2026, **27** (1), 1–18. doi 10.26089/NumMet.v27r101

doi 10.26089/NumMet.v27r101

УДК 519.852;
514.172.45;
004.032.24

О поиске оптимальной вершины на многограннике допустимых решений задачи линейного программирования

А. Э. Жулев
Южно-Уральский государственный университет (национальный исследовательский университет), Челябинск, Российская Федерация
ORCID: 0009-0004-6647-0934, e-mail: zhulevae@susu.ru

Л. Б. Соколинский
Южно-Уральский государственный университет (национальный исследовательский университет), Челябинск, Российская Федерация
ORCID: 0000-0001-9997-3918, e-mail: leonid.sokolinsky@susu.ru

Аннотация: Представлен новый масштабируемый алгоритм AlEM для решения задач линейного программирования на кластерных вычислительных системах. В отличие от классического симплекс-метода, алгоритм AlEM на каждом шаге исследует все исходящие из текущей вершины ребра многогранника допустимых решений, что гарантирует построение субоптимального пути и полностью исключает возможность зацикливания на вырожденных задачах. Приведено формализованное описание алгоритма и его параллельной реализации с использованием библиотеки MPI. Экспериментально исследована масштабируемость параллельной версии алгоритма на реальных и модельных задачах линейного программирования. Результаты вычислительных экспериментов подтверждают высокую параллельную эффективность алгоритма AlEM, которая сохраняется на уровне не менее 46% при использовании до 200 процессорных узлов. Показано, что алгоритм AlEM является возможной альтернативой существующим методам линейного программирования для высокопроизводительных вычислений.

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

Для цитирования: Жулев А.Э., Соколинский Л.Б. О поиске оптимальной вершины на многограннике допустимых решений задачи линейного программирования // Вычислительные методы и программирование. 2026. **27**, № 1. 1–18. doi 10.26089/NumMet.v27r101.

2026, **27** (1), 1–18. doi 10.26089/NumMet.v27r101

Finding optimal vertex on polytope of feasible solutions to linear programming problem

Alexander E. Zhulev
South Ural State University (National Research University),
Chelyabinsk, Russia
ORCID: 0009-0004-6647-0934, e-mail: zhulevae@susu.ru

Leonid B. Sokolinsky
South Ural State University (National Research University),
Chelyabinsk, Russia
ORCID: 0000-0001-9997-3918, e-mail: leonid.sokolinsky@susu.ru

Abstract: A new scalable AlEM algorithm for linear programming on cluster computing systems is presented. Unlike the classical simplex method, the AlEM algorithm, at each step, examines all edges coming from the current vertex of the polytope of feasible solutions, which guarantees the construction of a suboptimal path and completely prevents circling in degenerate problems. A formalized description of the algorithm and its parallel implementation using the MPI library is given. The scalability of the parallel version of the algorithm on real and synthetic linear programming problems has been experimentally investigated. The results of computational experiments confirm the high parallel efficiency of the AlEM algorithm, which remains at least 46% when using up to 200 processor nodes. It is shown that the AlEM algorithm is a possible alternative to existing linear programming methods for high-performance computing.

Keywords: linear programming, AlEM algorithm, projection method, parallel implementation, MPI, cluster computing system, scalability evaluation.

For citation: A. E. Zhulev, L. B. Sokolinsky, “Finding optimal vertex on polytope of feasible solutions to linear programming problem,” Numerical Methods and Programming. **27** (1), 1–18 (2026). doi 10.26089/NumMet.v27r101.

1. Введение. Линейное программирование (ЛП) является одной из наиболее востребованных оптимизационных математических моделей, используемых для решения практических задач в индустрии, экономике, науке и других областях [1–3]. К настоящему моменту разработано и исследовано большое количество методов и алгоритмов ЛП [4]. Наиболее популярным и широко используемым является симплекс-метод [5], позволяющий эффективно решать широкий класс реальных задач ЛП с числом переменных до 50000 [6]. Однако симплекс-методу присущи определенные недостатки, среди которых выделим следующие: чувствительность к вырожденным задачам и ограниченная масштабируемость.

Используя терминологию симплекс-метода, вырожденную задачу ЛП можно определить как задачу, у которой существуют два различных базиса, соответствующих одному и тому же допустимому базисному решению [7]. В этом случае итерации симплекс-метода могут выполняться без изменения значения целевой функции, что приводит к замедлению (stalling) работы алгоритма. Более того, может возникнуть ситуация, когда при выполнении подобных “холостых” итераций симплекс-метод придет к базису, который уже был построен ранее. Данный эффект называется зацикливанием (cycling). Известно, что зацикливание может встречаться при решении практических задач ЛП [8, 9]. Существует несколько известных подходов, позволяющих предотвратить зацикливание симплекс-метода, однако эти подходы оказываются эффективными не во всех случаях [10].

На практике встречаются большие задачи ЛП, включающие в себя десятки, а иногда и сотни тысяч ограничений и переменных [2, 11, 12]. Такие задачи могут потребовать значительных процессорных ресурсов. В соответствии с этим научным сообществом были предприняты интенсивные попытки по разработке различных подходов к распараллеливанию классического симплекс-метода и его вариаций. Определенные успехи были достигнуты в распараллеливании некоторых классов задач ЛП с сильно разреженными матрицами и матрицами блочно-диагонального типа. Однако реальная масштабируемость параллельных

2026, **27** (1), 1–18. doi 10.26089/NumMet.v27r101

реализаций симплекс-метода в большинстве случаев не превышала 16–32 процессорных узлов на кластерной вычислительной системе с распределенной памятью. До настоящего времени не удалось создать параллельную реализацию симплекс-метода, которая в общем случае превосходила бы по быстродействию последовательные коммерческие решатели [13].

В данной работе предлагается новый масштабируемый алгоритм проекционного типа AlEM (Along Edges Movement) для линейного программирования на многопроцессорных системах с распределенной памятью. Подобно симплекс-методу, алгоритм AlEM строит путь на границе многогранника допустимых решений задачи ЛП, проходя по ребрам от вершины к вершине, пока не достигнет вершины с оптимальным значением целевой функции. Принципиальное отличие алгоритма AlEM от симплекс-метода заключается в следующем. Пусть имеется задача ЛП с системой ограничений $A\boldsymbol{x} \leqslant \boldsymbol{b}$ и стартовая точка $\hat{\boldsymbol{x}}$, являющаяся вершиной многогранника допустимых решений $M = \{\, \boldsymbol{x} |\, A\boldsymbol{x} \leqslant \boldsymbol{b}\}$ в пространстве $\mathbb{R}^n$. Симплекс-метод выделяет из исходной системы базисную подсистему $\hat{A}\boldsymbol{x} \leqslant \hat{\boldsymbol{b}}$ такую, что $\hat{A}\hat{\boldsymbol{x}} = \hat{\boldsymbol{b}}$ и $\hat{A}$ — невырожденная матрица. Далее происходит замена одной базисной строки на другую так, чтобы при переходе к следующей вершине значение целевой функции увеличилось. При этом симплекс-метод использует только один вариант построения базиса из множества возможных, общее число которых вычисляется по формуле

$$C_m^n = \frac{m!}{n!(m-n)!},$$

где m обозначает количество уравнений в системе $A\boldsymbol{x} = \boldsymbol{b}$, для которых текущая вершина является решением. В отличие от симплекс-метода, алгоритм AlEM исследует все ребра, исходящие из текущей вершины. Это полностью исключает возможность зацикливания алгоритма AlEM на вырожденных задачах. Кроме того, алгоритм AlEM позволяет построить субоптимальный путь[1] к решению задачи ЛП. В частности, для куба Кле–Минти [14] размерности n алгоритм AlEM находит решение за одну итерацию, в то время как симплекс-методу для решения этой задачи требуется $n!$ итераций. В дополнение к вышесказанному, алгоритм AlEM допускает параллельную реализацию, эффективно масштабируемую на сотнях процессорных узлов кластерной вычислительной системы. Отметим, что AlEM неприменим к оптимизационным задачам, область допустимых решений которых представляет собой невыпуклый многогранник, так как при поиске максимума целевой функции AlEM может двигаться по ребрам только в сторону увеличения ее значений. При попадании в локальный максимум алгоритм не сможет оттуда выбраться.

Статья организована следующим образом. В разделе 2 представлены определения и утверждения, используемые при описании алгоритма AlEM. Раздел 3 содержит формализованное описание алгоритма AlEM. Раздел 4 посвящен 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 на реальных и модельных задачах ЛП. В разделе 6 суммируются полученные результаты и приводятся направления дальнейших исследований. Сводка обозначений, используемых в статье, приведена в разделе 7.

2. Основные понятия. Данный раздел содержит основные понятия, определения и утверждения, необходимые для формализованного описания алгоритма AlEM.

В евклидовом пространстве $\mathbb{R}^n$ рассматривается задача ЛП общего вида, содержащая m неравенств и k уравнений:

$$\begin{aligned}
&\langle \boldsymbol{a}_1, \boldsymbol{x}\rangle \leqslant b_1,\\
&\ldots\ldots\ldots\\
&\langle \boldsymbol{a}_m, \boldsymbol{x}\rangle \leqslant b_m,\\
&\langle \boldsymbol{a}_{m+1}, \boldsymbol{x}\rangle = b_{m+1},\\
&\ldots\ldots\ldots\\
&\langle \boldsymbol{a}_{m+k}, \boldsymbol{x}\rangle = b_{m+k}.
\end{aligned} \tag{1}$$

Здесь и далее $\langle *, *\rangle$ обозначает скалярное произведение двух векторов. Предполагается, что система (1)

[1]Путь является субоптимальным, если из всех возможных вершин в качестве следующей выбирается вершина с оптимальным значением целевой функции.

2026, **27** (1), 1–18. doi 10.26089/NumMet.v27r101

включает в себя также неравенства вида

$$-x_1 \leqslant 0,$$
$$\ldots\ldots\ldots$$
$$-x_n \leqslant 0.$$

Необходимо найти точку в области допустимых решений системы (1), в которой достигается максимум линейной целевой функции

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

Указанную систему ограничений можно представить в матричном виде

$$\hat{A}\boldsymbol{x} \leqslant \hat{\boldsymbol{b}},$$
$$\bar{A}\boldsymbol{x} = \bar{\boldsymbol{b}},$$

где $\hat{A} \in \mathbb{R}^{m\times n}$, $\bar{A} \in \mathbb{R}^{k\times n}$, $\hat{\boldsymbol{b}} \in \mathbb{R}^m$, $\bar{\boldsymbol{b}} \in \mathbb{R}^k$. Предполагается, что размерность пространства $n > 1$, количество уравнений $k \geqslant 0$. Без ограничения общности мы можем полагать, что матрица $\bar{A}$ имеет полный ранг:

$$\operatorname{rank}\left(\bar{A}\right) = k. \tag{2}$$

Отсюда следует, что

$$k < n,$$

так как в противном случае область допустимых решений вырождается в точку.

Обозначим через $\hat{I}$ множество индексов строк матрицы $\hat{A}$, и через $\bar{I}$ — множество индексов строк матрицы $\bar{A}$:

$$\hat{I} = \{1, \ldots, m\},$$
$$\bar{I} = \{m+1, \ldots, m+k\}.$$

Подсистема

$$\hat{A}\boldsymbol{x} \leqslant \hat{\boldsymbol{b}}$$

задает m замкнутых полупространств[2]

$$P_1 = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_1, \boldsymbol{x} \rangle \leqslant b_1\},$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \tag{3}$$
$$P_m = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_m, \boldsymbol{x} \rangle \leqslant b_m\}.$$

Здесь $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$ обозначают строки матрицы $\hat{A}$, $b_1, \ldots, b_m$ — элементы столбца $\hat{\boldsymbol{b}}$. Пересечение полупространств (3) образует замкнутый выпуклый многогранник $\hat{M}$, называемый ограничивающим:

$$\hat{M} = \bigcap_{i\in\hat{I}} P_i.$$

Полупространства (3) ограничиваются гиперплоскостями, называемыми граничными:

$$H_1 = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_1, \boldsymbol{x} \rangle = b_1\},$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$H_m = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_m, \boldsymbol{x} \rangle = b_m\}.$$

Соответствующие уравнения

$$\langle \boldsymbol{a}_1, \boldsymbol{x} \rangle = b_1,$$
$$\ldots\ldots\ldots\ldots \tag{4}$$
$$\langle \boldsymbol{a}_m, \boldsymbol{x} \rangle = b_m$$

также будем называть граничными.

[2]Здесь и далее мы опускаем прилагательное “аффинных” в отношении полупространств, гиперплоскостей и подпространств.

2026, **27** (1), 1–18. doi 10.26089/NumMet.v27r101

Подсистема

$$\bar{A}\boldsymbol{x} = \bar{\boldsymbol{b}}$$

задает k гиперплоскостей, называемых опорными:

$$H_{m+1} = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_{m+1}, \boldsymbol{x}\rangle = b_{m+1}\},$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$H_{m+k} = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_{m+k}, \boldsymbol{x}\rangle = b_{m+k}\}.$$

Соответствующие уравнения

$$\begin{aligned} &\langle \boldsymbol{a}_{m+1}, \boldsymbol{x}\rangle = b_{m+1},\\ &\ldots\ldots\ldots\ldots\ldots\\ &\langle \boldsymbol{a}_{m+k}, \boldsymbol{x}\rangle = b_{m+k} \end{aligned} \tag{5}$$

также будем называть опорными. Здесь $\boldsymbol{a}_{m+1}, \ldots, \boldsymbol{a}_{m+k}$ обозначают строки матрицы $\bar{A}$, $b_{m+1}, \ldots, b_{m+k}$ — элементы столбца $\bar{\boldsymbol{b}}$. Пересечение опорных гиперплоскостей образует подпространство, называемое опорным:

$$\bar{S} = \begin{cases} \bigcap\limits_{i\in\bar{I}} H_{m+i}, & \text{если} \quad \bar{I} \neq \emptyset,\\ \mathbb{R}^n, & \text{если} \quad \bar{I} = \emptyset. \end{cases}$$

Область допустимых решений системы (1) является пересечением ограничивающего многогранника $\hat{M}$ с опорным подпространством $\bar{S}$ и представляет собой замкнутый выпуклый многогранник M, называемый допустимым:

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

В частности,

$$M \subset \bar{S}.$$

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

Следующее утверждение будет использовано алгоритмом AlEM для прохода по ребру от одной вершины к другой.

Утверждение 1. *Пусть заданы точка $\boldsymbol{v} \in M$, произвольный вектор $\boldsymbol{d} \in \mathbb{R}^n$ и полупространство*

$$P_i = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_i, \boldsymbol{x}\rangle \leqslant b_i\},$$

ограниченное гиперплоскостью

$$H_i = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_i, \boldsymbol{x}\rangle = b_i\},$$

где $i \in \hat{I}$. Определим луч, исходящий из точки $\boldsymbol{v}$ в направлении вектора $\boldsymbol{d}$:

$$X = \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{x} = \boldsymbol{v} + \lambda\boldsymbol{d}, \lambda \geqslant 0\}.$$

Тогда

$$\text{если} \quad \langle \boldsymbol{a}_i, \boldsymbol{d}\rangle \leqslant 0, \quad \text{то} \quad X \subset P_i, \tag{6}$$

$$\text{если} \quad \langle \boldsymbol{a}_i, \boldsymbol{d}\rangle > 0, \quad \text{то} \quad X \cap H_i = \boldsymbol{v} + \frac{b_i - \langle \boldsymbol{a}_i, \boldsymbol{v}\rangle}{\langle \boldsymbol{a}_i, \boldsymbol{d}\rangle}\boldsymbol{d}. \tag{7}$$

Другими словами, луч, исходящий из принадлежащей допустимому многограннику M точки $\boldsymbol{v}$ в направлении $\boldsymbol{d}$, пересечет гиперплоскость H_i в том и только в том случае, когда $\langle \boldsymbol{a}_i, \boldsymbol{d}\rangle > 0$. При этом точка пересечения может быть вычислена по формуле (7).

Доказательство. Так как $\boldsymbol{v} \in M$, то $\boldsymbol{v} \in P_i$, т.е.

$$\langle \boldsymbol{a}_i, \boldsymbol{v}\rangle \leqslant b_i. \tag{8}$$

Пусть $\boldsymbol{y}$ — произвольная точка луча X:

$$\boldsymbol{y} = \boldsymbol{v} + \lambda\boldsymbol{d}.$$

2026, **27** (1), 1–18. doi 10.26089/NumMet.v27r101

Тогда

$$\langle \boldsymbol{a}_i, \boldsymbol{y} \rangle = \langle \boldsymbol{a}_i, \boldsymbol{v} \rangle + \lambda \langle \boldsymbol{a}_i, \boldsymbol{d} \rangle . \tag{9}$$

Предположим, что

$$\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle \leqslant 0. \tag{10}$$

Так как $\lambda \geqslant 0$, то из (8), (9) и (10) следует

$$\langle \boldsymbol{a}_i, \boldsymbol{y} \rangle \leqslant b_i,$$

т.е. $X \subset P_i$. Таким образом, (6) имеет место.

Пусть теперь

$$\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle > 0. \tag{11}$$

Положим

$$\lambda' = \frac{b_i - \langle \boldsymbol{a}_i, \boldsymbol{v} \rangle}{\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle} + 1.$$

Из (8) и (11) следует, что $\lambda' > 0$, т.е.

$$\boldsymbol{v} + \lambda' \boldsymbol{d} \in X.$$

Имеем:

$$\langle \boldsymbol{a}_i, \boldsymbol{v} + \lambda' \boldsymbol{d} \rangle = \left\langle \boldsymbol{a}_i, \boldsymbol{v} + \left(\frac{b_i - \langle \boldsymbol{a}_i, \boldsymbol{v} \rangle}{\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle} + 1 \right) \boldsymbol{d} \right\rangle = \left\langle \boldsymbol{a}_i, \boldsymbol{v} + \frac{b_i - \langle \boldsymbol{a}_i, \boldsymbol{v} \rangle}{\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle} \boldsymbol{d} + \boldsymbol{d} \right\rangle = b_i + \langle \boldsymbol{a}_i, \boldsymbol{d} \rangle.$$

Сопоставляя это с (11), получаем:

$$\langle \boldsymbol{a}_i, \boldsymbol{v} + \lambda' \boldsymbol{d} \rangle > b_i,$$

т.е. принадлежащая лучу X точка $\boldsymbol{v} + \lambda' \boldsymbol{d}$ не принадлежит полупространству P_i. Учитывая, что начальная точка луча принадлежит полупространству P_i, можно сделать вывод, что луч X пересекает граничную гиперплоскость H_i в единственной точке. В силу (8) и (11) точка $\boldsymbol{v} + \dfrac{b_i - \langle \boldsymbol{a}_i, \boldsymbol{v} \rangle}{\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle} \boldsymbol{d}$ принадлежит лучу X. Непосредственно проверяется, что эта точка также удовлетворяет уравнению

$$\left\langle \boldsymbol{a}_i, \boldsymbol{v} + \frac{b_i - \langle \boldsymbol{a}_i, \boldsymbol{v} \rangle}{\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle} \boldsymbol{d} \right\rangle = b_i,$$

т.е. принадлежит граничной гиперплоскости H_i. Утверждение 1 доказано.

Обозначим

$$A = \begin{bmatrix} \hat{A} \\ \bar{A} \end{bmatrix}.$$

Пусть $J \subseteq \hat{I} \cup \bar{I}$ и пусть A_J — матрица, включающая в себя строки матрицы A, индексы которых присутствуют в J.

Следующее утверждение позволяет проверить, что точка является вершиной допустимого многогранника.

Утверждение 2. *Пусть $\boldsymbol{v} \in M$, $\hat{\mathcal{I}}(\boldsymbol{v})$ — множество индексов всех граничных уравнений, которым удовлетворяет точка $\boldsymbol{v}$:*

$$\hat{\mathcal{I}}(\boldsymbol{v}) = \left\{ i \in \hat{I} \,\middle|\, \langle \boldsymbol{a}_i, \boldsymbol{v} \rangle = b_i \right\}.$$

Следующие условия являются равносильными:

(i) *$\boldsymbol{v}$ является вершиной допустимого многогранника M,*

(ii) $\operatorname{rank}\left(A_{\hat{\mathcal{I}}(\boldsymbol{v}) \cup \bar{I}} \right) = n.$

Доказательство. Сначала покажем, что из (i) следует (ii). Пусть $\boldsymbol{v}$ является вершиной допустимого многогранника M. С учетом (2) это означает, что точка $\boldsymbol{v}$ является единственным решением системы линейных алгебраических уравнений ранга n, в которую входят k опорных уравнений (5) и $n-k$ из числа граничных уравнений (4). Таким образом, имеет место следующее утверждение:

$$\exists \hat{J} \subseteq \hat{\mathcal{I}}(\boldsymbol{v}): \quad \left| \hat{J} \right| = n - k \ \wedge \ \operatorname{rank}\left(A_{\hat{J} \cup \bar{I}} \right) = n.$$

2026, **27** (1), 1–18. doi 10.26089/NumMet.v27r101

Следовательно,

$$\operatorname{rank}\left(A_{\hat{\mathcal{I}}(\boldsymbol{v})\cup\bar{I}}\right) \geqslant \operatorname{rank}\left(A_{\hat{\jmath}\cup\bar{I}}\right) = n.$$

Учитывая, что ранг матрицы $A_{\hat{\mathcal{I}}(\boldsymbol{v})\cup\bar{I}}$ не может быть больше n, так как в этой матрице n столбцов, получаем

$$\operatorname{rank}\left(A_{\hat{\mathcal{I}}(\boldsymbol{v})\cup\bar{I}}\right) = n,$$

т.е. условие (ii) выполняется.

Теперь покажем, что из (ii) следует (i). Пусть выполняется условие

$$\operatorname{rank}\left(A_{\hat{\mathcal{I}}(\boldsymbol{v})\cup\bar{I}}\right) = n.$$

Рассмотрим систему уравнений

$$A_{\hat{\mathcal{I}}(\boldsymbol{v})\cup\bar{I}}\boldsymbol{x} = \boldsymbol{b}_{\hat{\mathcal{I}}(\boldsymbol{v})\cup\bar{I}}. \tag{12}$$

Если количество уравнений в этой системе равно n, то точка $\boldsymbol{v}$ будет единственным решением. Это означает, что точка $\boldsymbol{v}$ является вершиной допустимого многогранника M. Предположим, что количество уравнений в системе (12) больше, чем n. В этом случае последовательно удалим из системы (12) все лишние уравнения, соответствующие граничным гиперплоскостям, так, чтобы ранг системы не изменился. В результате мы получим эквивалентную систему с квадратной матрицей ранга n. Точка $\boldsymbol{v}$ будет единственным решением этой системы, т.е. $\boldsymbol{v}$ будет вершиной допустимого многогранника M. Таким образом, условие (i) выполняется во всех случаях. Утверждение 2 доказано.

Следующее определение помогает идентифицировать ребра допустимого многогранника, выходящие из некоторой вершины.

Определение 1. *Вершинной прямой по отношению к вершине $\boldsymbol{v} \in M$ будем называть прямую $L^{\boldsymbol{v}}_{\hat{\jmath}}$, задаваемую формулой*

$$L^{\boldsymbol{v}}_{\hat{\jmath}} = \left\{ \boldsymbol{x} \in \bigcap_{i\in\hat{\jmath}\cup\bar{I}} H_i \;\middle|\; \hat{\jmath} \subseteq \hat{\mathcal{I}}(\boldsymbol{v}),\ \left|\hat{\jmath}\right| = n-k-1,\ \operatorname{rank}\left(A_{\hat{\jmath}}\right) = n-k-1,\ \operatorname{rank}\left(A_{\hat{\jmath}\cup\bar{I}}\right) = n-1 \right\}. \tag{13}$$

Любая вершинная прямая $L^{\boldsymbol{v}}_{\hat{\jmath}}$ проходит через вершину $\boldsymbol{v}$. Через $\boldsymbol{v}$ может проходить несколько различных вершинных прямых. Для каждого ребра D, исходящего из вершины $\boldsymbol{v}$, существует вершинная прямая $L^{\boldsymbol{v}}_{\hat{\jmath}}$ такая, что $D \subset L^{\boldsymbol{v}}_{\hat{\jmath}}$.

Вершинную прямую $L^{\boldsymbol{v}}_{\hat{\jmath}}$ можно задать в параметрическом виде следующим образом:

$$L^{\boldsymbol{v}}_{\hat{\jmath}} = \left\{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{x} = \boldsymbol{v} + \lambda\boldsymbol{d},\ \lambda \in \mathbb{R}\right\},$$

где $\boldsymbol{d} \in \mathbb{R}^n$ — направляющий вектор, вычисляемый по формуле

$$\boldsymbol{d} = \boldsymbol{w} - \boldsymbol{v}, \tag{14}$$

где

$$\boldsymbol{w} = \boldsymbol{\pi}_{\hat{\jmath}\cup\bar{I}}(\boldsymbol{v} + \boldsymbol{g}). \tag{15}$$

Здесь $\boldsymbol{\pi}_{\hat{\jmath}\cup\bar{I}}(*)$ обозначает ортогональную проекцию на подпространство $\bigcap\limits_{i\in\hat{\jmath}\cup\bar{I}} H_i$, образующее вершинную прямую $L^{\boldsymbol{v}}_{\hat{\jmath}}$ в соответствии с формулой (13). В качестве $\boldsymbol{g}$ может быть выбран произвольный ненулевой вектор, не являющийся ортогональным по отношению к прямой $L^{\boldsymbol{v}}_{\hat{\jmath}}$. В этом случае $\boldsymbol{w} \neq \boldsymbol{v}$. Ортогональная проекция $\boldsymbol{\pi}_{\hat{\jmath}\cup\bar{I}}(*)$ может быть вычислена по следующей формуле [15]:

$$\boldsymbol{\pi}_{\hat{\jmath}\cup\bar{I}}(\boldsymbol{x}) = \boldsymbol{x} - A^T_{\hat{\jmath}\cup\bar{I}}\left(A_{\hat{\jmath}\cup\bar{I}}A^T_{\hat{\jmath}\cup\bar{I}}\right)^{-1}\left(A_{\hat{\jmath}\cup\bar{I}}\boldsymbol{x} - \boldsymbol{b}_{\hat{\jmath}\cup\bar{I}}\right). \tag{16}$$

Заметим, что в соответствии с (2) и (13) матрица $A_{\hat{\jmath}\cup\bar{I}}$ имеет полностроочный ранг. Поэтому матрица $A_{\hat{\jmath}\cup\bar{I}}A^T_{\hat{\jmath}\cup\bar{I}}$ будет обратимой (см. утверждение 3F в [16]).

3. Алгоритм AlEM. Данный раздел содержит формализованное описание алгоритма AlEM, строящего из произвольной вершины допустимого многогранника субоптимальный путь вдоль его ребер к вершине, являющейся решением задачи ЛП. Субоптимальность пути заключается в том, что всегда выбирается ребро, имеющее наибольшее значение целевой функции в своей конечной точке. Алгоритм AlEM

2026, **27** (1), 1–18. doi 10.26089/NumMet.v27r101

требует, чтобы начальная точка $\boldsymbol{v} \in M$ являлась вершиной. В соответствии с утверждением 2 для этого необходимо проверить условие $\operatorname{rank}(A_{\hat{\mathcal{I}}(\boldsymbol{v})\cup\bar{I}}) = n$. Начальная вершина может быть получена, например, с помощью метода исключения переменных Фурье–Моцкина [17]. Если известна некоторая допустимая точка $\boldsymbol{u} \in M$, то вершина многогранника M может быть найдена с помощью алгоритма VeSP [18] не более чем за n итераций. Псевдокод алгоритма AlEM представлен в виде алгоритма 1.

Алгоритм 1. AlEM
Algorithm 1. AlEM

```
1: input A, b, c, v
2:     v_0 := v
3:     t := 0
4:     repeat
5:         v_{t+1} := Traverse(v_t)
6:         t := t + 1
7:     until v_t = v_{t-1}
8:     output v_t
9: stop
```

Прокомментируем шаги алгоритма 1. На шаге 1 вводятся исходные данные задачи ЛП и некоторая вершина $\boldsymbol{v}$ допустимого многогранника M, которая на шаге 2 используется в качестве начальной вершины $\boldsymbol{v}_0$. На шаге 3 счетчику итераций t присваивается значение 0. Цикл **repeat**/**until** с помощью функции Traverse, представленной ниже в виде алгоритма 2, строит субоптимальный путь, проходя от вершины к вершине по ребрам допустимого многогранника M. При этом функция Traverse в каждой вершине исследует все исходящие ребра и выбирает ребро, ведущее к вершине с наибольшим значением целевой функции. В результате получается последовательность вершин

$$\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_t, \boldsymbol{v}_{t+1}, \ldots,$$

такая, что значение целевой функции в каждой следующей вершине больше, чем в предыдущей:

$$\langle \boldsymbol{c}, \boldsymbol{v}_0\rangle < \langle \boldsymbol{c}, \boldsymbol{v}_1\rangle < \ldots < \langle \boldsymbol{c}, \boldsymbol{v}_t\rangle < \langle \boldsymbol{c}, \boldsymbol{v}_{t+1}\rangle < \ldots\ .$$

Поскольку количество ограничений в системе (1) конечно, количество вершин допустимого многогранника M также конечно. Поэтому алгоритм AlEM за конечное число итераций придет к вершине, у которой нет исходящих ребер, ведущих к вершинам с бóльшим значением целевой функции. В этом случае функция Traverse в качестве результата возвращает текущую вершину $\boldsymbol{v}$, которая и будет решением задачи ЛП.

Функция Traverse($\boldsymbol{v}$) выполняет переход из текущей вершины $\boldsymbol{v}$ в смежную вершину $\boldsymbol{v}_{\max}$, имеющую максимальное значение целевой функции среди всех смежных вершин. Для этого перебираются все вершинные прямые, проходящие через $\boldsymbol{v}$. Для каждой вершинной прямой вычисляется направляющий вектор, ориентированный таким образом, чтобы значение целевой функции возрастало (вершинные прямые, перпендикулярные градиенту целевой функции, отбрасываются). С помощью направляющих векторов вычисляются все вершины, смежные с $\boldsymbol{v}$, у которых значение целевой функции большее, чем у исходной вершины. Если такие вершины отсутствуют, функция Traverse возвращает в качестве результата текущую вершину $\boldsymbol{v}$. Если множество смежных вершин с бóльшим значением целевой функции не пусто, то Traverse возвращает в качестве результата вершину с максимальным значением целевой функции. Псевдокод функции Traverse представлен в виде алгоритма 2.

Прокомментируем шаги алгоритма 2. На шаге 2 координатам вершины $\boldsymbol{v}_{\max}$ присваиваются координаты исходной вершины $\boldsymbol{v}$. На шаге 3 переменной $F_{\max}$ присваивается значение целевой функции в исходной вершине. На шагах 4–9 вычисляется множество $\hat{I}_{\boldsymbol{v}}$, включающее в себя индексы всех граничных гиперплоскостей, проходящих через точку $\boldsymbol{v}$. На шагах 10–11 формируется первая комбинация $\hat{J}$, включающая в себя $n-k-1$ индексов граничных гиперплоскостей из множества $\hat{I}_{\boldsymbol{v}}$. Для этого используется функция Twiddle, основанная на одноименной процедуре [19]. Функция Twiddle $(X,\ l,\ combNo)$, где $combNo \in \left\{1, \ldots, C^l_{|X|}\right\}$, строит все сочетания из l элементов множества X без повторений в определенном порядке. При каждом вызове Twiddle$(\hat{I}_{\boldsymbol{v}}, n-k-1, combNo)$ возвращает комбинацию с порядковым номером $combNo$. Если значение $combNo$ превышает количество сочетаний, то Twiddle возвращает пустое множество. Цикл **while** (шаги 12–32) исследует все вершинные прямые, проходящие через $\boldsymbol{v}$. На шаге 13 проверяется последнее условие из формулы (13). На шагах 14–22 вычисляется направляющий вектор $\boldsymbol{d}$ в соответствии с формулой (14). Цикл **repeat**/**until** (шаги 14–17) вычисляет по формуле (15) точку $\boldsymbol{w}$, которая лежит на исследуемой вершинной прямой и не совпадает с текущей вершиной $\boldsymbol{v}$. Шаги 18–22 ориентируют $\boldsymbol{d}$ в направлении возрастания целевой функции. На шаге 23 функция LambdaM$(\boldsymbol{v}, \boldsymbol{d})$ (см. ниже алгоритм 3) вычисляет максимальное неотрицательное число λ_M такое, что $\boldsymbol{v} + \lambda_M \boldsymbol{d} \in M$. Заметим, что при $\lambda_M = 0$ вершинная прямая задает ложное ребро, имеющее нулевую длину. На шаге 24

2026, **27** (1), 1–18. doi 10.26089/NumMet.v27r101

Алгоритм 2. Функция Traverse (переход по ребру к следующей вершине)

Algorithm 2. Function Traverse (traverse edge to next vertex)

function Traverse($\boldsymbol{v}$)
  $\boldsymbol{v}_{\max} := \boldsymbol{v}$
  $F_{\max} := \langle \boldsymbol{c}, \boldsymbol{v} \rangle$
  $\hat{I}_{\boldsymbol{v}} := \emptyset$
  for $i \in \hat{I}$ **do**
    if $\langle \boldsymbol{a}_i, \boldsymbol{v} \rangle = b_i$ **then**
      $\hat{I}_{\boldsymbol{v}} := \hat{I}_{\boldsymbol{v}} \cup \{i\}$
    end if
  end for
  $combNo := 1$
  $\hat{J} := \text{Twiddle}(\hat{I}_{\boldsymbol{v}}, n-k-1, combNo)$
  while $\hat{J} \neq \emptyset$
    if $\text{rank}(A_{\hat{J} \cup \bar{I}}) = n-1$ **then**
      repeat
        $\boldsymbol{g} := \text{RandomNonZeroVector}()$
        $\boldsymbol{w} := \boldsymbol{\pi}_{\hat{J} \cup \bar{I}}(\boldsymbol{v} + \boldsymbol{g})$
      until $\boldsymbol{v} \neq \boldsymbol{w}$
      if $\langle \boldsymbol{c}, \boldsymbol{w} \rangle > \langle \boldsymbol{c}, \boldsymbol{v} \rangle$ **then**
        $\boldsymbol{d} := \boldsymbol{w} - \boldsymbol{v}$
      else
        $\boldsymbol{d} := \boldsymbol{v} - \boldsymbol{w}$
      end if
      $\lambda_M := \text{LambdaM}(\boldsymbol{v}, \boldsymbol{d})$
      $\boldsymbol{v}_{\text{next}} := \boldsymbol{v} + \lambda_M \boldsymbol{d}$
      if $\langle \boldsymbol{c}, \boldsymbol{v}_{\text{next}} \rangle > F_{\max}$ **then**
        $\boldsymbol{v}_{\max} := \boldsymbol{v}_{\text{next}}$
        $F_{\max} := \langle \boldsymbol{c}, \boldsymbol{v}_{\max} \rangle$
      end if
    end if
    $combNo := combNo + 1$
    $\hat{J} := \text{Twiddle}(\hat{I}_{\boldsymbol{v}}, n-k-1, combNo)$
  end while
  return $\boldsymbol{v}_{\max}$
end function

Алгоритм 3. Функция LambdaM (вычисление λ_M)

Algorithm 3. Function LambdaM (calculating λ_M)

function LambdaM($\boldsymbol{v}, \boldsymbol{d}$)
  for $i \in \hat{I}$ **do**
    if $\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle > 0 \ \wedge \ \langle \boldsymbol{a}_i, \boldsymbol{v} \rangle = b_i$ **then**
      return 0
    end if
  end for
  $\lambda_M := +\infty$
  for $i \in \hat{I}$ **do**
    if $\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle > 0$ **then**
      $\lambda_i := (b_i - \langle \boldsymbol{a}_i, \boldsymbol{v} \rangle) / \langle \boldsymbol{a}_i, \boldsymbol{d} \rangle$
      if $\lambda_i < \lambda_M$ **then**
        $\lambda_M := \lambda_i$
      end if
    end if
  end for
  return λ_M
end function

2026, **27** (1), 1–18. doi 10.26089/NumMet.v27r101

вычисляется смежная вершина $\boldsymbol{v}_{\text{next}}$, соответствующая найденному ребру (для ложного ребра получим $\boldsymbol{v}_{\text{next}} = \boldsymbol{v}$). Если значение целевой функции в вершине $\boldsymbol{v}_{\text{next}}$ больше $F_{\max}$, то обновляются значения $\boldsymbol{v}_{\max}$ и $F_{\max}$ (шаги 25–28). Шаги 30–31 вычисляют следующую комбинацию индексов $\hat{J}$. Шаг 33 в качестве результата возвращает вершину $\boldsymbol{v}_{\max}$, в которой достигается наибольшее значение целевой функции среди всех рассмотренных.

Алгоритм 2 использует функцию LambdaM, псевдокод которой представлен в виде алгоритма 3. Параметр $\boldsymbol{v}$ должен быть точкой допустимого многогранника M. Параметр $\boldsymbol{d}$ должен быть вектором, действующим в подпространстве, получаемом в результате пересечения всех опорных гиперплоскостей. Функция LambdaM$(\boldsymbol{v}, \boldsymbol{d})$ вычисляет максимальное неотрицательное число λ_M такое, что $\boldsymbol{v} + \lambda_M \boldsymbol{d} \in M$.

Прокомментируем шаги этого алгоритма. Цикл **for** на шагах 2–6 проверяет все граничные гиперплоскости. Если среди них имеется граничная гиперплоскость H_i, проходящая через точку $\boldsymbol{v}$, такая, что $\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle > 0$, то в соответствии с (7) это означает, что единственной общей точкой допустимого многогранника M и луча

$$X = \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{x} = \boldsymbol{v} + \lambda \boldsymbol{d}, \lambda \geqslant 0\}$$

является точка $\boldsymbol{v}$. В этом случае $\lambda_M = 0$. Шаг 7 присваивает переменной λ_M в качестве начального значения бесконечно большое положительное число. Цикл **for** на шагах 8–15 для всех граничных гиперплоскостей H_i, удовлетворяющих условию $\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle > 0$, вычисляет λ_i в соответствии с формулой (7). Среди них выбирается минимальное значение:

$$\lambda_M = \min \left\{ \lambda_i \middle| i \in \hat{I},\ \langle \boldsymbol{a}_i, \boldsymbol{d} \rangle > 0 \right\}.$$

Поскольку по предположению допустимый многогранник M является ограниченным множеством, этот минимум существует, причем $\lambda_M > 0$. В силу утверждения 1 значение λ_M будет наибольшим числом, для которого $\boldsymbol{v} + \lambda_M \boldsymbol{d} \in M$.

4. Параллельная версия алгоритма AlEM. Алгоритм 2 в цикле **while** (шаги 12–32) перебирает все возможные подмножества $\hat{J}$ из $n - k - 1$ элементов множества $\hat{I}_{\boldsymbol{v}}$. Обозначим $m_{\boldsymbol{v}} = |\hat{I}_{\boldsymbol{v}}|$. Тогда количество таких подмножеств вычисляется по формуле

$$C_{m_{\boldsymbol{v}}}^{n-k-1} = \frac{m_{\boldsymbol{v}}!}{(n-k-1)!\,(m_{\boldsymbol{v}} - n + k + 1)!}. \tag{17}$$

В простейшем случае, когда $m_{\boldsymbol{v}} = n - k$, формула (17) дает $n - k$ комбинаций. Однако на практике в большинстве случаев встречаются задачи ЛП, для которых $m_{\boldsymbol{v}} > n - k$. При решении таких задач с помощью алгоритма AlEM возникает комбинаторный перебор, который может потребовать использования суперкомпьютерных мощностей. Поэтому мы разработали параллельную версию алгоритма AlEM для кластерных вычислительных систем. Работа параллельных процессов организуется по схеме SIMD (single instruction, multiple data): все параллельные процессы выполняют один и тот же код, но над различными данными. Сочетания, конструируемые с помощью функции Twiddle, распределяются между параллельными процессами по принципу round-robin. Для этого мы преобразовали функцию Traverse (алгоритм 2) в функцию ParallelTraverse, псевдокод которой представлен в виде алгоритма 4.

В отличие от последовательной функции Traverse, функция ParallelTraverse начинает свою работу с комбинации, номер которой равен номеру текущего параллельного процесса (шаги 10–11). Здесь SYS_MyProcessNo обозначает системную функцию, возвращающую номер текущего параллельного процесса. При переходе к следующей итерации цикла **while** (шаги 13–33) номер комбинации увеличивается на число параллельных процессов (шаг 31). Здесь SYS_NumberOfProcesses обозначает системную функцию, возвращающую число параллельных процессов, задействованных в вычислениях. Шаг 34 с помощью системной функции SYS_AllReduceMaxLoc вычисляет $processesNo_{\max}$ — номер параллельного процесса, на котором получено максимальное значение целевой функции в точке $\boldsymbol{v}_{\max}$. Шаг 35 с помощью системной функции SYS_Broadcast пересылает координаты точки $\boldsymbol{v}_{\max}$ из параллельного процесса с номером $processesNo_{\max}$ всем остальным параллельным процессам. В результате функция ParallelTraverse возвращает координаты одной и той же новой вершины для всех параллельных процессов.

Соответствующая параллельная реализация алгоритма AlEM представлена в виде алгоритма 5. Кроме использования функции ParallelTraverse вместо функции Traverse, параллельная реализация имеет только одно отличие от последовательной реализации 1. Это отличие заключается в том, что добавлено условие, в соответствии с которым решение задачи ЛП будет выводить параллельный процесс с номером 0 (шаги 8–10).

2026, **27** (1), 1–18. doi 10.26089/NumMet.v27r101

Алгоритм 4. Функция ParallelTraverse (параллельное вычисление следующей вершины)

Algorithm 4. Function ParallelTraverse (parallel calculation of the next vertex)

function ParallelTraverse($\boldsymbol{v}$)
  $\boldsymbol{v}_{\max} := \boldsymbol{v}$
  $F_{\max} := \langle \boldsymbol{c}, \boldsymbol{v} \rangle$
  $\hat{I}_{\boldsymbol{v}} := \emptyset$
  for $i \in \hat{I}$ **do**
    if $\langle \boldsymbol{a}_i, \boldsymbol{v} \rangle = b_i$ **then**
      $\hat{I}_{\boldsymbol{v}} := \hat{I}_{\boldsymbol{v}} \cup \{i\}$
    end if
  end for
  myNodeNo := SYS_MyProcessNo()
  combNo := *myNodeNo*
  $\hat{J} :=$ Twiddle$(\hat{I}_{\boldsymbol{v}}, n-k-1,$ *combNo*$)$
  while $\hat{J} \neq \emptyset$
    if rank$(A_{\hat{J} \cup \bar{I}}) = n-1$ **then**
      repeat
        $\boldsymbol{g} :=$ RandomNonZeroVector()
        $\boldsymbol{w} := \boldsymbol{\pi}_{\hat{J} \cup \bar{I}}(\boldsymbol{v} + \boldsymbol{g})$
      until $\boldsymbol{v} \neq \boldsymbol{w}$
      if $\langle \boldsymbol{c}, \boldsymbol{w} \rangle > \langle \boldsymbol{c}, \boldsymbol{v} \rangle$ **then**
        $\boldsymbol{d} := \boldsymbol{w} - \boldsymbol{v}$
      else
        $\boldsymbol{d} := \boldsymbol{v} - \boldsymbol{w}$
      end if
      $\lambda_M :=$ LambdaM$(\boldsymbol{v}, \boldsymbol{d})$
      $\boldsymbol{v}_{\text{next}} := \boldsymbol{v} + \lambda_M \boldsymbol{d}$
      if $\langle \boldsymbol{c}, \boldsymbol{v}_{\text{next}} \rangle > F_{\max}$ **then**
        $\boldsymbol{v}_{\max} := \boldsymbol{v}_{\text{next}}$
        $F_{\max} := \langle \boldsymbol{c}, \boldsymbol{v}_{\max} \rangle$
      end if
    end if
    combNo := *combNo* + SYS_NumberOfProcesses()
    $\hat{J} :=$ Twiddle$(\hat{I}_{\boldsymbol{v}}, n-k-1,$ *combNo*$)$
  end while
  $processesNo_{\max} :=$ SYS_AllReduceMaxLoc $(\langle \boldsymbol{c}, \boldsymbol{v}_{\max} \rangle)$
  SYS_Broadcast$(processesNo_{\max}, \boldsymbol{v}_{\max})$
  return $\boldsymbol{v}_{\max}$
end function

Алгоритм 5. Параллельная версия AlEM

Algorithm 5. Parallel version of AlEM

input $A, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{v}$
  $\boldsymbol{v}_0 := \boldsymbol{v}$
  $t := 0$
  repeat
    $\boldsymbol{v}_{t+1} :=$ ParallelTraverse$(\boldsymbol{v}_t)$
    $t := t + 1$
  until $\boldsymbol{v}_t = \boldsymbol{v}_{t-1}$
  if SYS_MyProcessNo() $= 0$ **then**
    output $\boldsymbol{v}_t$
  end if
stop

2026, **27** (1), 1–18. doi 10.26089/NumMet.v27r101

5. Реализация и вычислительные эксперименты. Мы реализовали параллельную версию алгоритма AlEM на языке C++. Исходные коды параллельной программы доступны в репозитории GitHub по адресу https://github.com/zhulevae/AlEM. Для реализации системных функций, обеспечивающих параллельные вычисления в алгоритмах 4 и 5, была использована библиотека параллельного программирования MPI 3.0 [20]. Для получения общего количества процессов в качестве системной функции `SYS_NumberOfProcesses` была использована функция

```
MPI_Comm_size(MPI_COMM_WORLD, &processesNumber).
```

Первым аргументом этой функции является глобальный коммуникатор, включающий в себя все использующиеся процессы, а в качестве второго передается указатель на переменную для записи результата. Для получения номера текущего параллельного процесса в качестве системной функции `SYS_MyProcessNo` была использована функция

```
MPI_Comm_rank(MPI_COMM_WORLD, &myNodeNo).
```

В качестве системной функции `SYS_AllReduceMaxLoc` была использована функция

```
MPI_Allreduce(&local, &reduced, 1, MPI_DOUBLE_INT, MPI_MAXLOC, MPI_COMM_WORLD).
```

Первый и второй аргументы являются указателями на входной буфер и буфер с агрегированными данными. Третий аргумент указывает на количество элементов входного буфера. Тип `MPI_DOUBLE_INT` позволяет помещать в буферы пары: значение целевой функции в качестве критерия сравнения и номер текущего процесса в качестве индекса. В сочетании с оператором агрегации `MPI_MAXLOC` это позволяет получить в качестве результата агрегации номер процесса с максимальным значением целевой функции и само значение целевой функции. Системная функция `SYS_Broadcast` реализована с помощью функции

```
MPI_Bcast(&v_max, v_max.size(), MPI_DOUBLE, reduced.procNo, MPI_COMM_WORLD).
```

В качестве четвертого аргумента указывается номер процесса с максимальным значением целевой функции, который осуществляет рассылку данных всем остальным процессам.

С помощью разработанной программы мы исследовали масштабируемость параллельной версии алгоритма AlEM. Для проведения вычислительных экспериментов были использованы четыре задачи ЛП из репозитория https://github.com/leonid-sokolinsky/Pset01. Характеристики этих задач приведены в табл. 1.

Здесь m обозначает количество ограничений (включая ограничения вида $\boldsymbol{x} \geqslant 0$), n — количество переменных (размерность пространства решений), $\dim(M)$ — размерность многогранника допустимых решений. Задачи afiro и israel заимствованы из репозитория Netlib-LP [21]. Точные значения максимума целевой функции для этих задач взяты из работы [22]. Задачи tcube1K и tcube2K сконструированы авторами. Область допустимых решений этих задач представляет собой гиперкуб с отсеченной вершиной. Их описание можно найти в [23]. Файлы задач afiro и israel представлены в формате MPS [24]. Файлы задач tcube1K и tcube2K используют формат MatrixMarket [25]. Координаты стартовых вершин всех задач заданы в формате MatrixMarket.

Эксперименты проводились на вычислительном кластере “Торнадо ЮУрГУ” [26], характеристики которого представлены в табл. 2.

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. Задачи

Таблица 1. Задачи ЛП из репозитория https://github.com/leonid-sokolinsky/Pset01

Table 1. LP problems from repository https://github.com/leonid-sokolinsky/Pset01

Задача Problem	m	n	$\dim(M)$	Максимум целевой функции Maximum of objective function	Формат файлов File format
afiro	27	32	24	$0.464753142857142857142857142857142857 \times 10^3$	MPS
israel	174	142	142	$0.896644821863045729662004641960 45 \times 10^6$	MPS
tcube1K	2001	1000	1000	400 199 900	MatrixMarket
tcube2K	4001	2000	2000	100 099 900	MatrixMarket

2026, **27** (1), 1–18. doi 10.26089/NumMet.v27r101

Таблица 2. Характеристики кластера “Торнадо ЮУрГУ”

Table 2. Specifications of “Tornado SUSU” cluster

Параметр Parameter	Значение Value
Количество доступных процессорных узлов Number of available processor nodes	320
Процессоры Processors	Intel Xeon X5680 (6 cores, 3.33 GHz)
Число процессоров на узел Number of processors per node	2
Память на узел Memory per node	24 GB DDR3
Соединительная сеть Interconnect	InfiniBand QDR (40 Gbit/s)
Операционная система Operating system	Linux CentOS

Таблица 3. Результаты экспериментов с параллельной версией алгоритма AlEM

Table 3. Experimental results with the parallel version of AlEM algorithm

Задача Problem	Размерность Dimension	Число сочетаний Number of combinations	$K_{\max}$	$T_{K_{\max}}$	T_{20}	$\alpha(K_{\max})$	$\epsilon(K_{\max})$	Simplex/ AlEM
1	2	3	4	5	6	7	8	9
afiro	32	475020	140	2.6	13	4.97	0.71	0.0004
israel	142	10153	60	0.9	1.6	1.85	0.62	0.02
tcube1K	1000	1000	200	110	501	4.56	0.46	0.2
tcube2K	2000	2000	200	1978	9656	4.88	0.49	0.4

запускались на различном количестве процессорных узлов кластера. При этом общее число MPI-процессов было равно $12K$, где K — количество задействованных процессорных узлов. Таким образом, на каждом узле работало 12 MPI-процессов — по числу физических ядер. Результаты экспериментов представлены в табл. 3.

Поясним семантику данных, приведенных в этой таблице. В столбце 2 приведено количество переменных в ограничениях задачи ЛП, определяющее размерность пространства решений. В столбце 3 указано число сочетаний $n-1$ элементов из общего количества опорных и граничных гиперплоскостей, проходящих через текущую вершину допустимого многогранника, наблюдаемое при прохождении субоптимального пути. В столбце 4 приводится граница масштабируемости $K_{\max}$ — максимальное количество процессорных узлов, после которого ускорение перестает расти. В столбце 5 указано время решения задачи на $K_{\max}$ узлах[3]. Столбец 6 содержит время решения задачи на 20 узлах. Это время берется за базу при вычислении ускорения и параллельной эффективности. Значения ускорения, достигнутые на границе масштабируемости, представлены в столбце 7. Ускорение вычислялось по формуле

$$\alpha(K_{\max}) = \frac{T_{20}}{T_{K_{\max}}},$$

где T_{20} — время, затраченное на решение задачи ЛП на 20 процессорных узлах, $T_{K_{\max}}$ — время, затраченное на решение этой же задачи на $K_{\max}$ процессорных узлах. В столбце 8 содержится параллельная эффективность, достигаемая на границе масштабируемости. Параллельная эффективность вычислялась по формуле

$$\epsilon(K_{\max}) = \frac{20 \cdot T_{20}}{K_{\max} \cdot T_{K_{\max}}}.$$

Столбец 9 содержит отношение времени решения задачи ЛП параллельным симплекс-алгоритмом ко

[3]Время везде указано в секундах.

2026, **27** (1), 1–18. doi 10.26089/NumMet.v27r101

времени решения этой же задачи параллельным алгоритмом AlEM. Исходные коды реализации параллельного симплекс-алгоритма, использованного для сравнения, доступны по адресу https://github.com/leonid-sokolinsky/Simplex. Симплекс-алгоритм для всех исследуемых задач запускался на двух процессорных узлах по 12 MPI-процессов на узел (использование большего количества узлов приводило к деградации ускорения).

Результаты экспериментов показывают, что граница масштабируемости параллельной версии алгоритма AlEM зависит от двух ключевых параметров — размерности решаемой задачи и числа возможных сочетаний, перебираемых при переходе от текущей вершины к следующей. Так, при решении небольшой задачи afiro размерности 32 с большим количеством перебираемых сочетаний, равным 475020, граница масштабируемости достигается на 140 процессорных узлах. Задача israel имеет размерность в четыре с лишним раза больше, однако число перебираемых сочетаний у этой задачи в 47 раз меньше. В результате, граница масштабируемости для задачи israel достигается уже на 60 процессорных узлах. Задачи большой размерности tcube1K и tcube2K характеризуются относительно малым количеством перебираемых сочетаний, равным 1000 и 2000 соответственно. Однако они имеют границу масштабируемости примерно на 200 вычислительных узлах. Следует отметить, что параллельная эффективность на границе масштабируемости во всех случаях не опускается ниже 46%, что является неплохим результатом для параллельной реализации численного метода.

Сравнение параллельной версии алгоритма AlEM с параллельной реализацией симплекс-метода на исследуемых задачах показало, что AlEM проигрывает симплекс-методу, однако при увеличении размерности задачи разница в быстродействии выравнивается. При этом необходимо учитывать, что алгоритм AlEM полностью исключает возможность зацикливания на вырожденных задачах, чего нельзя сказать о симплекс-методе. Мы проверили алгоритм AlEM и симплекс-алгоритм на вырожденных задачах hamck26e и hamck26s из работы [10]. На обеих задачах алгоритм AlEM демонстрировал отсутствие зацикливания. В то же время симплекс-алгоритм попадал в состояние зацикливания на каждой из этих задач.

Графики ускорения и параллельной эффективности на различных задачах ЛП приведены на рис. 1. Ломаный характер линий объясняется тем, что не во всех случаях число перебираемых сочетаний было кратным количеству MPI-процессов.

6. Заключение. Представлен новый масштабируемый проекционный алгоритм AlEM (Along Edges Movement) для решения задач ЛП на многопроцессорных вычислительных системах с распределенной памятью. Основные научные и практические результаты работы заключаются в следующем. Дано формализованное описание алгоритма AlEM, который, аналогично симплекс-методу, строит путь по ребрам многогранника допустимых решений от произвольной начальной вершины к вершине с оптимальным зна-

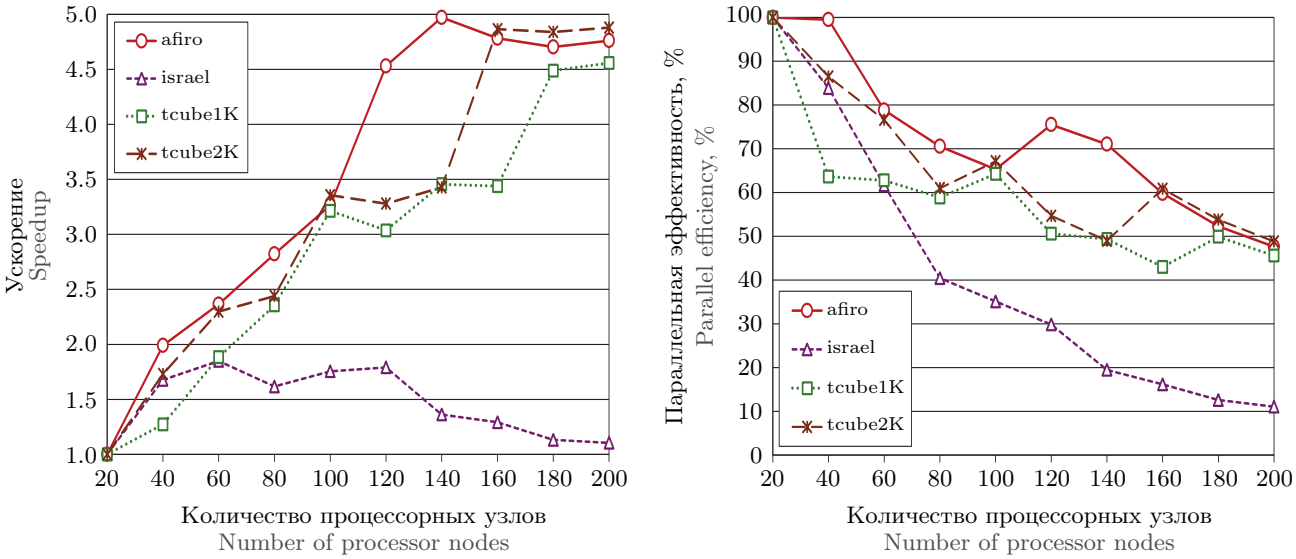

Рис. 1. Ускорение и эффективность параллельной версии алгоритма AlEM

Fig. 1. Speedup and efficiency of parallel version of AlEM algorithm

2026, **27** (1), 1–18. doi 10.26089/NumMet.v27r101

чением целевой функции. Ключевым отличием и преимуществом AlEM является исследование на каждом шаге всех исходящих из текущей вершины ребер, что гарантирует построение субоптимального пути и полностью исключает возможность зацикливания на вырожденных задачах в отличие от классического симплекс-метода. Создана и реализована параллельная версия алгоритма AlEM с использованием библиотеки параллельного программирования MPI. Предложенная схема распределенных вычислений (по принципу round-robin) позволяет эффективно распараллелить перебор вершинных прямых в случае, когда их число превышает количество параллельных процессов. Проведено исследование масштабируемости предложенной параллельной версии алгоритма AlEM на вычислительном кластере. Результаты экспериментов на реальных и модельных задачах ЛП различной размерности показали, что алгоритм демонстрирует хорошую масштабируемость вплоть до 200 процессорных узлов. При этом параллельная эффективность на границе масштабируемости не опускается ниже 46%. Выполнено сравнительное тестирование параллельной версии алгоритма AlEM с параллельной реализацией симплекс-метода. Хотя алгоритм AlEM несколько уступает в абсолютной скорости симплекс-методу, это отставание сокращается с ростом размерности задачи. Важнейшим качественным преимуществом AlEM является его устойчивость к зацикливанию, что было подтверждено тестами на известных вырожденных задачах. Таким образом, алгоритм AlEM представляет собой эффективную альтернативу существующим методам линейного программирования для случаев, когда критически важны гарантии быстрой сходимости на вырожденных задачах и требуется использование вычислительных систем с массовым параллелизмом. В качестве направлений дальнейших исследований можно выделить следующие.

- Разработка гибридной версии алгоритма AlEM, использующей подобно симплекс-методу технику замены только одной базисной строки при переходе к следующей вершине, что устраняет необходимость комбинаторного перебора вершинных прямых.
- Интеграция в алгоритм AlEM искусственной нейронной сети для анализа многомерных образов допустимого многогранника, что позволит исключить операцию ортогональной проекции на подпространство по формуле (16), требующую вычисления обратной матрицы.

7. Обозначения. В данном разделе приведены основные обозначения, использованные в статье.

n — число переменных в системе ограничений (размерность пространства).
m — количество неравенств в системе ограничений.
k — количество уравнений в системе ограничений.
$\mathbb{R}^n$ — вещественное евклидово пространство размерности n.
$\langle *, * \rangle$ — скалярное произведение двух векторов.
$\hat{A}$ — матрица коэффициентов неравенств, входящих в систему ограничений.
$\bar{A}$ — матрица коэффициентов уравнений, входящих в систему ограничений.
$\hat{\boldsymbol{b}}$ — столбец правых частей неравенств, входящих в систему ограничений.
$\bar{\boldsymbol{b}}$ — столбец правых частей уравнений, входящих в систему ограничений.
$\boldsymbol{a}_i$ — i-я строка матрицы $\begin{bmatrix}\hat{A}\\ \bar{A}\end{bmatrix}$ $(i = 1, \ldots, m + k)$.
$\hat{I}$ — индексы (номера) строк матрицы $\hat{A}$: $\hat{I} = \{1, \ldots, m\}$.
$\bar{I}$ — индексы (номера) строк матрицы $\bar{A}$: $\bar{I} = \{m + 1, \ldots, m + k\}$.
P_i — полупространство, определяемое неравенством $\langle \boldsymbol{a}_i, \boldsymbol{x} \rangle \leqslant b_i$ при $i \in \hat{I}$.
$\hat{H}_i$ — граничная гиперплоскость, определяемая уравнением $\langle \boldsymbol{a}_i, \boldsymbol{x} \rangle = b_i$ при $i \in \hat{I}$.
$\bar{H}_i$ — опорная гиперплоскость, определяемая уравнением $\langle \boldsymbol{a}_i, \boldsymbol{x} \rangle = b_i$ при $i \in \bar{I}$.
$\hat{M}$ — ограничивающий многогранник (пересечение всех полупространств P_i).
$\bar{S}$ — опорное подпространство (пересечение всех опорных гиперплоскостей $\bar{H}_i$).
M — допустимый многогранник (область допустимых решений): $M = \hat{M} \cap \bar{S}$.
$\hat{I}_{\boldsymbol{v}}$ — индексы граничных гиперплоскостей, проходящие через вершину $\boldsymbol{v}$.
$\operatorname{rank}(A)$ — ранг матрицы A.
$A_{\hat{J} \cup \bar{I}}$ — матрица, включающая строки из $\hat{A}$ с индексами из $\hat{J}$ и все строки из $\bar{A}$.
$\boldsymbol{\pi}_{\hat{J} \cup \bar{I}}(\boldsymbol{x})$ — ортогональная проекция на подпространство $\bigcap\limits_{i \in \hat{J} \cup \bar{I}} H_i$.

Список литературы

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. Cham: Springer, 2019. 269–292. doi 10.1007/978-3-030-28565-4_23.
2. *Chung W.* Applying large-scale linear programming in business analytics // 2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), Singapore, December 06–09, 2015. IEEE Press, 2015. pp. 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 // J. Comput. Appl. Math. 2014. **263**. 392–404. doi 10.1016/j.cam.2013.12.003.
4. *Murty K.G.* Linear equations, inequalities, linear programs, and a new efficient algorithm // Tutorials in operations research: models, methods, and applications for innovative decision making. Catonsville: INFORMS, 2006. 1–36. doi 10.1287/educ.1063.0024.
5. *Данциг Д.Б.* Линейное программирование, его применения и обобщения. Москва: Изд-во "Прогресс", 1966.
6. *Tolla P.* A survey of some linear programming methods // Concepts of combinatorial optimization. 2-nd ed. Hoboken: Wiley, 2014. 157–188. doi 10.1002/9781119005216.ch7.
7. *Схрейвер А.* Теория линейного и целочисленного программирования. Т. 1. Москва: Мир, 1991.
8. *Kotiah T.C.T., Steinberg D.I.* Letter to the editor — On the possibility of cycling with the simplex method // Oper. Res. 1978. **26**, N 2. 374–376. doi 10.1287/OPRE.26.2.374.
9. *Gass S.I., Vinjamuri S.* Cycling in linear programming problems // Comput. Oper. Res. 2004. **31**, N 2. 303–311. doi 10.1016/S0305-0548(02)00226-5.
10. *Hall J.A.J., McKinnon K.I.M.* The simplest examples where the simplex method cycles and conditions where EXPAND fails to prevent cycling // Math. Program. Ser. B. 2004. **100**, N 1. 133–150. doi 10.1007/s10107-003-0488-1.
11. *Maros I.* Large scale LP problems // Computational techniques of the simplex method. International series in operations research and management science, Vol. 61. Boston: Springer, 2003. 49–56. doi 10.1007/978-1-4615-0257-9_3.
12. *Schlenkrich M., Parragh S.N.* Solving large scale industrial production scheduling problems with complex constraints: an overview of the state-of-the-art // 4th International Conference on Industry 4.0 and Smart Manufacturing. Elsevier: Procedia Computer Science, Vol. 217, 2023. pp. 1028–1037.
13. *Mamalis B., Pantziou G.* Advances in the parallelization of the simplex method // Algorithms, probability, networks, and games. Lecture Notes in Computer Science, Vol. 9295. Cham: Springer, 2015. 281–307. doi 10.1007/978-3-319-24024-4_17.
14. *Klee V., Minty G.J.* How good is the simplex algorithm? // Inequalities — III. Proceedings of the Third Symposium on Inequalities Held at the University of California, Los Angeles, Sept. 1–9, 1969. New York: Academic Press, 1972. pp. 159–175.
15. *Murty K.G.* Computational and algorithmic linear algebra and n-dimensional geometry. Singapore: World Scientific Publishing Company, 2014. doi 10.1142/8261.
16. *Стренг Г.* Линейная алгебра и ее применения. Москва: Мир, 1980.
17. *Khachiyan L.* Fourier–Motzkin elimination method // Encyclopedia of optimization. Boston: Springer, 2008. 1074–1077. doi 10.1007/978-0-387-74759-0_187.
18. *Жулев А.Э., Соколинский Л.Б., Соколинская И.М.* О вычислении вершины многогранника допустимых решений системы линейных ограничений // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2025. **14**, № 3. 5–27. doi 10.14529/cmse250301.
19. *Chase P.J.* Algorithm 382: combinations of M out of N objects [G6] // Communications of the ACM. 1970. **13**, N 6. 368–368. doi 10.1145/362384.362502.
20. *Gropp W.* MPI 3 and beyond: why MPI is successful and what challenges it faces // Recent Advances in the Message Passing Interface. EuroMPI 2012. Lecture Notes in Computer Science, Vol. 7490. Berlin: Springer, 2012. pp. 1–9.
21. *Gay D.M.* Electronic mail distribution of linear programming test problems // Mathematical Programming Society COAL Newsletter. 1985. **13**. 10–12.
22. *Koch T.* The final NETLIB-LP results // Operations Research Letters. 2004. **32**, N 2. 138–142. doi 10.1016/S0167-6377(03)00094-4.
23. *Жулев А.Э., Соколинский Л.Б.* AlEM: новый параллельный алгоритм линейного программирования для кластерных вычислительных систем // Суперкомпьютерные дни в России: Труды международной конференции. Москва, сентябрь 29–30, 2025 г., Москва: МАКС Пресс, 2025. 4–24. doi 10.29003/m4750.978-5-317-07451-7.

24. *Муртаф Б.* Современное линейное программирование. М.: Мир, 1984.
25. *Boisvert R.F., Pozo R., Remington K.A.* The matrix market exchange formats: initial design: tech. rep. / NISTIR 5935. Gaithersburg, MD: NIST, 1996. https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir5935.pdf. Cited December 29, 2025.
26. *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. Cham: Springer, 2022. pp. 43–55. doi 10.1007/978-3-031-11623-0_4.

Получена 4 декабря 2025 г. | *Принята* 17 декабря 2025 г. | *Опубликована* 21 января 2026 г.

Информация об авторах

Александр Эдуардович Жулев — аспирант; Южно-Уральский государственный университет (национальный исследовательский университет), пр-кт Ленина, д. 76, 454080, Челябинск, Российская Федерация.

Леонид Борисович Соколинский — д.ф.-м.н., заведующий кафедрой системного программирования; Южно-Уральский государственный университет (национальный исследовательский университет), пр-кт Ленина, д. 76, 454080, Челябинск, Российская Федерация.

References

1. Y. Xu, “Solving Large Scale Optimization Problems in the Transportation Industry and Beyond Through Column Generation,” in *Optimization in Large Scale Problems. Springer Optimization and Its Applications, Vol. 152* (Springer, Cham, 2019), pp. 269–292. doi 10.1007/978-3-030-28565-4_23.
2. W. Chung, “Applying Large-Scale Linear Programming in Business Analytics,” in *2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), Singapore, December 06–09, 2015* (IEEE Press, 2015), pp. 1860–1864. doi 10.1109/IEEM.2015.7385970.
3. J. Gondzio, J. A. Gruca, J. A. J. Hall, et al., “Solving Large-Scale Optimization Problems Related to Bell’s Theorem,” J. Comput. Appl. Math. **263**, 392–404 (2014). doi 10.1016/j.cam.2013.12.003.
4. K. G. Murty, “Linear Equations, Inequalities, Linear Programs, and a New Efficient Algorithm,” in *Tutorials in Operations Research: Models, Methods, and Applications for Innovative Decision Making* (INFORMS, Catonsville, 2006), pp. 1–36. doi 10.1287/educ.1063.0024.
5. G. B. Dantzig, *Linear Programming and Extensions* (Princeton University Press, Princeton, N.J., 1998).
6. P. Tolla, “A Survey of Some Linear Programming Methods,” in *Concepts of Combinatorial Optimization. 2-nd ed.* (Wiley, Hoboken, 2014), pp. 157–188. doi 10.1002/9781119005216.ch7.
7. A. Schrijver, *Theory of Linear and Integer Programming* (Wiley and Sons, New York, 1998).
8. T. C. T. Kotiah and D. I. Steinberg, “Letter to the Editor — On the Possibility of Cycling with the Simplex Method,” Oper. Res. **26** (2), 374–376 (1978). doi 10.1287/OPRE.26.2.374.
9. S. I. Gass and S. Vinjamuri, “Cycling in Linear Programming Problems,” Comput. Oper. Res. **31** (2), 303–311 (2004). doi 10.1016/S0305-0548(02)00226-5.
10. J. A. J. Hall and K. I. M. McKinnon, “The Simplest Examples where the Simplex Method Cycles and Conditions where EXPAND Fails to Prevent Cycling,” Math. Program. Ser. B **100** (1), 133–150 (2004). doi 10.1007/s10107-003-0488-1.
11. I. Maros, “Large Scale LP Problems,” in *Computational Techniques of the Simplex Method* (International Series in Operations Research and Management Science, Vol. 61) (Springer, Boston, 2003), pp. 49–56. doi 10.1007/978-1-4615-0257-9_3.
12. M. Schlenkrich and S. N. Parragh, “Solving Large Scale Industrial Production Scheduling Problems with Complex Constraints: An Overview of the State-of-the-Art,” in *4th International Conference on Industry 4.0 and Smart Manufacturing* (Procedia Computer Science, Vol. 217. Elsevier, 2023), pp. 1028–1037.
13. B. Mamalis and G. Pantziou, “Advances in the Parallelization of the Simplex Method,” in *Algorithms, Probability, Networks, and Games* (Lecture Notes in Computer Science, Vol. 9295) (Springer, Cham, 2015), pp. 281–307. doi 10.1007/978-3-319-24024-4_17.

14. V. Klee and G. J. Minty, "How Good Is the Simplex Algorithm?" in *Inequalities — III. Proceedings of the Third Symposium on Inequalities Held at the University of California, Los Angeles, Sept. 1–9, 1969* (Academic Press, New York, 1972), pp. 159–175.
15. K. G. Murty, *Computational and Algorithmic Linear Algebra and n-Dimensional Geometry* (World Scientific Publishing Company, Singapore, 2014). doi 10.1142/8261.
16. G. Strang, *Linear Algebra and Its Applications* (Academic Press, New York, 1980).
17. L. Khachiyan, "Fourier–Motzkin Elimination Method," in *Encyclopedia of Optimization* (Springer, Boston, 2008), pp. 1074–1077. doi 10.1007/978-0-387-74759-0_187.
18. A. E. Zhulev, L. B. Sokolinsky, and I. M. Sokolinskaya, "On Calculating a Vertex of Feasible Solutions Polytope of Linear Constraints System," Bull. South Ural State Univ. Ser. Comput. Math. Softw. Eng. **14** (3), 5–27 (2025). doi 10.14529/cmse250301.
19. P. J. Chase, "Algorithm 382: Combinations of M out of N Objects [G6]," Communications of the ACM **13** (6), 368–368 (1970). doi 10.1145/362384.362502.
20. W. Gropp, "MPI 3 and Beyond: Why MPI Is Successful and What Challenges It Faces," in *Recent Advances in the Message Passing Interface. EuroMPI 2012. Lecture Notes in Computer Science* (Springer, Berlin, 2012), Vol. 7490, pp. 1–9.
21. D. M. Gay, "Electronic Mail Distribution of Linear Programming Test Problems," Mathematical Programming Society COAL Newsletter **13**, 10–12 (1985).
22. T. Koch, "The Final NETLIB-LP Results," Operations Research Letters **32** (2), 138–142 (2004). doi 10.1016/S0167-6377(03)00094-4.
23. A. E. Zhulev and L. B. Sokolinsky, "AlEM: A New Parallel Linear Programming Algorithm for Cluster Computing Systems," in *Russian Supercomputing Days: Proceedings of the International Conference. Moscow, Russia, September 29—30, 2025* (MAKS Press, Moscow, 2025), pp. 4–24. doi 10.29003/m4750.978-5-317-07451-7. [in Russian].
24. B. A. Murtagh, *Advanced Linear Programming: Computation and Practice* (McGraw-Hill, New York, 1981).
25. R. F. Boisvert, R. Pozo, and K. A. Remington, *The Matrix Market Exchange Formats: Initial Design: tech. rep. / NISTIR 5935* (NIST, Gaithersburg, MD, 1996). https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir5935.pdf. Cited December 29, 2025.
26. N. Dolganina, E. Ivanova, R. Bilenko, and A. Rekachinsky, "HPC Resources of South Ural State University," in *Parallel Computational Technologies. PCT 2022. Communications in Computer and Information Science* (Cham: Springer, 2022), Vol. 1618, pp. 43–55. doi 10.1007/978-3-031-11623-0_4.

Received December 4, 2025 | *Accepted* December 17, 2025 | *Published* January 21, 2026

Information about the authors

Alexander E. Zhulev — PhD student; South Ural State University (National Research University), Lenin prospekt 76, 454080, Chelyabinsk, Russia.

Leoind B. Sokolinsky — Dr. Sci., Head of Computer Science Department; South Ural State University (National Research University), Lenin prospekt 76, 454080, Chelyabinsk, Russia.