1 - s_v)\pi_v \\
\text{s.t.} \quad & \sum_{e\in\delta(S)} x_e \geqslant s_v, && \forall S \subseteq V\setminus\{r\},\ v \in S, \qquad (11)\\
& x_e \geqslant 0, && \forall e \in E, \\
& s_v \geqslant 0, && \forall v \in V\setminus\{r\}.
\end{aligned}$$

Такая постановка существенно отличается от той что была использована в (6), но там такая форма была нужна чтобы свести к ней задачу k-MST. Переменные s_v здесь имеют смысл индикаторов того что вершина v взята в дерево.

Теперь выпишем двойственную к ней

$$\begin{aligned}
\underset{y_S:\ S\subset V\setminus\{r\}}{\text{maximize}} \quad & \sum_{S\in V\setminus\{r\}} y_S \\
\text{s.t.} \quad & \sum_{S: e\in\delta(S)} y_S \leqslant c_e, && \forall e \in E, \qquad (12)\\
& \sum_{S\subseteq T} y_S \leqslant \sum_{v\in T} \pi_v, && \forall T \subset V\setminus\{r\}, \\
& y_S \geqslant 0, && \forall S \subset V\setminus\{r\}.
\end{aligned}$$

Идейно алгоритм решения схож с Алгоритмом 1. Мы опять же пытаемся равномерно увеличивать двойственные переменные y_S. Но теперь нам надо еще поддерживать второй набор ограничений, соответственно теперь ε будет определятся как минимум из двух (по каждому из типов ограничений).

Если происходит нарушение условия первого типа, то для некоторого ребра первое неравенство становится равенством, и мы добавляем это ребро в наше наращиваемое множество F.

Если происходит нарушение условия второго типа то алгоритм будет деактивировать компоненту (в некотором смысле это значит, что добавление этой компоненты уже не даст выигрыша). Тогда алгоритм помечает все вершины в этой компоненте номером этой компоненты.

Ниже приведен псевдокод данного алгоритма.

Функция λ здесь имеет смысл индикатора активности очередной компоненты (и в чем то она похожа на функцию f что была в предыдущем алгоритме). Так же как и для Алгоритма 1 для этого можно доказать (см. [11]) результат касающийся аппроксимационых свойств.

Теорема 3. *Алгоритм 2 является α-аппроксимирующим для задачи $PCST$ с $\alpha = 2 - \frac{2}{n-1}$ где n — число вершин в графе G.*

Algorithm 2: $(2-\varepsilon)$-аппроксимирующий алгоритм для задачи $PCST$

Data: Взвешенный неорентированный граф $G = (V, E)$ с неотрицательными весами ребер $c_i \geqslant 0$, призами $\pi_i \geqslant 0$ и корнем r

Result: Дерево F', включающее вершину r

Шаг 1, наращивание

begin

  $F \longleftarrow \emptyset$

  $Z_{DRLP} \longleftarrow 0$

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

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

    Обрать метку с v

    $d(v) \longleftarrow 0$

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

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

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

  end

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

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

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

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

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

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

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

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

      end

    end

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

      $\lambda(C^*) \longleftarrow 0$ Пометить все непомеченные в C^* вершины меткой 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^* соединяет компоненты C_q и 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 $\lambda(C_p \cup C_q) \longleftarrow 1$

  end

end

Шаг 2, прореживание

F' получается из F удалением как можно большего числа ребер так чтобы выполнялись два условия: во-первых каждая вершина без метки должна быть соединена с корнем r, а во-вторых если вершина с меткой C уже подсоединена к корню то так же должны быть подсоединены и все вершины имеющие в своем множестве меток метку C.

4 Численные эксперименты

4.1 Описание алгоритмов

В данной секции будут описаны все алгоритмы восстановления матрицы суперпозиции по зашумленному варианту, что сравнивались экспериментально.

4.1.1 DFS

Обход дерева жадно (по ребрам наибольшего веса, что соответствует взятию наибольших вероятностей) вглубь. При этом обход вершины прекращается если число выходящих ребер становится равным арности функции.

Передается список из $n-1$ арностей, т.к. для корневой функции $*$ арность известна и равна 1.

Algorithm 3: Восстановление дерева суперпозиции на основе DFS

Data: Зашумленная матрица суперпозиции $M \in \mathbb{R}_+^{n\times(n+1)}$, список l из $n-1$ арностей функций
Result: Корректная (с точки зрения арностей) матрица суперпозиции M_{res}

```
begin
    l ⟵ [1] + l      (добавляем 1 в начало списка)
    M' ⟵ матрица n × (n + 1) из нулей
    used ⟵ {0}
    Function DFS_rec(i):
        foreach j ∈ used do
            M[i][j] = 0
        end
        arity ⟵ l[i]
        Найти множество pos из положений arity наибольших значений в M[i]
         (в i-й строке)
        foreach j ∈ pos do
            M_res[i][j] = 1
            if j ≠ n then
                used ⟵ used ∪ pos
            end
        end
        foreach j ∈ pos do
            if j ≠ n then
                DFS_rec(j)
            end
        end
        return
    DFS_rec(0)
end
```

Множество *used* использованных вершин заполняется до перехода, чтобы следующие вершины не могли их использовать. Условие $pos \neq n$ нужно чтобы отбрасывать

столбец переменных. Строки $6-8$ в алгоритме нужны чтобы убрать их рассмотрения вершины что уже были использованы.

Еще есть вопрос с тем как поступать, когда осталось меньше вершин чем арность очередной функции. В формате описания суперпозиции обсуждалось, что если в строке меньше элементов, чем арность то это значит что столбец переменной задействуется несколько раз. Поэтому в случае вышеобозначенного события алгоритм просто не совершает новых переходов (считаем что остальные переходы по переменным). Так же эта проблема решается и в двух следующих алгоритмах.

4.1.2 BFS

В целом то же, но обход в ширину.

Algorithm 4: Восстановление дерева суперпозиции на основе BFS

Data: Зашумленная матрица суперпозиции $M \in \mathbb{R}_{+}^{n\times(n+1)}$, список l из $n-1$ арностей функций
Result: Корректная (с точки зрения арностей) матрица суперпозиции M_{res}

```
1  begin
2      l ⟵ [1] + l      (добавляем 1 в начало списка)
3      M' ⟵ матрица n × (n + 1) из нулей
4      used ⟵ {0}
5      queue ⟵ {0}
6      while queue ≠ ∅ do
7          foreach j ∈ used do
8              M[i][j] = 0
9          end
10         arity ⟵ l[i]
11         Найти множество pos из положений arity наибольших значений в M[i]
12         foreach j ∈ pos do
13             M_res[i][j] = 1
14             if j ≠ n then
15                 used ⟵ used ∪ pos
16             end
17             queue ⟵ queue + [j]      (добавляем j в конец очереди)
18         end
19     end
20 end
```

4.1.3 Алгоритм Прима

Вариант алгоритма Прима поиска минимального остовного дерева в графе с ограничениями арности, добавленными в виде дополнительных проверок в процессе поиска минимального ребра. После добавления очередной вершины из множества ребер кандидатов удаляются все ребра, что ведут в эту вершину (за счет этого дерево получается ориентированным). Так же, если из вершины уже выходит слишком много (арность вершины) ребер, то из множества кандидатов удаляются все ребра начинающиеся из нее.

Основное достоинство этого алгоритма заключается в том что он никак не зависит от обхода. Более того, при малых шумах (в смысле обсуждавшемся ранее) этот алгоритм восстанавливает дерево суперпозиции корректно (т.к. элементы дерева отделимы от шума, и значит жадное построение остановится как раз после постройки дерева). Единственный случай, когда этот алгоримт может неправильно работать при малых шумах это случай когда переменная задействуется несколько раз.

Algorithm 5: Восстановление дерева суперпозиции на основе алгоритма Прима

Data: Зашумленная матрица суперпозиции $M \in \mathbb{R}_+^{n\times(n+1)}$, список l из $n-1$ арностей функций
Result: Корректная (с точки зрения арностей) матрица суперпозиции M_{res}

```
1  begin
2      l ⟵ [1] + l      (добавляем 1 в начало списка)
3      M' ⟵ матрица n × (n + 1) из нулей
4      used ⟵ {0}
5      edges ⟵ ∅
6      foreach j ∈ range(0, n) do
7          if j ∉ used then
8              edges ⟵ edges ∪ (0, j, M[0][j])      (откуда, куда, вес)
9          end
10     end
11     while edges ≠ ∅ do
12         Найти тройку (from, to, w) с максимальным весом w из edges
13         foreach j ∈ used do
14             M[to][j] = 0
15         end
16         foreach j ∈ range(0, n) do
17             if j ∉ used then
18                 edges ⟵ edges ∪ (to, j, M[to][j])      (откуда, куда, вес)
19             end
20         end
21         if to ≠ n then
22             edges ⟵ edges\(from, to, w)
23             l[from] ⟵ l[from] − 1
24         end
25         Удалить из edges все тройки (i, j, w) где j = to
26         if l[to] = 0 then
27             Удалить из edges все тройки (i, j, w) где i = from
28         end
29     end
30 end
```

Важно, что удаление проводимого ребра в строке 18 не делается для ребер идущих в вершину —переменную. Тем не менее, отдельно надо проверять, что мы не возьмем его несколько раз (кроме случая разобранного ниже). В реализации это сделано следующим образом — при поиске ребра с максимальным весом максимум ищется

слева-направо, а т.к. столбец переменной в нашем описании стоит справа то другие претенденты тоже просматриваются. В результате ребро в переменную берется в случае, когда других вариантов нет).

Этот случай можно отдельно рассмотреть в коде (добавить проверку того что взятое ребро входит в переменную, и оставить его если это максимум на данный момент), и именно такой вариант и используется в экспериментах.

4.1.4 Алгоритмы на основе PCST

В начале мы должны свести матрицу к неориентированной форме. Для этого мы берем матрицу M, отбрасываем последний столбец (переходя к квадратной матрице M', столбец переменных можно восстановить отдельно, а в алгоритме $PCST$ арности мы все равно учесть не можем) после чего даем алгоритму решения задачи $PCST$ на вход граф с матрицей смежности $1-(M'+M'^T)/2$ и призами 0.5 за каждую вершину.

Здесь отнятие от 1 следует понимать как поэлементную операцию (таким образом штраф за ребро это то насколько оно не равно 1).

Призы следует выставлять в 0.5 т.к. при меньших значениях дерево будет усеченным (при шуме в 0.5 могут существовать отброшенные вершины, что должны были попасть в дерево), а если взять большее значение то наоборот, дерево $PCST$ будет разрастаться, захватывая шумовые выершины.

Algorithm 6: Восстановление дерева суперпозиции на основе алгоритма решения задачи $PCST$

Data: Зашумленная матрица суперпозиции $M \in \mathbb{R}_+^{n\times(n+1)}$, список l из $n-1$ арностей функций

Result: Корректная (с точки зрения арностей) матрица суперпозиции M_{res}

begin

- Отбрасываем последний столбец матрицы M, получаем матрицу M'
- $M'_{new} = 1 - \frac{M'+M'^T}{2}$
- $M'_{pcst} = PCST(M'_{new}, 0.5)$
- Добавить столбец из нулей справа к M'_{pcst}, получить M_{pcst}
- Восстановить дерево в M_{pcst} каким либо обходом из корневой вершины, получить M_{res}

end

Восстановление может производится одним из алгоритмов выше.

Так же можно использовать результаты $PCST$ подавая их на вход другим алгоритмам, к примеру

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

т.е. использовать алгоритм $PCST$ как рекомендательный для других, такой подход так же был исследован при экспериментах.

Помимо этого исследовались варианты алгоритмов, которые получались когда на вход $PCST$ подавалася матрица M' а не ее приведенный к симметричной форме вариант. В некотором смысле такой подход близок к нашему использованию алгоритма Прима (т.к. это алгоритм построения минимального остовного дерева для неориентированного графа, а мы используем его для ориентированных графов).

Ниже представлен список всех 11 сравниваемых алгоритмов

- DFS
- BFS
- Алг. Прима
- $k - MST$ через $PCST$
- $k - MST + DFS$
- $k - MST + BFS$
- $k - MST$ + алг. Прима
- $k - MST$ через $PCST$, ориентированный
- $k - MST + DFS$, ориентированный
- $k - MST + BFS$, ориентированный
- $k - MST$ + алг. Прима, ориентированный

Последние четыре — как раз случаи, когда мы подаем на вход алгоритму решения задачи $PCST$ ориентированный граф.

4.2 Описание данных, и процедуры их порождения

В процессе генерации данных были сделаны следующие предположения

- Наборы арностей функций имеют биномиальное распределение (много функций малой арности).
- Только функции одной переменной (случай нескольких переменных рассмотрен выше).

Необходимо восстановить корректное дерево суперпозиции, так что восстановлением считаем лишь полное совпадение с исходной матрицей.

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

где N — функция зашумления, а R — алгоритм восстановления.

- 50 наборов арностей (длиной 5 —20)
- 20 функций для каждого набора
- 5 зашумлений каждой функции
- Шум — равномерно распределенный
- Калибровка — линейная в отрезок $[0, 1]$

Замечание. Шум накладывается следующим образом. Пусть α — предельная сила шума. Тогда зашумленная матрица $N(M)$ получается по следующей формуле

$$N(M) = Cal\left(M + U(M.size, [-\alpha, \alpha])\right),$$

где $U(M.size, [-\alpha, \alpha])$ возвращает матрицу (того же размера что и матрица M) у которой каждый элемент есть реализация равномерно распределенной на $[-\alpha, \alpha]$ случайной величины, а $Cal(M)$ — это линейная $MinMax$ калибровка в отрезок $[0, 1]$, т.е.

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

где минимум и максимум ищутся по всей матрице, а вычитание и деление — поэлементное.

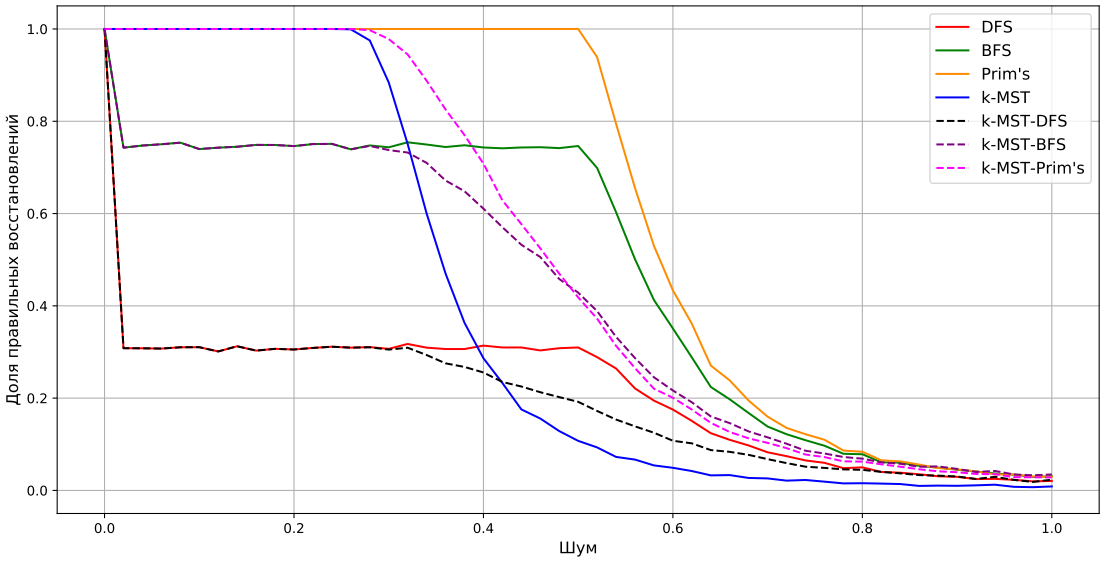


Рис. 1: Зависимость доли правильных восстановлений от силы шума, функции арности 1 —2. k-MST алгоритмы используют симметризованную матрицу смежности

4.3 Результаты экспериментов

Код всех экспериментов представлен в [17]. В начале представим результаты работы алгоритмов (неориентированные варианты) для функций арностей 1 и 2 (Рис. 1).

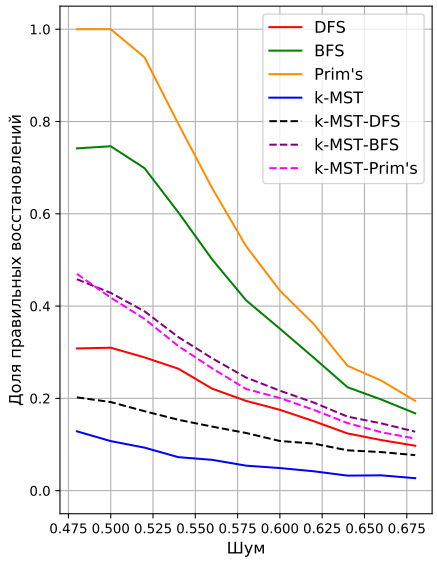


Рис. 2: Качество работы алгоритмов на границе, малые арности, равномерный шум, неориентированный вариант

Шум	.50	.52	.54	.56	.58
DFS	.31	.29	.26	.22	.19
BFS	.75	.70	.60	.50	.41
Прим	**1.0**	**.94**	**.79**	**.66**	**.53**
k-MST	.11	.09	.07	.07	.05
k-MST-DFS	.19	.17	.15	.14	.12
k-MST-BFS	.43	.39	.33	.29	.25
k-MST-$Prim's$	.42	.37	.31	.26	.22

Таблица 1: Качество при шуме ~ 0.5

На Рис. 2 представлены результаты работы алгоритмов при шуме ~ 0.5 (см. Рис. 1). Наибольшее качество показывает алгоритм основанный на алгоритме Прима, следом за ним — алгоритм на основе BFS. В Таблице 1 показаны значения качества работы алгоритмов при данных шумах.

Падение качества алгоритмов основанных на $PCST$ при шуме ~ 0.25 можно связать с тем, что при таком шуме в усредненной матрице $(M' + M'^T)/2$ шумовые ребра могут стать неотличимы от настоящих. Поясним это.

Пусть шум при котором в усредненной матрице возникают шумовые ребра того же веса что и реальные равен $0.25 + \varepsilon = \alpha$. Ребро веса 1 после зашумления можнт перейти в отрезок $[1 - \alpha, 1 + \alpha]$, а после калибровки оно перейдет в $\left[\frac{1}{1+2\alpha}, 1\right]$ (минимальное значение если есть ребро веса 0 что перешло в $-\alpha$, и 2 ребра 1 что перешли

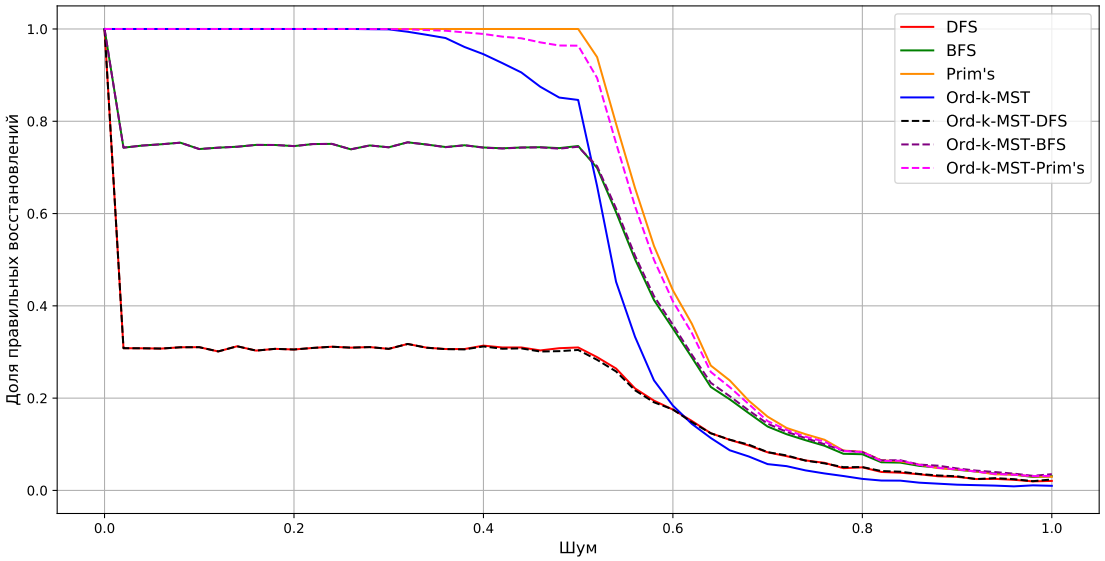


Рис. 3: Зависимость доли правильных восстановлений от силы шума, функции арности 1 —2. k-MST алгоритмы используют исходную матрицу смежности. Для DFS и BFS видим почти полное совпадение качества работы обычного и k-MST варианта

в $1+\alpha$ и $1-\alpha$).

Аналогично, ребро веса 0 может перейти в отрезок $\left[0, \frac{2\alpha}{1+2\alpha}\right]$. Теперь, когда мы усредняем матрицу у нас может произойти одно из двух: либо мы усреднили ребро 0 и ребро 1, либо просто два нулевых. Соответственно получаем, что после усреднения отрезки имеют вид

$$(0,0) \to \left[0, \frac{2\alpha}{1+2\alpha}\right], \ (0,1) \to \left[\frac{1}{2(1+2\alpha)}, \frac{1+2\alpha+2\alpha}{2(1+2\alpha)}\right],$$

нам нужно чтобы эти отрезки пересеклись, из чего

$$\frac{2\alpha}{1+2\alpha} = \frac{1}{2(1+2\alpha)}$$

из чего $\alpha = 0.25$, т.е. $\varepsilon = 0$, т.е. пересечения начинаются при шуме в 0.25.

Перейдем к ориентированным вариантам алгоритмов на основе k-MST. Результаты эксперимента для этого случая представлены на Рис. 3. Качество основанных на k-MST алгоритмов существенно выше чем в неориентированном случае.

Алгоритм, показанный синей линией — это $PCST$ с ориентированным графом на входе и восстановление полученной матрицы алгоритмом BFS. Смешанные алгоритмы мало отличаются по качеству от качества алгоритма которым производится восстановление (рунктирные линии идут рядом со сплошными).

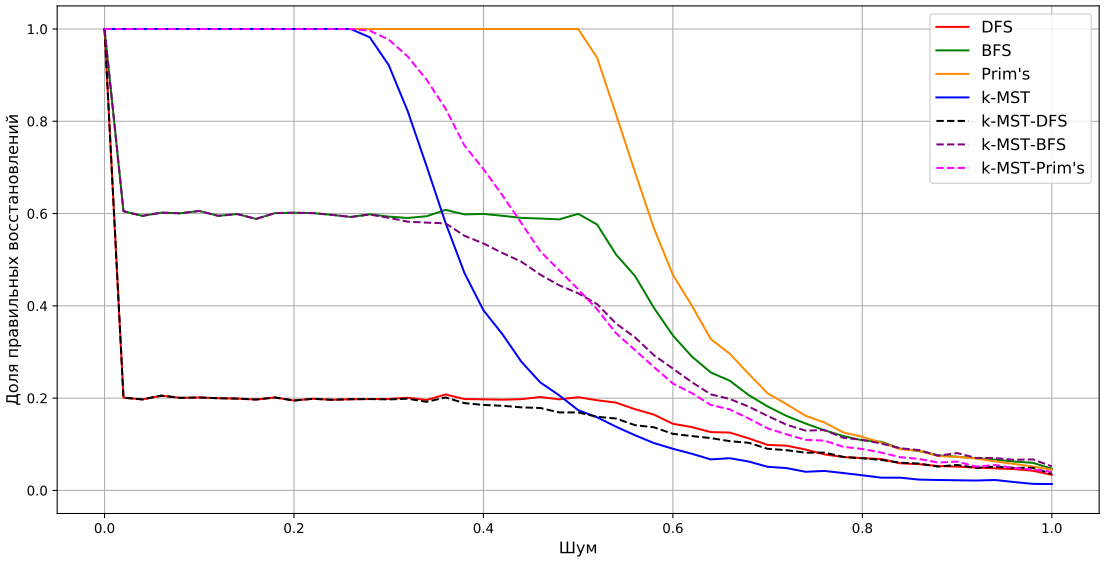


Рис. 5: Зависимость доли правильных восстановлений от силы шума, функции арности 1 —5. k-MST алгоритмы используют симметризованную матрицу смежности

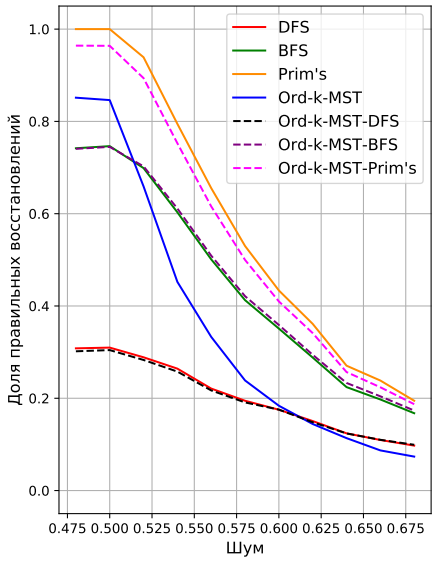


Рис. 4: Качество работы алгоритмов на границе, малые арности, равномерный шум, ориентированный вариант

Шум	.50	.52	.54	.56	.58
DFS	.31	.29	.26	.22	.19
BFS	.75	.70	.60	.50	.41
Прим	**1.0**	**.94**	**.79**	**.66**	**.53**
$Ord\text{-}k\text{-}MST$	.85	.66	.45	.33	.24
$Ord\text{-}k\text{-}MST\text{-}DFS$	.30	.28	.26	.22	.19
$Ord\text{-}k\text{-}MST\text{-}BFS$	.74	.70	.61	.51	.42
$Ord\text{-}k\text{-}MST\text{-}Prim's$	.96	.89	.75	.62	.50

Таблица 2: Качество при шуме ~ 0.5

На Рис. 4 представлены результаты работы алгоритмов при шуме ~ 0.5 (см. Рис. 3). Лучше всего работает алгоритм основанный на алгоритме Прима и алгоритмы на основе k-MST (обычный и смешанный с алгоритмом Прима). В Таблице 2 показаны значения качества работы алгоритмов при данных шумах.

Теперь предьявим результаты работы алгоритмов при наличии в наборе функций арностей 1 —5. Как можно видеть ((Рис. 5), особенно сильно падает качество алгоритмов с фиксированным порядком обхода (DFS, BFS). При этом k-MST с восстановлением с помощью алгоритма BFS работает почти так же хорошо как и в случае функций малой арности.

В ориентированных вариантах (Рис. 6) алгоритмов видим ту же ситуацию.

Наконец в Таблице 3 представлено полное время работы каждого алгоритма. Всего в датасете было $51 \cdot 50 \cdot 20 \cdot 5 \approx 250000$ матриц для восстановления.

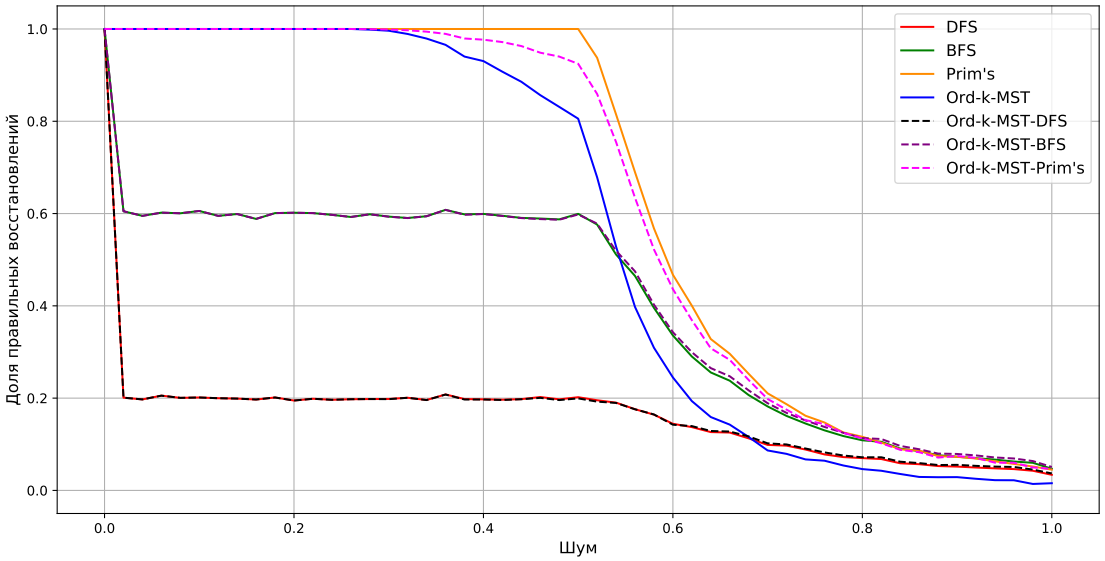


Рис. 6: Зависимость доли правильных восстановлений от силы шума, функции арности 1 —5. k-MST алгоритмы используют исходную матрицу смежности

Алгоритм	Малые арности время, с.	Большие арности время, с.
DFS	51	48
BFS	53	51
$Prim's$	85	67
k-MST	182	151
k-MST-DFS	196	162
k-MST-BFS	204	166
k-MST-$Prim's$	241	187
Ord-k-MST	168	127
Ord-k-MST-DFS	175	131
Ord-k-MST-BFS	171	133
Ord-k-MST-$Prim's$	195	149

Таблица 3: Полное время работы на датасете (250000 запусков)

5 Заключение

В работе были исследованы алгоритмы восстановления матриц суперпозиций по их зашумленным образцам. Наилучшим образом себя показал алгоритм на основе алгоритма Прима, оказавшись единственным алгоритмом что был способен корректно восстанавливать матрицы суперпозиции при малых шумах.

Алгоритм на основе эвристического алгоритма решения задачи $PCST$ также демонстрирует точное восстановление, но при меньших шумах. Хуже всего себя проявляют алгоритмы с фиксированным обходом — BFS и DFS, хотя $PCST$ алгоритм с восстановлением с помощью BFS работает намного лучше и является промежуточным по качеству между неориентированным $PCST$ м алгоритмом Прима.

Список литературы

[1] Davis L. Handbook of genetic algorithms. CUMINCAD, 1991.

[2] Koza J. R. Genetic programming as a means for programming computers by natural selection // Statistics and computing. 1994. Vol. 4, no. 2. P. 87–112.

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

[4] Searson D. P. GPTIPS 2: an open-source software platform for symbolic data mining // Handbook of genetic programming applications. Springer, 2015. P. 551–573.

[5] Бочкарев А. М., Софронов И. Л., Стрижов В. В. Порождение экспертно-интерпретируемых моделей для прогноза проницаемости горной породы // Системы и средства информатики. 2017. Vol. 27, no. 3. P. 74–87.

[6] Lozovanu D., Zelikovsky A. Minimal and bounded tree problems // Tezele Congresului XVIII al Academiei Romano-Americane. 1993. P. 25–26.

[7] Ravi R., Sundaram R., Marathe M. V. et al. Spanning trees—short or small // SIAM Journal on Discrete Mathematics. 1996. Vol. 9, no. 2. P. 178–200.

[8] Chudak F. A., Roughgarden T., Williamson D. P. Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation // Mathematical Programming. 2004. — Jun. Vol. 100, no. 2. P. 411–421. URL: https://doi.org/10.1007/s10107-003-0479-2.

[9] Awerbuch B., Azar Y., Blum A., Vempala S. New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen // SIAM Journal on computing. 1998. Vol. 28, no. 1. P. 254–262.

[10] Arora S., Karakostas G. A $2 + \varepsilon$ approximation algorithm for the k-MST problem // Mathematical Programming. 2006. — Jul. Vol. 107, no. 3. P. 491–504. URL: https://doi.org/10.1007/s10107-005-0693-1.

[11] Goemans M. X., Williamson D. P. A general approximation technique for constrained forest problems // SIAM Journal on Computing. 1995. Vol. 24, no. 2. P. 296–317.

[12] Hegde C., Indyk P., Schmidt L. A fast, adaptive variant of the Goemans-Williamson scheme for the prize-collecting Steiner tree problem // Workshop of the 11th DIMACS Implementation Challenge. Providence, Rhode Island: Workshop of the 11th DIMACS Implementation Challenge. 2014.

[13] Hegde C., Indyk P., Schmidt L. A nearly-linear time framework for graph-structured sparsity // International Conference on Machine Learning. 2015. P. 928–937.

[14] URL: https://github.com/fraenkel-lab/pcst_fast.

[15] Ras C., Swanepoel K., Thomas D. A. Approximate Euclidean Steiner Trees // Journal of Optimization Theory and Applications. 2017. — Mar. Vol. 172, no. 3. P. 845–873. URL: https://doi.org/10.1007/s10957-016-1036-5.

[16] Cole R. B., Hariharan R., Lewenstein M., Porat E. A faster implementation of the Goemans-Williamson clustering algorithm // SODA. 2001.

[17] URL: https://gitlab.com/Polkut/PCST_superposition_tree_reconstruction.