AlEM: новый параллельный алгоритм линейного программирования для кластерных вычислительных систем

А.Э. Жулев, Л.Б. Соколинский

Южно-Уральский государственный университет
(национальный исследовательский университет)

Работа посвящена новому масштабируемому проекционному алгоритму AlEM (Along Edges Movement) для линейного программирования на кластерных вычислительных системах. Алгоритм начинает свою работу в произвольной вершине многогранника допустимых решений и строит оптимальный путь по ребрам до точки оптимума. Представлено формализованное описание алгоритма. Описана его параллельная реализация. Исследована масштабируемость параллельной реализации на кластерной вычислительной системе. Приведены результаты вычислительных экспериментов, подтверждающие высокую параллельную эффективность алгоритма AlEM.

Ключевые слова: линейное программирование, алгоритм AlEM, проекционный алгоритм, параллельная реализация, MPI, кластерная вычислительная система, исследование масштабируемости.

1. Введение

Линейное программирование (ЛП) является одной из наиболее востребованных оптимизационных математических моделей, используемых для решения практических масштабных задач в индустрии, экономике, науке и других областях [1–3]. Наиболее популярными подходами к решению задач ЛП являются симплекс-метод [4] и метод внутренних точек [5]. Однако параллельные алгоритмы, основанные на этих методах, в общем случае не поддаются эффективному распараллеливанию на больших вычислительных системах с распределенной памятью. В соответствии с этим остается актуальной задача разработки новых высоко-масштабируемых методов и алгоритмов для решения задач ЛП.

Одним из перспективных подходов является использование проекционных методов для решения задач ЛП. Впервые проекционные методы были применены для решения систем линейных алгебраических уравнений Стефаном Качмажом [6] и Джанфранко Чиммино [7] в 1930-х годах. Идея проекционного метода состоит в последовательном или совместном ортогональном проектировании текущего приближения на гиперплоскости, соответствующие линейным алгебраическим уравнениям, образующим систему. Результатом будет следующее приближение, находящееся ближе к точному решению, чем предыдущее. Доказано, что такой итерационный процесс сходится к точному решению, если система совместна. Шмуэль Агмон [8] обобщил проекционный метод для систем линейных неравенств (задача линейной совместности). В последние десятилетия класс проекционных методов активно развивался [9].

И.И. Ерёмин одним из первых предложил использовать проекционные методы для решения задач ЛП [10]. Затем этот подход был развит в целом ряде работ (см., например, [11–15]). Главным недостатком проекционных методов является их низкая линейная скорость сходимости, зависящая от угла между гиперплоскостями [16]. С другой стороны, проекционные методы допускают эффективное распараллеливание на многопроцессорных системах с распределенной памятью, так как ортогональные проекции могут вычисляться независимо [17]. Однако, как показано в [15], граница эффективной масштабируемости параллельного проекционного метода не превышает $O(\sqrt{m})$ процессорных узлов с распре-

деленной памятью, где m — число ограничений задачи ЛП.

В работе [15] был представлен апекс-метод — принципиально новый проекционный метод решения задач ЛП, строящий на поверхности допустимого многогранника (многогранника, ограничивающего область допустимых решений) оптимальный путь к точке максимума целевой функции. Для построения оптимального пути апекс-метод использует операцию псевдопроекции, $\boldsymbol{\rho}$, являющуюся обобщением понятия метрической проекции на выпуклое множество. Псевдопроекция является предельной точкой последовательности приближений, получаемых путем ортогонального проектирования на гиперплоскости, ограничивающие область допустимых решений. Основная идея апекс-метода представлена на рис. 1а. К

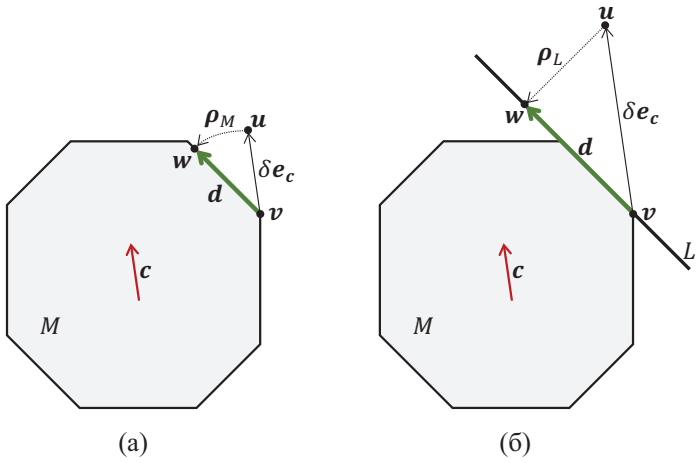

Рис. 1: Построение вектора движения $\boldsymbol{d}$:
а) в апекс-методе; б) в алгоритма AlFaMove.
M — многогранник, ограничивающий область допустимых решений;
$\boldsymbol{c}$ — градиент линейной целевой функции; $\boldsymbol{e_c}$ — единичный вектор,
сонаправленный с вектором $\boldsymbol{c}$; δ — положительная вещественная
константа; $\boldsymbol{\rho}$ — операция построения псевдопроекции на выпуклое
множество.

точке текущего приближения $\boldsymbol{v}$ прибавляется вектор $\delta \boldsymbol{e_c}$, сонаправленный с градиентом $\boldsymbol{c}$ линейной целевой функции. Здесь $\boldsymbol{e_c}$ — единичный вектор, δ — положительная вещественная константа, являющаяся параметром алгоритма. Из получившейся точки $\boldsymbol{u}$ строится псевдопроекция $\boldsymbol{w}$ на допустимый многогранник M:

$$\boldsymbol{w} = \boldsymbol{\rho}_M(\boldsymbol{v}).$$

Вектор $\boldsymbol{d} = \boldsymbol{w} - \boldsymbol{v}$ указывает направление движения по поверхности многогранника M к следующему приближению, в качестве которого берется граничная точка, максимально удаленная от $\boldsymbol{v}$ в направлении вектора $\boldsymbol{d}$. За конечное число шагов этот алгоритм приходит к решению задачи ЛП при условии, что $M \neq \emptyset$. Указанный метод имеет два недостатка. Во-первых, доказано, что метод сходится, если псевдопроекцию заменить на метрическую проекцию [15]. При использовании псевдопроекции апекс-метод может в отдельных случаях зацикливаться. Во-вторых, апекс-метод требует, чтобы точка $\boldsymbol{w}$ находилась на той же грани, что и точка $\boldsymbol{v}$. Следовательно, малый размер грани потребует малого значения вещественного параметра δ, что приводит к значимой потере точности при вычислении вектора движения $\boldsymbol{d}$.

В недавней статье [18] был представлен новый проекционный алгоритм AlFaMove (Along Faces Movement), устраняющий описанные недостатки. Принципиальное отличие алгоритма AlFaMove от апекс-метода заключается в том, что псевдопроекция из точки $\boldsymbol{u}$ строится не

на допустимый многогранник M, а на линейное многообразие L, порождаемое пересечением ограничивающих гиперплоскостей, проходящих через точку $\boldsymbol{v}$ (см. рис. 1б). В этом случае величина положительной константы δ может быть сколь угодно большой, что обеспечивает достаточную точность вычисления вектора $\boldsymbol{d}$ даже для очень маленьких граней. Более того, в работе [19] было доказано, что в случае линейного многообразия псевдопроекция совпадает с ортогональной проекцией, что обеспечивает сходимость алгоритма. Основной недостаток алгоритма AlFaMove заключается в том, что он имеет экспоненциальную вычислительную сложность. Действительно, если через точку $\boldsymbol{v}$ проходит k ограничивающих гиперплоскостей, то при выборе оптимального направления движения нам необходимо рассмотреть $(2^k - 1)$ линейных многообразий.

В этой статье предлагается и исследуется новый проекционный алгоритм линейного программирования AlEM (Along Edges Movement), ориентированный на кластерные вычислительные системы. В отличие от алгоритма AlFaMove алгоритм AlEM использует для построения оптимального пути только ребра допустимого многогранника, соответствующие одномерным линейным многообразиям. Алгоритм AlEM позволяет существенно уменьшить вычислительную сложность по сравнению с алгоритмом AlFaMove. Ели в n-мерном евклидовом пространстве через вершину допустимого многогранника проходят q ограничивающих гиперплоскостей, то для выбора оптимального пути необходимо рассмотреть C_q^{n-1} сочетаний, где

$$C_q^{n-1} = \frac{q!}{(q-n+1)!(n+1)!}.$$

Так, например, в случае гиперкуба AlFaMove в каждой вершине должен обработать $2^{(n-1)}$ комбинаций в то время, как AlEM должен будет обработать только n комбинаций.

Статья организована следующим образом. В разделе 2 представлен необходимый теоретический базис. Раздел 3 содержит формализованное описание алгоритма AlEM. Раздел 4 посвящен описанию параллельной версии алгоритма AlEM. В разделе 5 представлены информация о программной реализации параллельной версии алгоритма AlEM и результаты экспериментов на кластерной вычислительной системе по исследованию ее масштабируемости. В заключении суммируются полученные результаты и намечаются направления дальнейших исследований.

2. Теоретический базис

Данный раздел содержит необходимый теоретический базис, используемый для формализованного описания алгоритма AlEM.

В евклидовом пространстве $\mathbb{R}^n$ рассматривается задача ЛП в общем виде с системой ограничений

$$\begin{cases} \langle \boldsymbol{a}_1, \boldsymbol{x} \rangle \leqslant b_1 \\ \ldots\ldots \\ \langle \boldsymbol{a}_k, \boldsymbol{x} \rangle \leqslant b_k \\ \langle \boldsymbol{a}_{k+1}, \boldsymbol{x} \rangle = b_{k+1} \\ \ldots\ldots \\ \langle \boldsymbol{a}_{k+l}, \boldsymbol{x} \rangle = b_{k+l}, \end{cases} \tag{1}$$

включающей в себя k линейных неравенств и l линейных уравнений. Здесь и далее $\langle \cdot, \cdot \rangle$ обозначает скалярное произведение двух векторов. Предполагается, что система (1) включает в себя также неравенства вида

$$\begin{gathered} -x_1 \leqslant 0; \\ \ldots\ldots\ldots \\ -x_n \leqslant 0. \end{gathered} \tag{2}$$

Общее число ограничений m определяется по формуле

$$m = k + l, \tag{3}$$

где $k > 0$ и $l \geqslant 0$. Решением задачи ЛП считается точка, удовлетворяющая системе ограничений (1), в которой достигается максимум линейной целевой функции

$$F(\boldsymbol{x}) = \langle \boldsymbol{c}, \boldsymbol{x} \rangle . \tag{4}$$

Неравенствам из системы ограничений (1) соответствует k гиперплоскостей, называемых граничными:

$$\begin{gathered} H_1 = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_1, \boldsymbol{x} \rangle = b_1\}; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ H_k = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_k, \boldsymbol{x} \rangle = b_k\}. \end{gathered} \tag{5}$$

Без ограничения общности мы можем предполагать, что среди граничных гиперплоскостей нет совпадающих. Граничные гиперплоскости отделяют k закрытых полупространств:

$$\begin{gathered} \hat{H}_1 = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_1, \boldsymbol{x} \rangle \leqslant b_1\}; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \hat{H}_k = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_k, \boldsymbol{x} \rangle \leqslant b_k\}. \end{gathered}$$

Пересечение этих полупространств образует замкнутый выпуклый многогранник $\hat{M}$:

$$\hat{M} = \bigcap_{i=1}^{k} \hat{H}_i.$$

Уравнениям из системы ограничений (1) соответствует l гиперплоскостей, называемых базисными:

$$\begin{gathered} H_{k+1} = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_{k+1}, \boldsymbol{x} \rangle = b_{k+1}\}; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ H_{k+l} = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_{k+l}, \boldsymbol{x} \rangle = b_{k+l}\}. \end{gathered} \tag{6}$$

Если $l > 0$, то пересечение базисных гиперплоскостей образует базисное линейное многообразие $\bar{\bar{L}}$:

$$\bar{\bar{L}} = \bigcap_{i=1}^{l} H_{k+i}. \tag{7}$$

Если уравнения в системе ограничений отсутствуют ($l = 0$), полагаем $\bar{\bar{L}} = \mathbb{R}^n$. Область допустимых решений задачи ЛП представляет собой замкнутый выпуклый многогранник M, называемый допустимым:

$$M = \bar{\bar{L}} \cap \hat{M}.$$

Мы будем предполагать, что допустимый многогранник M является непустым ограниченным множеством. В этом случае задача ЛП имеет решение.

Следующие уравнения, соответствующие граничным гиперплоскостям (5), будем называть граничными:

$$\begin{gathered} \langle \boldsymbol{a}_1, \boldsymbol{x} \rangle = b_1; \\ \ldots\ldots\ldots\ldots \\ \langle \boldsymbol{a}_k, \boldsymbol{x} \rangle = b_k. \end{gathered} \tag{8}$$

Следующие уравнения, соответствующие базисным гиперплоскостям (6), будем называть базисными:

$$\begin{gathered} \langle \boldsymbol{a}_{k+1}, \boldsymbol{x} \rangle = b_{k+1}; \\ \ldots\ldots\ldots\ldots\ldots \\ \langle \boldsymbol{a}_{k+l}, \boldsymbol{x} \rangle = b_{k+l}. \end{gathered} \tag{9}$$

Обозначим через $\bar{\bar{A}}$ основную матрицу (далее — просто матрицу) базисных уравнений (9). Везде далее мы будем предполагать, что

$$\operatorname{rank}\left(\bar{\bar{A}}\right) = l. \tag{10}$$

В этом случае базисное линейное многообразие $\bar{\bar{L}}$, определяемое формулой (7), имеет размерность

$$\dim\left(\bar{\bar{A}}\right) = n - l.$$

При этом мы также можем считать, что

$$l < n, \tag{11}$$

так как в противном случае допустимый многогранник M вырождается в точку. Поскольку система ограничений (1) включает в себя неравенства вида (2), а допустимый многогранник M по предположению является ограниченным, имеем

$$k > n. \tag{12}$$

Введем следующие обозначения:

$$\mathcal{I} = \{1, \ldots, m\}; \tag{13}$$
$$\hat{\mathcal{I}} = \{1, \ldots, k\}; \tag{14}$$
$$\bar{\bar{\mathcal{I}}} = \{k+1, \ldots, k+l\}. \tag{15}$$

Для произвольного $\mathcal{J} \subseteq \hat{\mathcal{I}}$ обозначим через $\hat{\bar{A}}_{\mathcal{J}}$ матрицу системы, включающей в себя граничные уравнения с индексами из множества $\mathcal{J}$ и все базисные уравнения. Везде далее мы будем предполагать, что система ограничений (1) удовлетворяет условию

$$\forall \mathcal{J} \subseteq \hat{\mathcal{I}} \left(|\mathcal{J}| = n - l \Rightarrow \operatorname{rank}\left(\hat{\bar{A}}_{\mathcal{J}}\right) = n \right). \tag{16}$$

Другими словами, если к базисным уравнениям (9) добавить $n - l$ произвольных граничных уравнений (8), то матрица получившейся системы будет иметь ранг n. В этих предположениях справедливы следующее три утверждения, являющиеся теоретической основой алгоритма AlEM.

Утверждение 1. Пусть выполняется условие (16), $\mathcal{J} \subset \hat{\mathcal{I}}$ и $|\mathcal{J}| = n - l - 1$. Положим

$$\hat{\bar{L}}_{\mathcal{J}} = \bigcap_{i \in \mathcal{J} \cup \bar{\bar{\mathcal{I}}}} H_i \tag{17}$$

Тогда $\hat{\bar{L}}_{\mathcal{J}}$ является линейным многообразием размерности 1:

$$\dim\left(\hat{\bar{L}}_{\mathcal{J}}\right) = 1,$$

то есть $\hat{\bar{L}}_{\mathcal{J}}$ — прямая в пространстве $\mathbb{R}^n$.

Доказательство. Выберем любой $i' \in \hat{\mathcal{I}}$ такой, что $i' \notin \mathcal{J}$. Он существует в силу (12). Положим

$$\mathcal{J}' = \mathcal{J} \cup \{i'\}.$$

Обозначим через $\hat{\bar{A}}_{\mathcal{J}'}$ матрицу системы, включающей в себя граничные уравнения с индексами из множества $\mathcal{J}'$ и все базисные уравнения. По предположению (16) имеем

$$\operatorname{rank}\left(\hat{\bar{A}}_{\mathcal{J}'}\right) = n.$$

Матрица $\hat{\tilde{A}}_{\mathcal{J}'}$ имеет n строк и n столбцов. Удалив любую строку из этой матрицы, мы получим матрицу ранга $n-1$. Удалив строку, соответствующую добавленному неравенству с индексом i', получим матрицу $\hat{\tilde{A}}_{\mathcal{J}}$ ранга $n-1$:

$$\operatorname{rank}\left(\hat{\tilde{A}}_{\mathcal{J}}\right) = n-1.$$

Отсюда следует, что $\dim\left(\hat{\tilde{L}}_{\mathcal{J}}\right) = 1$. □

Обозначим через $\Gamma(M)$ множество граничных точек допустимого многогранника M. Следующее условие является необходимым и достаточным для того, чтобы граничная точка $\boldsymbol{v} \in \Gamma(M)$ была вершиной допустимого многогранника M.

Утверждение 2. Пусть выполняется условие (16) и $\boldsymbol{v} \in \Gamma(M)$. Тогда следующее условие является необходимым и достаточным для того, чтобы точка $\boldsymbol{v}$ являлась вершиной допустимого многогранника M:

$$\exists\, \mathcal{J} \subseteq \hat{\mathcal{I}} \left(|\mathcal{J}| = n-l \ \wedge\ \boldsymbol{v} \in \bigcap_{i \in \mathcal{J}} H_i \right). \tag{18}$$

Другими словами, граничная точка $\boldsymbol{v} \in \Gamma(M)$ тогда и только тогда является вершиной допустимого многогранника M, когда через нее проходят $n-l$ ограничивающих гиперплоскостей.

Доказательство. Сначала докажем «необходимость». Пусть граничная точка $\boldsymbol{v} \in \Gamma(M)$ является вершиной допустимого многогранника M. Это означает, что выполняется следующее условие:

$$\exists\, \mathcal{J}' \subseteq \hat{\mathcal{I}} \left(|\mathcal{J}'| \geqslant n-l \ \wedge\ \boldsymbol{v} = \bigcap_{i \in \mathcal{J}' \cup \bar{\bar{\mathcal{I}}}} H_i \right). \tag{19}$$

Выберем произвольное $\mathcal{J} \subseteq \mathcal{J}'$ такое, что $|\mathcal{J}| = n-l$. Из (19) следует, что

$$\boldsymbol{v} \in \bigcap_{i \in \mathcal{J}} H_i,$$

то есть условие (18) выполняется.

Теперь докажем «достаточность». Пусть для граничной точки $\boldsymbol{v} \in \Gamma(M)$ выполняется условие (18). Поскольку $\boldsymbol{v} \in M$, имеем

$$\boldsymbol{v} \in \bigcap_{i \in \bar{\bar{\mathcal{I}}}} H_i.$$

Сопоставляя это с (18), получаем

$$\boldsymbol{v} \in \bigcap_{i \in \mathcal{J} \cup \bar{\bar{\mathcal{I}}}} H_i. \tag{20}$$

Рассмотрим матрицу $\hat{\tilde{A}}_{\mathcal{J}}$ из коэффициентов уравнений, соответствующих гиперплоскостям в (20). Эта матрица имеет n строк и n столбцов. В соответствии с (16) имеем

$$\operatorname{rank}\left(\hat{\tilde{A}}_{\mathcal{J}}\right) = n.$$

Это означает, что

$$\dim\left(\bigcap_{i \in \mathcal{J} \cup \bar{\bar{\mathcal{I}}}} H_i \right) = 0.$$

Отсюда и из (20) следует

$$\boldsymbol{v} = \bigcap_{i \in \mathcal{J} \cup \bar{\bar{\mathcal{I}}}} H_i,$$

то есть выполняется условие (19), означающее, что $\boldsymbol{v}$ является вершиной допустимого многогранника M. □

Следующее утверждение будет использоваться для идентификации ребер допустимого многогранника M.

Утверждение 3. Пусть выполняется условие (16). Пусть D — ребро допустимого многогранника M, $\boldsymbol{v'}$ — его начальная вершина и $\boldsymbol{v''}$ — его конечная вершина:

$$D = \left\{ \boldsymbol{x} \in \mathbb{R}^n \,\middle|\, \boldsymbol{x} = (1-\lambda)\boldsymbol{v'} + \lambda \boldsymbol{v''},\ 0 \leqslant \lambda \leqslant 1,\ \boldsymbol{v'} \neq \boldsymbol{v''} \right\}.$$

Определим прямую L, включающую в себя ребро D:

$$L = \left\{ \boldsymbol{x} \in \mathbb{R}^n \,\middle|\, \boldsymbol{x} = \boldsymbol{v'} + \delta(\boldsymbol{v''} - \boldsymbol{v'}),\ \delta \in \mathbb{R},\ \boldsymbol{v'} \neq \boldsymbol{v''} \right\}.$$

Тогда

$$\exists\, \mathcal{J} \subseteq \hat{\mathcal{I}} \left(|\mathcal{J}| = n - l - 1 \ \wedge\ \forall i \in \mathcal{J} \left(\boldsymbol{v'} \in H_i \right) \ \wedge\ L = \bigcap_{i \in \mathcal{J} \cup \bar{\bar{\mathcal{I}}}} H_i \right). \tag{21}$$

Другими словами, если из всего множества граничных гиперплоскостей выбирать $n-l-1$ гиперплоскостей, проходящих через точку $\boldsymbol{v'}$, добавлять к ним все базисные гиперплоскости, и путем их пересечения получать прямые, то среди этих прямых обязательно найдется прямая L, содержащая ребро D.

Доказательство. Так как прямая L включает в себя ребро D допустимого многогранника M, то выполняется следующее условие:

$$\exists\, \mathcal{J}' \subseteq \hat{\mathcal{I}} \left(|\mathcal{J}'| \geqslant n - l - 1 \ \wedge\ \forall i \in \mathcal{J}' \left(\boldsymbol{v'} \in H_i \right) \ \wedge\ L = \bigcap_{i \in \mathcal{J}' \cup \bar{\bar{\mathcal{I}}}} H_i \right). \tag{22}$$

Выберем произвольное $\mathcal{J} \subseteq \mathcal{J}'$ такое, что $|\mathcal{J}| = n - l - 1$. Тогда из (22) следует

$$L \subseteq \bigcap_{i \in \mathcal{J} \cup \bar{\bar{\mathcal{I}}}} H_i. \tag{23}$$

В соответствии с утверждением 1 имеем

$$\dim \left(\bigcap_{i \in \mathcal{J} \cup \bar{\bar{\mathcal{I}}}} H_i \right) = 1. \tag{24}$$

Из (23) и (24) следует

$$L = \bigcap_{i \in \mathcal{J} \cup \bar{\bar{\mathcal{I}}}} H_i, \tag{25}$$

то есть условие (21) выполняется. □

3. Формализованное описание алгоритма AlEM

Данный раздел содержит формализованное описание нового алгоритма AlEM, строящего из произвольной вершины допустимого многогранника оптимальный путь по его ребрам к вершине, являющейся решением задачи ЛП. Оптимальность пути заключается в том, что всегда выбирается ребро, имеющее максимальное значение целевой функции в своей конечной точке.

Основной функцией алгоритма AlEM является функция Move$(\cdot)$, осуществляющая переход к вершине с максимальным значением целевой функции. Функция Move$(\cdot)$, в свою очередь, использует следующие две функции: $\boldsymbol{\rho}_{\mathcal{J}}(\cdot)$ — вычисление псевдопроекции на линейное многообразие и $Jump(\cdot)$ — перемещение по вектору.

Рассмотрим сначала функцию вычисления псевдопроекции $\boldsymbol{\rho}_{\mathcal{J}}(\boldsymbol{x})$ точки $\boldsymbol{x}$ на линейное многообразие

$$L_{\mathcal{J}} = \bigcap_{j \in \mathcal{J}} H_j,$$

где $\mathcal{J} \subseteq \{1, \ldots, m\}$. Основной элементарной операцией, используемой при вычислении псевдопроекции, является ортогональное проектирование $\boldsymbol{\pi}_i(\boldsymbol{x})$ точки $\boldsymbol{x}$ на гиперплоскость H_i, вычисляемое по известной формуле [20]:

$$\boldsymbol{\pi}_i(\boldsymbol{x}) = \boldsymbol{x} - \frac{\langle \boldsymbol{a}_i, \boldsymbol{x} \rangle - b_i}{\|\boldsymbol{a}_i\|^2} \boldsymbol{a}_i.$$

На основе ортогональной проекции строиться проекционное отображение $\boldsymbol{\varphi}_{\mathcal{J}}(\cdot)$:

$$\boldsymbol{\varphi}_{\mathcal{J}}(\boldsymbol{x}) = \frac{1}{|\mathcal{J}|} \sum_{j \in \mathcal{J}} \boldsymbol{\pi}_j(\boldsymbol{x}). \tag{26}$$

Известно [11], что отображение $\boldsymbol{\varphi}_{\mathcal{J}}(\boldsymbol{x})$ является непрерывным $L_{\mathcal{J}}$-фейеровским отображением, и последовательность точек

$$\left\{ \boldsymbol{x}_t = \boldsymbol{\varphi}_{\mathcal{J}}^t(\boldsymbol{x}_0) \right\}_{t=1}^{\infty}, \tag{27}$$

порождаемая этим отображением, начиная с произвольной точки $\boldsymbol{x}_0 \in \mathbb{R}^n$, сходится к точке, принадлежащей $L_{\mathcal{J}}$:

$$\boldsymbol{x}_t \to \tilde{\boldsymbol{x}} \in L_{\mathcal{J}}.$$

В соответствии с этим мы можем определить псевдопроекцию $\boldsymbol{\rho}_{\mathcal{J}}(\boldsymbol{x})$ точки $\boldsymbol{x}$ на линейное многообразие $L_{\mathcal{J}}$ как предельную точку последовательности (27):

$$\lim_{k \to \infty} \left\| \boldsymbol{\rho}_{\mathcal{J}}(\boldsymbol{x}) - \boldsymbol{\varphi}^k(\boldsymbol{x}) \right\| = 0.$$

В работе [19] было доказано, что в случае линейного многообразия псевдопроекция совпадает с ортогональной проекцией:

$$\boldsymbol{\rho}_{\mathcal{J}}(\boldsymbol{x}) = \boldsymbol{\pi}_{\mathcal{J}}(\boldsymbol{x}). \tag{28}$$

Реализация приближенного вычисления псевдопроекции представлена в виде алгоритма 1. Прокомментируем шаги этого алгоритма. На шаге 2 счетчик итераций t устанавливается в значение ноль. Шаг 3 задает начальное приближение $\boldsymbol{x}_0$. Шаг 4 открывает итерационный цикл вычисления псевдопроекции. Шаги 5–9 вычисляют по формуле (26) отображение $\boldsymbol{\varphi}_{\mathcal{J}}(\cdot)$ для текущего приближения $\boldsymbol{x}_k$. На шаге 10 вычисляется следующее приближение $\boldsymbol{x}_{k+1}$. Шаг 11 увеличивает счетчик итераций t на единицу. Шаг 12 завершает итерационный цикл, когда расстояние между соседними приближениями станет меньше малого параметра ε.

Алгоритм 1 Вычисление псевдопроекции $\boldsymbol{\rho}_{\mathcal{J}}(\boldsymbol{x})$

Require: $\mathcal{J} \subseteq \{1, \ldots, m\}$; $\mathcal{J} \neq \emptyset$

function $\boldsymbol{\rho}_{\mathcal{J}}(\boldsymbol{x})$
  $t := 0$
  $\boldsymbol{x}_0 := \boldsymbol{x}$
  repeat
    $\boldsymbol{\Sigma} := 0$
    for $j \in \mathcal{J}$ **do**
      $\boldsymbol{\Sigma} := \boldsymbol{\Sigma} + \boldsymbol{\pi}_j(\boldsymbol{x})$
    end for
    $\boldsymbol{\varphi} := \boldsymbol{\Sigma} / |\mathcal{J}|$
    $\boldsymbol{x}_{(t+1)} := \boldsymbol{\varphi}$
    $t := t + 1$
  until $\|\boldsymbol{x}_t - \boldsymbol{x}_{(t-1)}\| \leqslant \varepsilon$
  return $\boldsymbol{x}_t$
end function

Теперь рассмотрим функцию $\mathrm{Jump}(\boldsymbol{x}, \boldsymbol{d})$, осуществляющую перемещение по направлению вектора $\boldsymbol{d}$ от допустимой точки $\boldsymbol{x}$ к допустимой точке, максимально удаленной от $\boldsymbol{x}$. Основной элементарной операцией, используемой для перемещения, является d-проекция $\boldsymbol{\gamma}_i^{\boldsymbol{d}}(\boldsymbol{x})$ (косоугольная проекция) точки $\boldsymbol{x}$ на гиперплоскость H_i по направлению вектора $\boldsymbol{d}$, вычисляемая по формуле

$$\boldsymbol{\gamma}_i^{\boldsymbol{d}}(\boldsymbol{x}) = \begin{cases} \boldsymbol{x} - \frac{\langle \boldsymbol{a}_i, \boldsymbol{x} \rangle - b_i}{\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle} \boldsymbol{d}, & \text{если} \quad \langle \boldsymbol{a}_i, \boldsymbol{d} \rangle \neq 0; \\ \infty, & \text{если} \quad \langle \boldsymbol{a}_i, \boldsymbol{d} \rangle = 0 \end{cases} \tag{29}$$

(см. определение 1 и утверждение 2 в [19]). Различные случаи построения d-проекции $\boldsymbol{\gamma}_i^{\boldsymbol{d}}(\boldsymbol{x})$ показаны на рис. 2. Реализация функции $\mathrm{Jump}(\boldsymbol{x}, \boldsymbol{d})$ представлена в виде алгоритма 2. Для

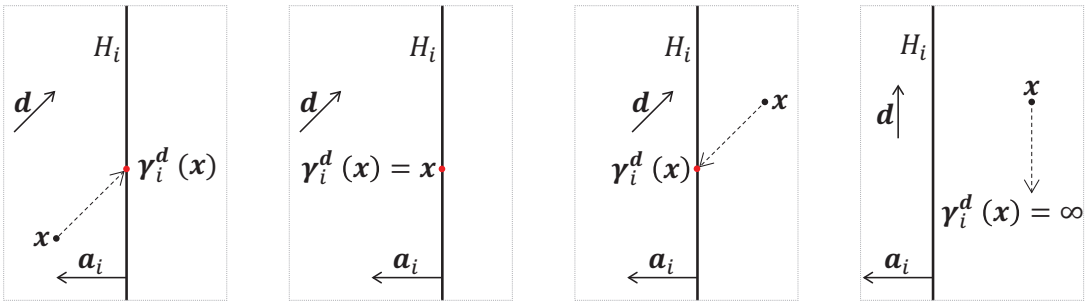

Рис. 2: Различные случаи построения d-проекции $\boldsymbol{\gamma}_i^{\boldsymbol{d}}(\boldsymbol{x})$:
а) $\langle \boldsymbol{a}_i, \boldsymbol{x} \rangle - b_i > 0$; б) $\langle \boldsymbol{a}_i, \boldsymbol{x} \rangle - b_i = 0$; в) $\langle \boldsymbol{a}_i, \boldsymbol{x} \rangle - b_i < 0$; г) $\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle = 0$.

работы алгоритма необходимо, чтобы точка $\boldsymbol{x}$ принадлежала допустимому многограннику M и выполнялось условие

$$\forall j \in \{1, \ldots l\} : \boldsymbol{x} + \boldsymbol{d} \in H_{k+j}, \tag{30}$$

которое означает, что точка, получающаяся путем прибавления вектора $\boldsymbol{d}$ к точке $\boldsymbol{x}$, должна принадлежать всем базисным гиперплоскостям (6). Другими словами, точка $(\boldsymbol{x} + \boldsymbol{d})$

Алгоритм 2 Реализация функции Jump$(\boldsymbol{x}, \boldsymbol{d})$

Require: $\boldsymbol{x} \in M$; $\forall j \in \{1, \ldots, l\} : \boldsymbol{x} + \boldsymbol{d} \in H_{k+j}$

1: **function** Jump$(\boldsymbol{x}, \boldsymbol{d})$
2: $\quad \boldsymbol{x}_{min} := \boldsymbol{x}$
3: $\quad$ **for** $i = 1 \ldots k$ **do**
4: $\quad\quad$ **if** $\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle > 0$ **then**
5: $\quad\quad\quad \boldsymbol{x}_\gamma := \boldsymbol{\gamma}_i^{\boldsymbol{d}}(\boldsymbol{x})$
6: $\quad\quad\quad$ **if** $\|\boldsymbol{x}_\gamma - \boldsymbol{x}\| < \|\boldsymbol{x}_{min} - \boldsymbol{x}\|$ **then**
7: $\quad\quad\quad\quad \boldsymbol{x}_{min} := \boldsymbol{x}_\gamma$
8: $\quad\quad\quad$ **end if**
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: $\quad$ **return** $\boldsymbol{x}_{min}$
12: **end function**

обязана удовлетворять всем уравнениям из системы ограничений (1). Схема работы алгоритма 2 представлена на рис. 3. В примере, показанном на рисунке, система ограничений

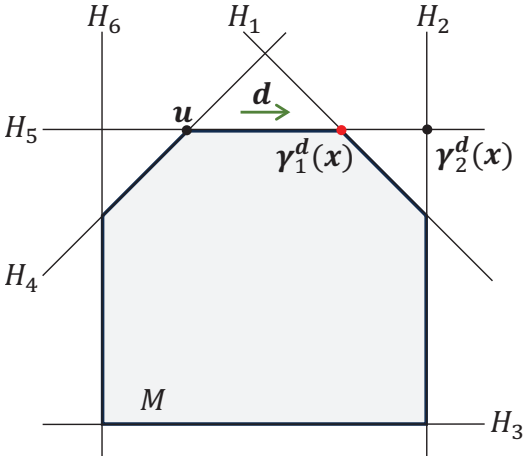

Рис. 3: Действие функции Jump:
Jump$(\boldsymbol{x}, \boldsymbol{d}) = \boldsymbol{\gamma}_1^{\boldsymbol{d}}(\boldsymbol{u})$.

не включает в себя уравнения. Граничные гиперплоскости H_3–H_6 не удовлетворяют неравенству

$$\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle > 0,$$

проверяемому на шаге 4 алгоритма 2. Это означает, что точка $(\boldsymbol{x} + \lambda \boldsymbol{d})$ будет принадлежать полупространствам $\hat{H}_3$–$\hat{H}_6$ при любом положительном λ. Среди оставшихся граничных гиперплоскостей $\hat{H}_1$ и $\hat{H}_2$ на шагах 5–8 алгоритма 2 выбирается d-проекция, в результате которой получается точка, находящаяся на минимальном расстоянии от точки $\boldsymbol{x}$. В примере на рис. 3 это — проекция на гиперплоскость H_1.

Реализация функции Move$(\boldsymbol{v})$, осуществляющей переход от вершины $\boldsymbol{v}$ к смежной оптимальной вершине $\boldsymbol{v}_{max}$, представлена в виде алгоритма 3. Оптимальность означает, что алгоритм всегда выбирает то ребро, инцидентное вершине $\boldsymbol{v}$, конечная точка которого имеет максимальное значение целевой функции. В реализации алгоритма 3 используется генератор TWIDDLE [21], строящий все возможные сочетания из элементов заданного конечного

Алгоритм 3 Переход к следующей вершине

Require: $\boldsymbol{v} \in \Gamma(M);\ \exists\, \mathcal{J} \subseteq \{1, \ldots, k\} \left(|\mathcal{J}| = n - l \wedge \boldsymbol{v} \in \bigcap\limits_{i \in \mathcal{J}} H_i \right)$

1: **function** Move($\boldsymbol{v}$)
2: $\quad \boldsymbol{u} := \boldsymbol{v} + \delta \boldsymbol{c} / \|\boldsymbol{c}\|$
3: $\quad \mathcal{V} := \emptyset$
4: $\quad$ **for** $i = 1 \ldots k$ **do**
5: $\quad\quad$ **if** $\langle \boldsymbol{a}_i, \boldsymbol{v} \rangle = b_i$ **then**
6: $\quad\quad\quad \mathcal{V} := \mathcal{V} \cup \{i\}$
7: $\quad\quad$ **end if**
8: $\quad$ **end for**
9: $\quad F_{max} := -\infty$
10: $\quad \boldsymbol{v}_{max} := \boldsymbol{v}$
11: $\quad$ TWIDDLE_Init($\mathcal{V}, n - l - 1$)
12: $\quad$ **repeat**
13: $\quad\quad \mathcal{J} :=$ TWIDDLE_NextCombination()
14: $\quad\quad \mathcal{J} := \mathcal{J} \cup \{k + 1, \ldots, k + l\}$
15: $\quad\quad \boldsymbol{w} := \boldsymbol{\rho}_{\mathcal{J}}(\boldsymbol{u})$
16: $\quad\quad \boldsymbol{d} := \boldsymbol{w} - \boldsymbol{v}$
17: $\quad\quad \boldsymbol{v}_{next} :=$ Jump($\boldsymbol{v}, \boldsymbol{d}$)
18: $\quad\quad$ **if** $\langle \boldsymbol{c}, \boldsymbol{v}_{next} \rangle > F_{max}$ **then**
19: $\quad\quad\quad \boldsymbol{v}_{max} := \boldsymbol{v}_{next}$
20: $\quad\quad\quad F_{max} := \langle \boldsymbol{c}, \boldsymbol{v}_{max} \rangle$
21: $\quad\quad$ **end if**
22: $\quad$ **until** TWIDDLE_Done()
23: $\quad$ **return** $\boldsymbol{v}_{max}$
24: **end function**

множества. Дадим краткий комментарий по шагам алгоритма 3. Начальная точка $\boldsymbol{v}$ должна являться вершиной, то есть удовлетворять условиям утверждения 2. На шаге 2 строится точка $\boldsymbol{u}$ (см. рис. 1б). Здесь δ — «большой» положительный параметр: чем больше δ, тем точнее будет вычислен вектор перемещения $\boldsymbol{d}$, используемый функцией Jump на шаге 17. Шаги 4–8 строят множество $\mathcal{V}$, включающее в себя индексы всех граничных гиперплоскостей, проходящих через точку $\boldsymbol{v}$. Шаг 9 присваивает переменной F_{max}, обозначающей максимум целевой функции, значение $-\infty$. На шаге 10 вектору $\boldsymbol{v}_{max}$ в качестве начального значения присваивается вектор $\boldsymbol{v}$. Шаг 11 устанавливает генератор сочетаний TWIDDLE в начальное состояние. Шаг 12 открывает цикл **repeat**/**until**, вычисляющий оптимальное ребро и следующую вершину $\boldsymbol{v}_{max}$. На шаге 13 генератор TWIDDLE возвращает очередное сочетание $(n - l - 1)$ из k индексов граничных гиперплоскостей, именуемое как множество $\mathcal{J}$. На шаге 14 в множество $\mathcal{J}$ добавляются индексы всех базисных гиперплоскостей. В результате получается множество, содержащее $n - 1$ индексов различных гиперплоскостей, пересечение которых образует прямую, соответствующую некоторому ребру, инцидентному вершине $\boldsymbol{v}$. На шаге 15 из точки $\boldsymbol{u}$ на эту прямую строится псевдопроекция $\boldsymbol{w}$, которая в этом случае будет совпадать с ортогональной проекцией. Шаг 16 вычисляет вектор движения по ребру $\boldsymbol{d}$ в направлении увеличения значения целевой функции. Шаг 17 осуществляет переход в конечную точку ребра $\boldsymbol{v}_{next}$. Если значение целевой функции в этой точке больше F_{max}, то на шагах 18-21 обновляются значения $\boldsymbol{v}_{max}$ и F_{max}. Шаг 22 завершает выполнение цикла **repeat**/**until** после того, как генератор TWIDDLE выдал по-

Алгоритм 4 AlEM (Along Edges Movement)

Require: $\boldsymbol{v}_0 \in \Gamma(M);\ \exists\, \mathcal{J} \subseteq \{1,\ldots,k\} \left(|\mathcal{J}| = n - l \wedge \boldsymbol{v}_0 \in \bigcap\limits_{i \in \mathcal{J}} H_i \right)$

1: $t := 0$
2: **repeat**
3: $\quad \boldsymbol{v}_{t+1} := \mathrm{Move}(\boldsymbol{v}_t)$
4: $\quad t := t + 1$
5: **until** $\boldsymbol{v}_t = \boldsymbol{v}_{t-1}$
6: **output** $\boldsymbol{v}_t$
7: **stop**

следнее сочетание. Шаг 23 возвращает вектор $\boldsymbol{v}_{max}$ в качестве результата функции $\mathrm{Move}(\boldsymbol{v})$. Отметим, что корректность алгоритма 3 обеспечивается утверждением 3.

Реализация алгоритма AlEM, строящего оптимальный путь по ребрам допустимого многогранника к решению задачи ЛП (1), представлена в виде алгоритма 4. Начальная точка $\boldsymbol{v}_0$ должна являться вершиной допустимого многогранника, то есть удовлетворять условиям утверждения 2. На шаге 1 счетчик итераций t устанавливается в значение 0. Шаг 2 открывает цикл **repeat**/**until**, строящий с помощью функции $Move(\cdot)$ оптимальный путь по ребрам допустимого многогранника от начальной точки $\boldsymbol{v}_0$ к решению задачи ЛП. Шаг 5 завершает выполнение цикла **repeat**/**until**, когда следующее приближение $\boldsymbol{v}_k$ совпадет с предыдущим $\boldsymbol{v}_{k-1}$. Шаг 6 выводит последнее приближение в качестве решения задачи ЛП. Шаг 7 завершает работу алгоритма AlEM.

Как уже было сказано выше, алгоритм AlEM требует в качестве начальной точки вершину допустимого многогранника M. Мы получаем такую вершину в два этапа. На первом этапе используется алгоритм BIP (Block-iterative projection) [17], реализованный нами в виде параллельной программы на языке C++, исходные тексты которой доступны в репозитории GitHub[1]. На вход алгоритму BIP подается произвольная точка $\boldsymbol{x}_0 \in \mathbb{R}^n$, не являющаяся внутренней по отношению к допустимому многограннику M. В случае задачи ЛП в качестве такой точки всегда может служить точка $\mathbf{0}$. Алгоритм BIP возвращает точку $\boldsymbol{z}_0$, лежащую на границе допустимого многогранника M. На втором этапе мы применяем разработанный нами параллельный алгоритм VeSP (Vertex Search by Projecting), исходные коды которого также доступны в репозитории GitHub[2]. В качестве начальной точки алгоритм VeSP использует произвольную граничную точку $\boldsymbol{z}_0$ допустимого многогранника M. Затем алгоритм VeSP идет по граням допустимого многогранника M в направлении уменьшения их размерности и за конечное число итераций доходит до вершины допустимого многогранника M, которая и служит начальной точкой $\boldsymbol{v}_0$ алгоритма AlEM. Отметим, что алгоритм VeSP использует подходы, примененные при разработке алгоритма AlFaMove[18].

4. Параллельная версия алгоритма AlEM

Наиболее ресурсоемкой операцией функции $Move(\cdot)$ (см. алгоритм 3) является операция вычисления псевдопроекции на линейное многообразие на шаге 15. Выполнение вектор-функции $\boldsymbol{\rho}_{\mathcal{J}}(\cdot)$ заключается в последовательном применении проекционного отображения $\varphi(\cdot)$, задаваемого формулой (26), к исходной точке (см. алгоритм 1, реализующий вычисление псевдопроекции). Известно, что в случае больших задач ЛП проекционный метод может потребовать значительных временных затрат [22]. Кроме того, следует отметить,

[1] `https://github.com/leonid-sokolinsky/BIP`
[2] `https://github.com/leonid-sokolinsky/VeSP`

что алгоритм 3 в цикле на шаге 13 перебирает все сочетания из k по $(n-l-1)$ индексов граничных гиперплоскостей. Число таких сочетаний вычисляется по формуле

$$C_k^{n-l-1} = \frac{k!}{(k-n+l+1)!(n-l-1)!}$$

и может достигать больших значений при решении практических задач ЛП. Комбинаторный перебор может потребовать использования суперкомпьютерных мощностей. Потому мы разработали параллельную версию алгоритма AlEM. Работа параллельных узлов организуется по схеме SIMD (single instruction, multiple data): все процессорные узлы выполняют один и тот же код, но над различными данными. Сочетания, конструируемые генератором TWIDDLE, распределяются между процессорными узлами по принципу round-robin. Для этого мы преобразовали функцию $Move(\cdot)$ (алгоритм 3) в функцию $RoundRobinMove(\cdot)$, реализация которой представлена в виде алгоритма 5. Кроме исходной вершины $\boldsymbol{v}$, в функцию

Алгоритм 5 Переход к следующей вершине по принципу round-robin

Require: $\boldsymbol{v} \in \Gamma(M);\ \exists\, \mathcal{J} \subseteq \{1,\dots,k\} \left(|\mathcal{J}| = n-l \wedge \boldsymbol{v} \in \bigcap\limits_{i\in\mathcal{J}} H_i \right)$

1: **function** RoundRobinMove($\boldsymbol{v}$, *myRank*, *numOfNodes*)
2: $\quad$ *twiddleNo* := *myRank*
3: $\quad$ $\boldsymbol{u} := \boldsymbol{v} + \delta\boldsymbol{c}/\|\boldsymbol{c}\|$
4: $\quad$ $\mathcal{V} := \emptyset$
5: $\quad$ **for** $i = 1 \dots k$ **do**
6: $\quad\quad$ **if** $\langle \boldsymbol{a}_i, \boldsymbol{v} \rangle = b_i$ **then**
7: $\quad\quad\quad$ $\mathcal{V} := \mathcal{V} \cup \{i\}$
8: $\quad\quad$ **end if**
9: $\quad$ **end for**
10: $\quad$ $F_{max} := -\infty$
11: $\quad$ $\boldsymbol{v}_{max} := \boldsymbol{v}$
12: $\quad$ TWIDDLE_Init($\mathcal{V}, n-l-1$)
13: $\quad$ $\mathcal{J}$:= TWIDDLE_Combination(*twiddleNo*)
14: $\quad$ **while not** TWIDDLE_Done() **do**
15: $\quad\quad$ $\mathcal{J} := \mathcal{J} \cup \{k+1, \dots, k+l\}$
16: $\quad\quad$ $\boldsymbol{w} := \boldsymbol{\rho}_{\mathcal{J}}(\boldsymbol{u})$
17: $\quad\quad$ $\boldsymbol{d} := \boldsymbol{w} - \boldsymbol{v}$
18: $\quad\quad$ $\boldsymbol{v}_{next}$:= Jump($\boldsymbol{v}, \boldsymbol{d}$)
19: $\quad\quad$ **if** $\langle \boldsymbol{c}, \boldsymbol{v}_{next} \rangle > F_{max}$ **then**
20: $\quad\quad\quad$ $\boldsymbol{v}_{max} := \boldsymbol{v}_{next}$
21: $\quad\quad\quad$ $F_{max} := \langle \boldsymbol{c}, \boldsymbol{v}_{max} \rangle$
22: $\quad\quad$ **end if**
23: $\quad\quad$ *twiddleNo* := *twiddleNo* + *numOfNodes*
24: $\quad\quad$ $\mathcal{J}$:= TWIDDLE_Combination(*twiddleNo*)
25: $\quad$ **end while**
26: $\quad$ **return** $\boldsymbol{v}_{max}$
27: **end function**

$RoundRobinMove(\cdot)$ передаются номер процессорного узла *myRank* и количество процессорных узлов *numOfNodes*. Функция без параметров TWIDDLE_NextCombination() преобразована в функцию с одним параметром TWIDDLE_Combination(*twiddleNo*), возвращающую сочетание с порядковым номером *twiddleNo*. Если значение переменной *twiddleNo*

Алгоритм 6 Параллельная версия алгоритма AlEM

Require: $\boldsymbol{v}_0 \in \Gamma(M);\ \exists\, \mathcal{J} \subseteq \{1,\ldots,k\} \left(|\mathcal{J}| = n - l \wedge \boldsymbol{v}_0 \in \bigcap\limits_{i \in \mathcal{J}} H_i \right)$

1: $t := 0$
2: *myRank* := SYSTEM_Rank()
3: *numOfNodes* := SYSTEM_NumberOfNodes()
4: **repeat**
5: $\quad \boldsymbol{v}_{t+1} :=$ RoundRobinMove($\boldsymbol{v}_t$, *myRank*, *numOfNodes*)
6: $\quad rank_{max} :=$ SYSTEM_AllReduceMax ($\langle \boldsymbol{c}, \boldsymbol{v}_{t+1} \rangle$)
7: $\quad$ SYSTEM_Broadcast($rank_{max}, \boldsymbol{v}_{t+1}$)
8: $\quad t := t + 1$
9: **until** $\boldsymbol{v}_t = \boldsymbol{v}_{t-1}$
10: **if** *myRank* $= rank_{max}$ **then**
11: $\quad$ **output** $\boldsymbol{v}_t$
12: **end if**
13: **stop**

превышает общее число сочетаний, то функция TWIDDLE_Combination(*twiddleNo*) возвращает последнее сочетание, а функция TWIDDLE_Done() возвращает значение «истина». Во всех остальных случаях функция TWIDDLE_Done() возвращает значение «ложь». В соответствии с принципом round-robin переменной *twiddleNo* на шаге 2 присваивается номер процессорного узла[3], а затем на каждой итерации цикла **while**, она увеличивается на количество процессорных узлов *numOfNodes* (шаг 23). Таким образом, множества сочетаний, обрабатываемых на различных процессорных узлах, не будут пересекаться, а их объединение будет равно множеству всех возможных сочетаний.

Параллельная версия алгоритма AlEM представлена в виде алгоритма 6. Этот алгоритм выполняется на всех используемых процессорных узлах. Кратко прокомментируем шаги параллельного алгоритма 6. Для его работы по-прежнему требуется, чтобы начальная точка $\boldsymbol{v}_0$ была вершиной допустимого многогранника M. На шаге 1 счетчику циклов t присваивается значение 0. На шаге 2 вызывается системная функция SYSTEM_Rank(), возвращающая номер процессорного узла, который сохраняется в переменной *myRank*. На шаге 3 вызывается системная функция SYSTEM_NumberOfNodes(), возвращающая количество используемых процессорных узлов. Полученное значение сохраняется в ременной *numOfNodes*. Шаг 4 открывает цикл **repeat**/**until**, строящий оптимальный путь по ребрам допустимого многогранника от начальной точки $\boldsymbol{v}_0$ к решению задачи ЛП. На шаге 5 каждый узел по принципу round-robin перебирает свои комбинации и находит следующую вершину $\boldsymbol{v}_{t+1}$ с локальным максимумом целевой функции. На шаге 6 выполняется системная функция SYSTEM_AllReduceMax ($\langle c, \boldsymbol{v}_{t+1} \rangle$), возвращающая номер узла, на котором целевая функция достигает наибольшего значения в точках локальных максимумов $\boldsymbol{v}_{t+1}$. Шаг 7 выполняет системную функцию SYSTEM_Broadcast($rank_{max}, \boldsymbol{v}_{t+1}$), в результате которой процессорный узел с номером $rank_{max}$ передает остальным узлам вектор $\boldsymbol{v}_{t+1}$. Шаг 9 завершает выполнение цикла **repeat**/**until**, когда следующее приближение $\boldsymbol{v}_k$ совпадет с предыдущим $\boldsymbol{v}_{k-1}$. Это означает, что достигнут максимум целевой фцнкции. На шагах 10–12 процессорный узел с номером *myRank* выводит последнее приближение в качестве решения задачи ЛП. Шаг 13 завершает работу параллельного алгоритма AlEM.

[3]Мы предполагаем, что нумерация процессорных узлов начинается с нуля, равно как и нумерация сочетаний, выдаваемых генератором TWIDDLE, также начинается с нуля.

5. Реализация и вычислительные эксперименты

Мы реализовали параллельную версию алгоритма AlEM на языке C++ с использованием библиотеки параллельных вычислений MPI 3.0. Исходные коды параллельной программы доступны в репозитории GitHub по адресу `https://github.com/zhulevae/AlEM`.

С помощью разработанной параллельной программы мы исследовали масштабируемость алгоритма AlEM. В экспериментах мы использовали параметризованную задачу ЛП «гиперкуб с отсеченной вершиной», для которой размерность пространства n является параметром. Ограничения этой задачи содержат $2n + 1$ неравенств следующего вида:

$$\begin{cases} x_1 & & & & & \leqslant & 200 \\ & x_2 & & & & \leqslant & 200 \\ & & & & & \vdots & \\ & & & x_n & & \leqslant & 200 \\ x_1 & + & x_2 & \cdots & + \quad x_n & \leqslant & 200(n-1)+100 \\ x_1 \geqslant 0, & & x_2 \geqslant 0, & \cdots & , x_n \geqslant 0. & & \end{cases}$$

Градиент целевой функции задается вектором

$$\boldsymbol{c} = (1, 2, \ldots, n)\,.$$

Задача предполагает нахождение максимума целевой функции и имеет единственное решение в точке $(100, 200, \ldots, 200)$ со значением целевой функции, равным $100(n^2 + n - 1)$. Для произвольного n эта задача может быть получена в формате MTX [23] с помощью генератора FRaGenLP[4] [24], если в качестве количества случайных неравенств задать 0. Эти задачи ЛП для различных n доступны по адресу `https://github.com/leonid-sokolinsky/Set-of-LP-Problems/tree/main/Rnd-LP` под именами `lp_rnd<n>-0`, где в качестве `<n>` указана размерность пространства.

Масштабные вычислительные эксперименты проводились на вычислительном кластере «Торнадо ЮУрГУ» [25], характеристики которого представлены в табл. 1. Для сборки программы использовался компилятор g++, распространяемый в рамках пакета компиляторов GCC 10, и библиотека Intel MPI 5.0. Компиляция выполнялась с опцией оптимизации O3.

Таблица 1: Характеристики кластера «Торнадо ЮУрГУ»

Параметр	Значение
Количество доступных процессорных узлов	260
Процессоры	Intel Xeon X5680 (6 cores, 3.33 GHz)
Число процессоров на узел	2
Память на узел	24 GB DDR3
Соединительная сеть	InfiniBand QDR (40 Gbit/s)
Операционная система	Linux CentOS

В описанной среде нами была выполнена серия вычислительных экспериментов, в которой для задач ЛП различной размерности исследовались ускорение и параллельная эффективность в зависимости от количества используемых процессорных узлов кластера. На

[4] `https://github.com/leonid-sokolinsky/FRaGenLP`

каждом процессорном узле выполнялся только один MPI-процесс. Результаты этих экспериментов представлены на рис. 4. Ускорение $\alpha(P)$ определялось как отношение времени $T(1)$

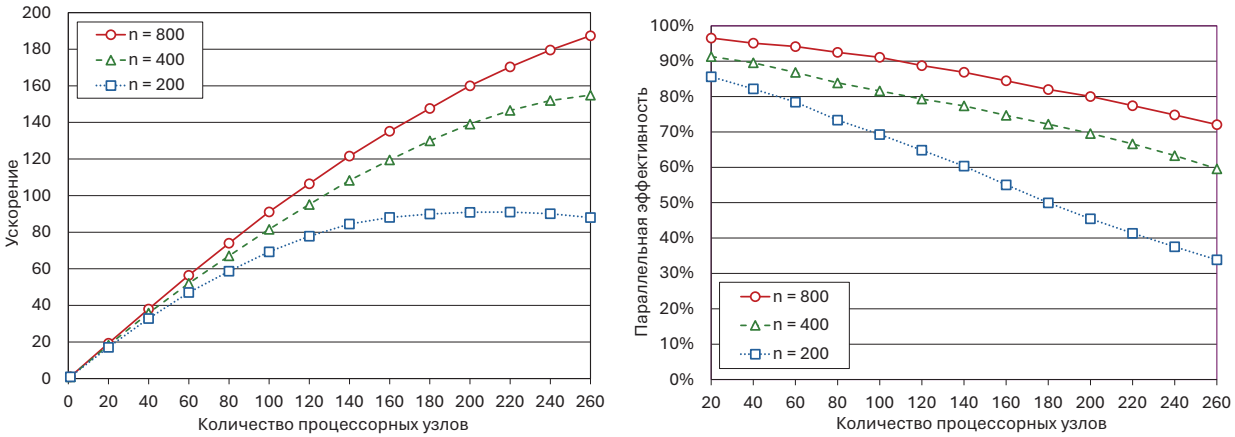

Рис. 4: Ускорение и эффективность параллельной реализации алгоритма AlEM.

решения задачи на конфигурации с одним процессорным узлом ко времени $T(P)$ решения той же задачи на конфигурации с P процессорными узлами:

$$\alpha(P) = \frac{T(1)}{T(P)}.$$

Параллельная эффективность вычислялась по формуле

$$\epsilon(P) = \frac{T(1)}{P \cdot T(P)}.$$

Вычисления проводились для следующих размерностей: 200, 400 и 800. Число ограничений соответственно составило 401, 801 и 1601. Эксперименты показали, что алгоритм AlEM хорошо масштабируется на задачах больших размерностей. Задача размерности $n = 800$ показала на 260 процессорных узлах ускорение, близкое к линейному, с эффективностью распараллеливания 72%. Однако, при уменьшении размерности задачи граница масштабируемости снижается и эффективность распараллеливания падает. Так для задачи размерности $n = 400$ граница масштабируемости наблюдалась в районе 260 процессорных узлов с эффективностью распараллеливания 60%. А для задачи размерности $n = 400$ граница масштабируемости упала до 200 процессорных узлов. Параллельная эффективность в этом случае составила только 50%. Время решения задачи «гиперкуб с отсеченной вершиной» для различных размерностей приведено в табл. 2. В случае, когда достигалась

Таблица 2: Время решения задачи «гиперкуб с отсеченной вершиной»

Размерность	Кол-во узлов	Время (сек)
200	200	1
400	260	2.5
800	260	18.5

граница масштабируемости (размерности 200 и 400), время решения оказалось сравнимым

с симплекс-методом. Для решения задачи размерности 800 необходим вычислительный кластер с большим числом процессорных узлов, чем «Торнадо ЮУрГУ».

Заключение

В статье предложен новый масштабируемый алгоритм линейного программирования, получивший название «AlEM». Ключевой особенностью этого метода является построение оптимального пути по ребрам допустимого многогранника от начальной точки к решению задачи линейного программирования. Под оптимальным путем понимается путь в направлении максимального увеличения значений целевой функции. Практическая значимость предложенного метода состоит в том, что он открывает возможность применения искусственных нейронных сетей прямого распространения для решения нестационарных многомерных задач линейного программирования в режиме реального времени.

Теоретической основой алгоритма AlEM является операция построения псевдопроекции на линейные многообразия, формирующие ребра допустимого многогранника. Псевдопроекция реализуется на основе фейеровского процесса и является обобщением понятия метрической проекции на выпуклое множество. В случае линейного многообразия псевдопроекция сходится к точке ортогональной проекции. Приведен алгоритм проекционного типа для построения псевдопроекции на линейные многообразия, образуемые пересечением гиперплоскостей. Представлено формализованное описание алгоритма AlEM, строящего оптимальный путь по ребрам допустимого многогранника. В основе алгоритма AlEM лежит функция перехода по ребру допустимого многогранника из текущей вершины к следующей вершине с максимальным значением целевой функции. Дано формализованное описание этой функции.

Алгоритмы проекционного типа характеризуются низкой скоростью сходимости, зависящей от углов между гиперплоскостями, образующими линейное многообразие. Также отмечено, что при вычислении вектора движения возникает переборная задача комбинаторного типа, имеющая высокую временную сложность. Представлена параллельная версия алгоритма AlEM, ориентированная на кластерные вычислительные системы. Параллельная версия реализована на языке C++ с использованием библиотеки MPI. Проведены эксперименты по исследованию масштабируемости параллельной версии алгоритма AlEM на кластерной вычислительной системе. Вычислительные эксперименты показали, что задача линейного программирования, включающая 800 переменных и 1601 ограничение, демонстрирует ускорение близкое к линейному на 260 процессорных узлах кластера. В случае, когда достигается граница масштабируемости, время решения задачи ЛП методом AlEM сравнимо с симплекс-методом.

В качестве направлений дальнейших исследований выделим следующие. Для ускорения вычислений мы предполагаем использовать технологию OpenMP при вычислении псевдопроекции. Также мы планируем разработать новый, более эффективный метод построения пути по ребрам допустимого многогранника, приводящего к решению задачи линейного программирования. Основная идея состоит в построении многомерного образа задачи ЛП и использования искусственной нейронной сети прямого распространения для идентификации ребер допустимого многогранника. Это позволит отказаться от дорогостоящей операции псевдопроекции и выполнить решение большой задачи ЛП в режиме реального времени.

Литература

1. Xu Y. Solving Large Scale Optimization Problems in the Transportation Industry and Beyond Through Column Generation // Optimization in Large Scale Problems. Springer Optimization and Its Applications, vol 152. Springer, 2019. P. 269–292. DOI: 10.1007/978-3-030-28565-4_23/COVER.

2. Chung W. Applying large-scale linear programming in business analytics // 2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM). IEEE, 2015. P. 1860–1864. DOI: 10.1109/IEEM.2015.7385970.

3. Gondzio J., Gruca J.A., Hall J.A.J., *et al.* Solving large-scale optimization problems related to Bell's Theorem // Journal of Computational and Applied Mathematics. 2014. Vol. 263. P. 392–404. DOI: 10.1016/j.cam.2013.12.003.

4. Dantzig G.B. Linear programming and extensions. Princeton, N.J.: Princeton university press, 1998. 656 p.

5. Зоркальцев В.И., Мокрый И.В. Алгоритмы внутренних точек в линейной оптимизации // Сибирский журнал индустриальной математики. 2018. Т. 21, 1 (73). С. 11–20. DOI: 10.17377/sibjim.2018.21.102.

6. Kaczmarz S. Angenherte Auflsung von Systemen linearer Gleichungen // Bulletin International de l'Acadmie Polonaise des Sciences et des Lettres. Classe des Sciences Mathmatiques et Naturelles. Srie A, Sciences Mathmatiques. 1937. Vol. 35. P. 355–357.

7. Cimmino G. Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari // La Ricerca Scientifica, XVI, Series II, Anno IX, 1. 1938. P. 326–333.

8. Agmon S. The relaxation method for linear inequalities // Canadian Journal of Mathematics. 1954. Vol. 6. P. 382–392. DOI: 10.4153/CJM-1954-037-2.

9. Censor Y., Chen W., Combettes P.L., *et al.* On the effectiveness of projection methods for convex feasibility problems with linear inequality constraints // Computational Optimization and Applications. 2011. Vol. 51, no. 3. P. 1065–1088. DOI: 10.1007/S10589-011-9401-7.

10. Ерёмин И.И. Применение метода фейеровских приближений к решению задач выпуклого программирования с негладкими ограничениями // Журнал вычислительной математики и математической физики. 1969. Т. 9, № 5. С. 1153–1160.

11. Васин В.В., Ерёмин И.И. Операторы и итерационные процессы фейеровского типа. Теория и приложения. Екатеринбург: УрО РАН, 2005. 211 с.

12. Ерёмин И.И., Попов Л.Д. Фейеровские процессы в теории и практике: обзор последних результатов // Известия вузов. Математика. 2009. № 1. С. 44–65. URL: https://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=ivm&paperid=1253.

13. Nurminski E.A. Single-projection procedure for linear optimization // Journal of Global Optimization. 2016. Vol. 66, no. 1. P. 95–110. DOI: 10.1007/S10898-015-0337-9.

14. Censor Y. Can linear superiorization be useful for linear optimization problems? // Inverse Problems. 2017. Vol. 33, no. 4. P. 044006. DOI: 10.1088/1361-6420/33/4/044006.

15. Соколинский Л.Б., Соколинская И.М. О новой версии апекс-метода для решения задач линейного программирования // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2023. Т. 12, № 2. С. 5–46. DOI: 10.14529/cmse230201.

16. Deutsch F., Hundal H. The rate of convergence for the cyclic projections algorithm I: Angles between convex sets // Journal of Approximation Theory. 2006. Vol. 142, no. 1. P. 36–55. DOI: 10.1016/J.JAT.2006.02.005.

17. Aharoni R., Censor Y. Block-iterative projection methods for parallel computation of solutions to convex feasibility problems // Linear Algebra and its Applications. 1989. Vol. 120. P. 165–175. DOI: 10.1016/0024-3795(89)90375-3.

18. Соколинский Л.Б., Ольховский Н.А., Соколинская И.М. Численная реализация метода поверхностного движения для решения задач линейного программирования // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2024. Т. 13, № 3. С. 5–31. DOI: 10.14529/cmse240301.

19. Ольховский Н.А., Соколинский Л.Б. О новом методе линейного программирования // Вычислительные методы и программирование. 2023. Т. 24, № 4. С. 408–429. DOI: 10.26089/NumMet.v24r428.

20. Мальцев А.И. Основы линейной алгебры. Москва: Наука. Главная редакция физико-математической литературы, 1970. 402 с.

21. Chase P.J. Algorithm 382: combinations of M out of N objects [G6] // Communications of the ACM. 1970. Vol. 13, no. 6. P. 368. DOI: 10.1145/362384.362502.

22. Gould N.I. How good are projection methods for convex feasibility problems? // Computational Optimization and Applications. 2008. Vol. 40, no. 1. P. 1–12. DOI: 10.1007/S10589-007-9073-5.

23. Boisvert R.F., Pozo R., Remington K.A. The Matrix Market Exchange Formats: Initial Design: tech. rep. / NISTIR 5935. National Institute of Standards; Technology. Gaithersburg, MD, 1996. P. 14. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir5935.pdf.

24. Соколинский Л.Б., Соколинская И.М. О генерации случайных задач линейного программирования на кластерных вычислительных системах // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2021. Т. 10, № 2. С. 38–52. DOI: 10.14529/cmse210103.

25. Dolganina N., Ivanova E., Bilenko R., Rekachinsky A. HPC Resources of South Ural State University // Parallel Computational Technologies. PCT 2022. Communications in Computer and Information Science, vol. 1618 / ed. by L. Sokolinsky, M. Zymbler. City: Cham: Springer, 2022. P. 43–55. DOI: 10.1007/978-3-031-11623-0_4.