УДК 519.852, 514.172.45, 519.168 **DOI: 10.14529/cmse260101**

О ПРОЕКЦИОННОМ МЕТОДЕ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ

© 2026 Л.Б. Соколинский, А.Э. Жулев, И.М. Соколинская

Южно-Уральский государственный университет
(454080 Челябинск, пр. им. В.И. Ленина, д. 76)
E-mail: leonid.sokolinsky@susu.ru, zhulevae@susu.ru, irina.sokolinskaya@susu.ru

Поступила в редакцию: 14.11.2025

В статье рассматривается проблема разработки эффективных методов проекционного типа для решения задач линейного программирования (ЛП). Предложен новый гибридный проекционный алгоритм HAlEM (Hybrid Along Edges Movement), сочетающий идеи проекционного подхода и симплекс-метода. Алгоритм начинает работу из произвольной вершины многогранника допустимых решений и осуществляет движение вдоль его ребер к оптимальной вершине. Для вычисления направления движения используется оригинальный метод двух-факторной проекции, обеспечивающий высокую вычислительную точность для любых задач ЛП. Основным преимуществом HAlEM перед предшествующими разработками (AlFaMove, AlEM) является отказ от комбинаторного перебора всех возможных комбинаций гиперплоскостей за счет использования техники обновления матричного базиса, заимствованной из симплекс-метода. Это позволяет избежать экспоненциальной вычислительной сложности. В статье представлена параллельная версия алгоритма, основанная на модели параллельных вычислений BSF и схеме «мастер-рабочие», позволяющих получить эффективную реализацию для суперкомпьютерных систем кластерного типа. Проведены вычислительные эксперименты на эталонных задачах из репозитория Netlib-LP и на серии параметризованных задач. Результаты экспериментов демонстрируют, что HAlEM, в отличие от симплекс-метода, обладает более высоким ресурсом параллелизма, позволяя эффективно использовать до нескольких десятков процессорных узлов, и показывает хорошую масштабируемость с параллельной эффективностью не ниже 51%. При этом алгоритм обеспечивает высокую точность вычислений на уровне машинного нуля.

Ключевые слова: линейное программирование, алгоритм HAlEM, проекционный метод, параллельная реализация, MPI, кластерная вычислительная система, исследование масштабируемости, Netlb-LP.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Соколинский Л.Б., Жулев А.Э., Соколинская И.М. О проекционном методе линейного программирования // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2026. Т. 15, № 1. С. 5–37. DOI: 10.14529/cmse260101.

Введение

Линейное программирование (ЛП) [1, 2] является одной из самых востребованных математических моделей для решения задач оптимизации в индустрии, экономике, науке и многих других сферах человеческой деятельности [3–6]. Основным методом решения задач ЛП на практике является симплекс-метод [7], позволяющий эффективно решать широкий класс оптимизационных задач с числом неизвестных до 50 000 [8]. Однако симплекс-методу присущ ряд недостатков, среди которых выделим следующие. Во-первых, симплекс-метод в общем случае характеризуется ограниченной масштабируемостью и не допускает эффективного распараллеливания более чем на 32 процессорных узлах [9]. Во-вторых, симплекс-метод может зацикливаться на вырожденных задачах ЛП [10], что не гарантирует его сходимость к решению [11, 12]. Поэтому остается актуальной задача разработки альтернативных эффективных методов ЛП, свободных от указанных недостатков. Одной из таких альтернатив являются методы проекционного типа, строящие на поверхности многогранника допустимых решений путь к оптимальной точке.

Идея проекционного метода чрезвычайно проста. Пусть в евклидовом пространстве $\mathbb{R}^n$ задана задача ЛП стандартного вида

$$\arg\max\{\langle \boldsymbol{c}, \boldsymbol{x}\rangle \mid A\boldsymbol{x} \leqslant \boldsymbol{b},\ \boldsymbol{x} \geqslant \boldsymbol{0}\}$$

с ограниченной областью допустимых решений, представляющей собой замкнутый выпуклый многогранник M. Стартуя из произвольной граничной точки $\boldsymbol{v}_0$ многогранника M, проекционный метод вычисляет точку

$$\boldsymbol{z} = \boldsymbol{v}_0 + \delta \boldsymbol{e}_{\boldsymbol{c}},$$

где $\boldsymbol{e}_{\boldsymbol{c}}$ — единичный вектор, сонаправленный с вектором $\boldsymbol{c}$, δ — некоторое положительное вещественное число (см. рис. 1). Затем выполняется метрическая проекция точки $\boldsymbol{z}$, дающая

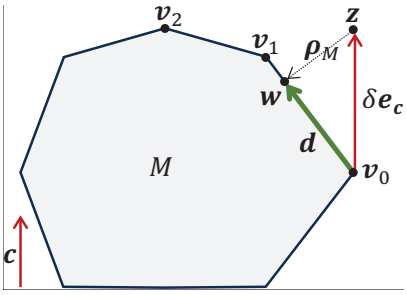

Рис. 1. Проекционный метод

на многограннике M точку $\boldsymbol{w}$:

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

Выбор числа δ должен гарантировать, что $\boldsymbol{w}$ будет на той же грани, что и $\boldsymbol{u}$. Далее вычисляется направляющий вектор $\boldsymbol{d} = \boldsymbol{w} - \boldsymbol{v}_0$, который позволяет получить следующее приближение $\boldsymbol{v}_1$:

$$\boldsymbol{v}_1 = \arg\max\{\|\boldsymbol{x} - \boldsymbol{v}_0\| \mid \boldsymbol{x} = \lambda \boldsymbol{d} + \boldsymbol{v}_0,\ \lambda \in \mathbb{R}_{>0},\ \boldsymbol{x} \in M\}.$$

Выполняя аналогичные действия для $\boldsymbol{v}_1$, получим приближение $\boldsymbol{v}_2$. Процесс заканчивается на k-том приближении, для которого

$$\boldsymbol{\rho}_M(\boldsymbol{v}_k + \delta \boldsymbol{e}_{\boldsymbol{c}}) = \boldsymbol{v}_k.$$

На рис. 1 этой ситуации соответствует точка $\boldsymbol{v}_2$. Таким образом получаем решение задачи ЛП.

В работе [13] описан апекс-метод решения задачи ЛП, реализующий этот подход в виде итерационного численного алгоритма. Для приближенного вычисления метрической проекции на выпуклый многогранник M апекс-метод использует M-фейеровское отображение $\boldsymbol{\varphi}_M : \mathbb{R}^n \to \mathbb{R}^n$, характеризующееся тем, что образ $\boldsymbol{\varphi}_M(\boldsymbol{x})$ любой точки $\boldsymbol{x} \notin M$ будет ближе к многограннику M, чем $\boldsymbol{x}$:

$$\forall \boldsymbol{x} \notin M, \forall \boldsymbol{y} \in M : \|\boldsymbol{\varphi}(\boldsymbol{x}) - \boldsymbol{y}\| < \|\boldsymbol{x} - \boldsymbol{y}\|.$$

Многократно применяя фейеровское отображение к произвольной внешней точке $\boldsymbol{x}^{(0)} \notin M$, мы получим точку на границе многогранника M, являющуюся некоторым приближением

метрической проекции. Указанный фейеровский процесс называется псевдопроектированием, а получившаяся точка — псевдопроекцией. Хотя апекс-метод способен решать некоторые реальные задачи ЛП, он обладает двумя существенными недостатками. Первый недостаток заключается в том, что фейеровский процесс, используемый для приближенного вычисления метрической проекции, имеет низкую линейную скорость сходимости:

$$\left\| \boldsymbol{x}^{(k+1)} - \boldsymbol{\varphi}_M \left(\boldsymbol{x}^{(0)} \right) \right\| \leqslant C q^k,$$

где $0 < C < +\infty$ — некоторая константа, а $q \in (0, 1)$ — параметр, зависящий от углов между гиперплоскостями, соответствующими граням многогранника M. Это означает, что расстояние между соседними приближениями с каждой итерацией уменьшается в геометрической прогрессии с коэффициентом, меньше единицы. Для малых углов скорость сходимости может падать до значений, близких к нулю. Второй недостаток связан с ограничением на длину «выводящего» вектора $\delta \boldsymbol{e_c}$: когда исходная граничная точка $\boldsymbol{v}$ находится близко к противоположному краю грани, значение δ должно быть достаточно малым, чтобы метрическая проекция не оказалась за пределами этой грани. Это существенно ограничивает точность вычисления метрической проекции.

В работах [14, 15] предложен алгоритм AlFaMove (Along Faces Movement), в котором для решения задачи ЛП предлагается вычислять метрическую проекцию не на многогранник, а на аффинное подпространство L, образующее грань допустимого многогранника M, как это показано на рис. 2. В этом случае выводящий вектор $\delta \boldsymbol{e_c}$ может иметь большую

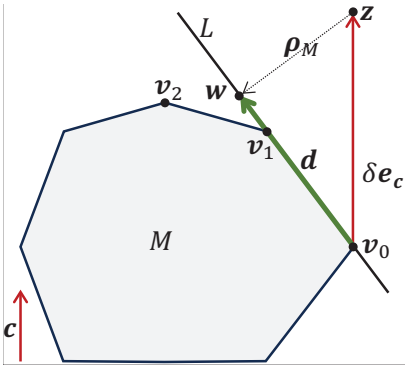

Рис. 2. Алгоритм AlFaMove

длину, что существенно увеличивает точность вычисления направляющего вектора $\boldsymbol{d}$. Для любой грани многогранника M существует комбинация гиперплоскостей из системы ограничений задачи ЛП, пересечением которых будет аффинное подпространство L, образующее эту грань. В соответствии с этим алгоритм AlFaMove перебирает все возможные комбинации гиперплоскостей, проходящих через точку $\boldsymbol{v}_0$. Для каждой комбинации вычисляется свой направляющий вектор $\boldsymbol{d}$. Отбрасываются комбинации, для которых луч $X = \{\boldsymbol{x} \in \mathbb{R} \mid \boldsymbol{x} = \boldsymbol{v}_0 + \lambda \boldsymbol{d},\ \lambda \in \mathbb{R}_{\geqslant 0}\}$ имеет с допустимым многогранником M только одну общую точку, являющуюся его начальной точкой $\boldsymbol{v}_0$. Для каждой оставшейся комбинации вычисляется точка $\boldsymbol{v}_{next}$ по формуле

$$\boldsymbol{v}_{next} = \boldsymbol{v}_0 + \frac{b_{i^*} - \langle \boldsymbol{a}_{i^*}, \boldsymbol{v}_0 \rangle}{\langle \boldsymbol{a}_{i^*}, \boldsymbol{d} \rangle} \boldsymbol{d}, \qquad \text{(i)}$$

где

$$i^* \in \operatorname{Arg}\min_i \left\{ \frac{b_i - \langle \boldsymbol{a}_i, \boldsymbol{v}_0 \rangle}{\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle} \;\middle|\; \langle \boldsymbol{a}_i, \boldsymbol{d} \rangle > 0 \right\}, \tag{ii}$$

i пробегает строки матрицы A. В качестве следующего приближения $\boldsymbol{v}_1$ из всех полученных точек $\boldsymbol{v}_{next}$ выбирается та, в которой достигается наибольшее значение целевой функции. Повторяя описанные действия, алгоритм AlFaMove за конечное число итераций придет в допустимую точку с максимальным значением целевой функции. Основным недостатком этого подхода является необходимость комбинаторного перебора возможных комбинаций гиперплоскостей, проходящих через точку текущего приближения. Если через точку проходит k гиперплоскостей, то количество комбинаций будет равно 2^k. Таким образом, алгоритм AlFaMove имеет экспоненциальную вычислительную сложность. Это делает затруднительным его применение даже в случае простых многогранников[1] размерности больше 30. Также отметим, что для вычисления метрической проекции алгоритм AlFaMove по-прежнему использует медленные фейеровские процессы.

В недавней работе [16] предложен новый проекционный алгоритм ЛП, получивший название AlEM (Along Edges Movement). В отличие от алгоритма AlFaMove, алгоритм AlEM начинает свою работу в произвольной вершине многогранника допустимых решений и двигается к оптимальной вершине только по ребрам. Для любого ребра существует $n-1$ гиперплоскостей из системы ограничений задачи ЛП, пересечением которых будет прямая, образующая это ребро. Если через вершину проходит k гиперплоскостей ($k \geqslant n$), то количество комбинаций может быть вычислено по следующей формуле:

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

Для вычисления метрической проекции точки на прямую алгоритм AlEM вместо фейеровских отображений использует формулу ортогональной проекции на аффинное подпространство [17], что оказывается несравненно более эффективным. Вычислительные эксперименты показали, что быстродействие алгоритма AlEM на задачах ЛП, область допустимых решений которых является простым многогранником, сравнимо с быстродействием симплекс-метода. Однако, в отличии от симплекс-метода, алгоритм AlEM допускает эффективное распараллеливание и исключает возможность зацикливания на вырожденных задачах. К сожалению, в реальных задачах ЛП количество гиперплоскостей, проходящих через вершины многогранника допустимых решений, существенно превышает размерность пространства, что приводит к необходимости комбинаторного перебора.

В этой статье мы предлагаем новый проекционный алгоритм HAlEM (Hybrid Along Edges Movement), который совмещает в себе свойства алгоритма AlEM и симплекс-метода. Принцип работы алгоритма HAlEM заключается в следующем. В начальной вершине $\boldsymbol{v}_0$ формируется матричный базис A_0 из n линейно независимых строк матрицы A, соответствующих n гиперплоскостям, проходящим через $\boldsymbol{v}_0$. Каждая комбинация из $n-1$ таких гиперплоскостей определяет прямую, являющуюся результатом их пересечения. Всего получится n линейно независимых прямых, называемых реберными. Каждая реберная прямая проходит через ребро, инцидентное вершине $\boldsymbol{v}_0$. С помощью ортогональной проекции для каждой реберной прямой вычисляется направляющий вектор $\boldsymbol{d}$. Затем для всех реберных

[1] Выпуклый многогранник размерности n является простым, если из любой его вершины выходит ровно n ребер.

прямых с помощью формул (i), (ii) вычисляются точки $\boldsymbol{v}_{next}$, являющиеся смежными вершинами по отношению к $\boldsymbol{v}_0$. В качестве $\boldsymbol{v}_1$ выбирается та вершина, в которой достигается наибольшее значение целевой функции. На основе матричного базиса A_0 формируется новый матричный базис A_1 путем замены строки, не участвовавшей в формировании ребра перехода, на некоторую строку из матрицы A, соответствующую гиперплоскости, проходящей через вершину $\boldsymbol{v}_1$, так, чтобы сохранялась линейная независимость матричного базиса A_1. Процесс продолжается до тех пор, пока не будет достигнута вершина, у которой отсутствуют ребра, ведущие к увеличению целевой функции. Эта вершина будет искомым решением задачи ЛП.

Статья организована следующим образом. Раздел 1 содержит обозначения, определения и утверждения, необходимые для описания алгоритма HAlEM и его численной реализации. В разделе 2 дается формализованное описание алгоритма HAlEM. Раздел 3 посвящен параллельной версии алгоритма HAlEM. В разделе 4 представлены информация о программной реализации алгоритма HAlEM и результаты вычислительных экспериментов. В заключении суммируются полученные результаты и намечаются направления дальнейших исследований. В конце статьи приводится сводка используемых математических обозначений.

1. Необходимые определения и утверждения

В этом разделе приводятся определения и утверждения, необходимые для формального описания алгоритма HAlEM.

В евклидовом пространстве $\mathbb{R}^n$ размерности $n > 1$ имеется задача ЛП общего вида с системой из m ограничений, включающей в себя k линейных уравнений и $m - k$ линейных неравенств:

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

Здесь и далее $\langle *, * \rangle$ обозначает скалярное произведение двух векторов. Предполагается, что система (1) включает в себя также неравенства вида

$$\begin{aligned}
&-x_1 \leqslant 0, \\
&\cdots\cdots\cdots \\
&-x_n \leqslant 0.
\end{aligned} \tag{2}$$

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

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

Для $J = \{j_1, \ldots, j_q\} \subseteq \{1, \ldots, m\}$, $|J| = q > 0$ определим матрицу

$$A_J = \begin{bmatrix} \boldsymbol{a}_{j_1} \\ \vdots \\ \boldsymbol{a}_{j_q} \end{bmatrix}$$

и столбец

$$\boldsymbol{b}_J = \begin{bmatrix} b_{j_1} \\ \vdots \\ b_{j_q} \end{bmatrix}.$$

Обозначим через $\bar{I}$ множество индексов уравнений системы (1), и через $\hat{I}$ — множество индексов неравенств системы (1):

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

Тогда систему ограничений (1) можно представить в матричном виде

$$A_{\bar{I}}\boldsymbol{x} = \boldsymbol{b}_{\bar{I}},$$
$$A_{\hat{I}}\boldsymbol{x} \leqslant \boldsymbol{b}_{\hat{I}}.$$

Без ограничения общности мы можем считать, что матрица $A_{\bar{I}}$ имеет полный ранг:

$$\operatorname{rank}(A_{\bar{I}}) = k. \tag{3}$$

Мы также будем полагать, что

$$k < n,$$

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

Известно, что область допустимых решений системы (1) представляет собой замкнутый выпуклый многогранник

$$M = \left\{\boldsymbol{x} \in \mathbb{R}^n \middle| A_{\bar{I}}\boldsymbol{x} = \boldsymbol{b}_{\bar{I}},\ A_{\hat{I}}\boldsymbol{x} \leqslant \boldsymbol{b}_{\hat{I}}\right\},$$

называемый допустимым. Мы будем предполагать, что допустимый многогранник M является ограниченным множеством. В этом случае задача ЛП всегда имеет решение. Поскольку система (1) включает в себя неравенства (2), то из ограниченности допустимого многогранника M следует, что общее число неравенств должно превышать размерность пространства:

$$m - k > n. \tag{4}$$

Каждому индексу $i \in \bar{I} \cup \hat{I}$ соответствует гиперплоскость

$$H_i = \{\boldsymbol{x} \in \mathbb{R}^n \mid \langle \boldsymbol{a}_i, \boldsymbol{x}\rangle = b_i\}. \tag{5}$$

В совокупности эти гиперплоскости образуют грани допустимого многогранника M.

Определение 1. Пусть точка $\boldsymbol{v}$ является вершиной допустимого многогранника M. Базисным индексом вершины $\boldsymbol{v}$ будем называть множество индексов $\mathcal{V}$, удовлетворяющее сле-

дующим условиям:

$$\mathcal{V} \subseteq \hat{I}, \tag{6}$$

$$|\mathcal{V}| = n - k, \tag{7}$$

$$\forall i \in \mathcal{V} : \langle \boldsymbol{a}_i, \boldsymbol{v} \rangle = b_i, \tag{8}$$

$$\operatorname{rank}\left(A_{\bar{I} \cup \mathcal{V}}\right) = n. \tag{9}$$

Квадратную матрицу $A_{\bar{I} \cup \mathcal{V}}$ размера $n \times n$ будем называть базисной.

С геометрической точки зрения это означает, что пересечение гиперплоскостей, соответствующих строкам базисной матрицы $A_{\bar{I} \cup \mathcal{V}}$, образует вершину $\boldsymbol{v}$:

$$\{\boldsymbol{v}\} = \bigcap_{j \in \bar{I} \cup \mathcal{V}} H_j.$$

Базисная матрица $A_{\bar{I} \cup \mathcal{V}}$ задает $n - k$ различных реберных прямых[2], проходящих через вершину $\boldsymbol{v}$:

$$\forall i \in \mathcal{V} : \; L_i = \bigcap_{j \in \bar{I} \cup \mathcal{E}_i} H_j, \tag{10}$$

где

$$\mathcal{E}_i = \mathcal{V} \setminus \{i\}.$$

Множество индексов $\mathcal{E}_i$, однозначно определяющих i-тую реберную прямую, будем называть реберным индексом. Реберную прямую L_i ($i \in \bar{I} \cup \mathcal{V}$) удобно представлять в параметрическом виде

$$L_i = \left\{ \boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{x} = \boldsymbol{v} + \lambda \boldsymbol{d}, \, \lambda \in \mathbb{R} \right\}.$$

Направляющий вектор $\boldsymbol{d} \in \mathbb{R}^n$ можно получить следующим образом. Выберем произвольный ненулевой «выводящий» вектор $\boldsymbol{g}$, не являющийся ортогональным по отношению к прямой L_i, и вычислим вторую точку $\boldsymbol{w} \neq \boldsymbol{v}$ на прямой L_i:

$$\boldsymbol{w} = \boldsymbol{\pi}_{\bar{I} \cup \mathcal{E}_i}(\boldsymbol{v} + \boldsymbol{g}).$$

Здесь $\boldsymbol{\pi}_{\bar{I} \cup \mathcal{E}_i}(*)$ обозначает ортогональную проекцию на подпространство[3] $\bigcap_{j \in \bar{I} \cup \mathcal{E}_i} H_j$, образующее реберную прямую L_i в соответствии с формулой (10). Ортогональная проекция на подпространство вычисляется через псевдообратную матрицу по известной формуле [17]:

$$\boldsymbol{\pi}_{\bar{I} \cup \mathcal{E}_i}(\boldsymbol{x}) = \boldsymbol{x} - A^T_{\bar{I} \cup \mathcal{E}_i} \left(A_{\bar{I} \cup \mathcal{E}_i} A^T_{\bar{I} \cup \mathcal{E}_i} \right)^{-1} \left(A_{\bar{I} \cup \mathcal{E}_i} \boldsymbol{x} - \boldsymbol{b}_{\bar{I} \cup \mathcal{E}_i} \right). \tag{11}$$

Заметим, что матрица $A_{\bar{I} \cup \mathcal{E}_i}$ имеет полностроч ный ранг в силу того, что базисная матрица $A_{\bar{I} \cup \mathcal{V}}$ имеет полный ранг и $\mathcal{E}_i \subset \mathcal{V}$. Поэтому матрица $A_{\bar{I} \cup \mathcal{E}_i} A^T_{\bar{I} \cup \mathcal{E}_i}$ является обратимой (см. утверждение 3F в [18]). В итоге получаем искомый направляющий вектор

$$\boldsymbol{d} = \boldsymbol{w} - \boldsymbol{v}.$$

Следующие утверждения понадобятся нам для выполнения прохода по ребру от одной вершины к другой.

[2]Реберная прямая — это прямая, которой принадлежит некоторое ребро многогранника.

[3]Здесь и далее мы опускаем прилагательное «аффинное».

Утверждение 1. [16] Пусть в контексте системы ограничений (1) 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 $\boldsymbol{d}$:

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

Тогда

$$\text{если} \quad \langle \boldsymbol{a}_i, \boldsymbol{d} \rangle \leqslant 0, \quad \text{то} \quad X \subset P_i, \tag{12}$$

$$\text{если} \quad \langle \boldsymbol{a}_i, \boldsymbol{d} \rangle > 0, \quad \text{то} \quad X \cap H_i = \left\{ \boldsymbol{v} + \frac{b_i - \langle \boldsymbol{a}_i, \boldsymbol{v} \rangle}{\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle} \boldsymbol{d} \right\}. \tag{13}$$

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 (13).

Утверждение 2. Пусть в контексте системы ограничений (1) заданы точка $\boldsymbol{v} \in M$ и вектор $\boldsymbol{d} \in \mathbb{R}^n$ такой, что

$$\forall i \in \bar{I}: \ \langle \boldsymbol{a}_i, \boldsymbol{d} \rangle = 0. \tag{14}$$

Определим луч, исходящий из точки $\boldsymbol{v}$ в направлении вектора $\boldsymbol{d}$:

$$X = \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{x} = \boldsymbol{v} + \lambda \boldsymbol{d},\ \lambda \in \mathbb{R}_{\geqslant 0}\}. \tag{15}$$

Тогда

$$X \cap M = \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{x} = \lambda \boldsymbol{v} + (1 - \lambda)\boldsymbol{q},\ 0 \leqslant \lambda \leqslant 1\},$$

где

$$\boldsymbol{q} = \boldsymbol{v} + \gamma \boldsymbol{d}, \tag{16}$$

$$\gamma = \min \left\{ \frac{b_i - \langle \boldsymbol{a}_i, \boldsymbol{v} \rangle}{\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle} \middle| i \in \hat{I},\ \langle \boldsymbol{a}_i, \boldsymbol{d} \rangle > 0 \right\}. \tag{17}$$

Другими словами, пересечение луча X и допустимого многогранника M представляет собой отрезок между точками $\boldsymbol{v}$ и $\boldsymbol{q}$, где $\boldsymbol{q}$ вычисляется по формулам (16) и (17).

Доказательство. Прежде всего удостоверимся, что точка $\boldsymbol{q}$, вычисляемая по формуле (16), принадлежит лучу X. Поскольку $\boldsymbol{v} \in M$, то для всех $i \in \hat{I}$ справедливо

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

Это равносильно

$$b_i - \langle \boldsymbol{a}_i, \boldsymbol{v} \rangle \geqslant 0.$$

Следовательно,

$$\text{если} \quad \langle \boldsymbol{a}_i, \boldsymbol{d} \rangle > 0, \quad \text{то} \quad \frac{b_i - \langle \boldsymbol{a}_i, \boldsymbol{v} \rangle}{\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle} \geqslant 0. \tag{18}$$

С учетом (17) отсюда получаем

$$\gamma \geqslant 0.$$

В соответствии с (15) и (16) это означает, что точка $\boldsymbol{q}$ принадлежит лучу X.

Теперь нужно удостовериться, что точка $\boldsymbol{q}$ принадлежит допустимому многограннику M. Сначала покажем, что точка $\boldsymbol{q}$, вычисляемая по формуле (16), удовлетворяет всем уравнениям системы ограничений (1). Возьмем произвольное $i \in \bar{I}$ и покажем, что $\boldsymbol{q}$ удовлетворяет уравнению

$$\langle \boldsymbol{a}_i, \boldsymbol{x} \rangle = b_i. \tag{19}$$

Так как $\boldsymbol{v} \in M$, имеем

$$\langle \boldsymbol{a}_i, \boldsymbol{v} \rangle = b_i. \tag{20}$$

Подставляя $\boldsymbol{q}$ в левую часть неравенства (19) вместо $\boldsymbol{x}$, с учетом (14), (16) и (20) получаем:

$$\langle \boldsymbol{a}_i, \boldsymbol{q} \rangle = \langle \boldsymbol{a}_i, \boldsymbol{v} + \gamma \boldsymbol{d} \rangle = \langle \boldsymbol{a}_i, \boldsymbol{v} \rangle + \gamma \langle \boldsymbol{a}_i, \boldsymbol{d} \rangle = \langle \boldsymbol{a}_i, \boldsymbol{v} \rangle = b_i.$$

То есть, $\boldsymbol{q}$ удовлетворяет уравнению (19).

Теперь покажем, что точка $\boldsymbol{q}$, вычисляемая по формулам (16) и (17), удовлетворяет всем неравенствам системы ограничений (1). Возьмем произвольное $i \in \hat{I}$ и покажем, что $\boldsymbol{q}$ удовлетворяет неравенству

$$\langle \boldsymbol{a}_i, \boldsymbol{q} \rangle \leqslant b_i. \tag{21}$$

Сначала предположим, что выполняется условие

$$\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle \leqslant 0. \tag{22}$$

В этом случае, в соответствии с (12), все точки луча X, включая $\boldsymbol{q}$, принадлежат полупространству P_i. Следовательно, $\boldsymbol{q}$ удовлетворяет неравенству (21).

Теперь предположим, что выполняется условие

$$\langle \boldsymbol{a}_i, \boldsymbol{d} \rangle > 0. \tag{23}$$

Согласно (13) луч X пересекает гиперплоскость H_i в точке

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

Это означает, что

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

Отсюда получаем

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

С учетом (17), (18) и (23) из последней формулы следует

$$\langle \boldsymbol{a}_i, \boldsymbol{v} \rangle + \gamma \langle \boldsymbol{a}_i, \boldsymbol{d} \rangle \leqslant b_i,$$

что равносильно

$$\langle \boldsymbol{a}_i, \boldsymbol{v} + \gamma \boldsymbol{d} \rangle \leqslant b_i.$$

Вместе с (16) это дает

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

Таким образом, точка $\boldsymbol{q}$, вычисляемая по формулам (16) и (17), принадлежит допустимому многограннику M. Поскольку M выпуклый, это означает, что весь отрезок между точками $\boldsymbol{v}$ и $\boldsymbol{q}$ принадлежит этому многограннику.

Осталось убедиться, что точка $\boldsymbol{q}$ является граничной точкой допустимого многогранника M. Для этого достаточно показать, что для любого $\varepsilon > 0$ точка $\boldsymbol{q}+\varepsilon\boldsymbol{d}$ не принадлежит M. В соответствии с (13), (16) и (17) существует $i' \in \hat{I}$ такой, что

$$\langle \boldsymbol{a}_{i'}, \boldsymbol{d} \rangle > 0 \tag{24}$$

и

$$\langle \boldsymbol{a}_{i'}, \boldsymbol{v} + \gamma \boldsymbol{d} \rangle = b_{i'}.$$

Последнее равносильно

$$\langle \boldsymbol{a}_{i'}, \boldsymbol{v} \rangle + \gamma \langle \boldsymbol{a}_{i'}, \boldsymbol{d} \rangle = b_{i'}.$$

В силу (24) отсюда следует

$$\langle \boldsymbol{a}_{i'}, \boldsymbol{v} \rangle + \gamma \langle \boldsymbol{a}_{i'}, \boldsymbol{d} \rangle + \varepsilon \langle \boldsymbol{a}_{i'}, \boldsymbol{d} \rangle > b_{i'},$$

что эквивалентно

$$\langle \boldsymbol{a}_{i'}, \boldsymbol{v} + \gamma \boldsymbol{d} + \varepsilon \boldsymbol{d} \rangle > b_{i'}.$$

Таким образом, с учетом (16) получаем

$$\langle \boldsymbol{a}_{i'}, \boldsymbol{q} + \varepsilon \boldsymbol{d} \rangle > b_{i'}.$$

Это означает, что точка $\boldsymbol{q} + \varepsilon\boldsymbol{d}$ не принадлежит полупространству $P_{i'}$ и, следовательно, не принадлежит допустимому многограннику M.

Утверждение доказано. □

Следствие 1. Пусть в контексте системы ограничений (1) имеется вершина $\boldsymbol{v}' \in M$, являющаяся начальной точкой ребра, соответствующего реберной прямой

$$L = \left\{ \boldsymbol{x} \in \mathbb{R}^n \middle| \boldsymbol{x} = \boldsymbol{v}' + \lambda \boldsymbol{d},\ \lambda \in \mathbb{R} \right\}.$$

Тогда вершина $\boldsymbol{v}'' \in M$, являющаяся конечной точкой этого ребра, может быть вычислена по формуле

$$\boldsymbol{v}'' = \boldsymbol{v}' + \frac{b_{j^*} - \langle \boldsymbol{a}_{j^*}, \boldsymbol{v} \rangle}{\langle \boldsymbol{a}_{j^*}, \boldsymbol{d} \rangle} \boldsymbol{d}, \tag{25}$$

где

$$j^* \in \operatorname{Arg\,min} \left\{ \frac{b_j - \langle \boldsymbol{a}_j, \boldsymbol{v} \rangle}{\langle \boldsymbol{a}_j, \boldsymbol{d} \rangle} \;\middle|\; j \in \hat{I},\ \langle \boldsymbol{a}_j, \boldsymbol{d} \rangle > 0 \right\}. \tag{26}$$

Гиперплоскость H_{j^*} в этом случае называется ограничивающей по отношению к реберной прямой L.

Утверждение 3. Пусть в контексте системы ограничений (1) имеется вершина $\boldsymbol{v}' \in M$ с базисным индексом $\mathcal{V}'$. Зафиксируем некоторый $i^* \in \mathcal{V}'$. Ему соответствует реберная прямая L_{i^*}, вычисляемая по формуле

$$L_{i^*} = \bigcap_{j \in \bar{I} \cup \mathcal{V}' \setminus \{i^*\}} H_j. \tag{27}$$

Пусть также известен направляющий вектор $\boldsymbol{d}$ прямой L_{i^*}:

$$L_{i^*} = \left\{ \boldsymbol{x} \in \mathbb{R}^n \middle| \boldsymbol{x} = \boldsymbol{v}' + \lambda \boldsymbol{d},\, \lambda \in \mathbb{R} \right\}. \tag{28}$$

Пусть точка $\boldsymbol{v}''$ является конечной вершиной рассматриваемого ребра. Положим

$$J^* = \operatorname{Arg\,min} \left\{ \frac{b_j - \langle \boldsymbol{a}_j, \boldsymbol{v}' \rangle}{\langle \boldsymbol{a}_j, \boldsymbol{d} \rangle} \middle| j \in \hat{I},\, \langle \boldsymbol{a}_j, \boldsymbol{d} \rangle > 0 \right\}. \tag{29}$$

Тогда для любого $j^* \in J^*$ множество

$$\mathcal{V}'' = \left(\mathcal{V}' \setminus \{i^*\} \right) \cup \{j^*\} \tag{30}$$

будет являться базисным индексом для вершины $\boldsymbol{v}''$.

Доказательство. Необходимо для вершины $\boldsymbol{v}''$ проверить выполнение условий (6)–(9) из определения 1. По построению очевидно, что выполняются условия (6) и (7).

Проверим выполнение условия (8). В соответствии со следствием 1 имеем

$$\boldsymbol{v}'' = \boldsymbol{v}' + \frac{b_{j^*} - \langle \boldsymbol{a}_{j^*}, \boldsymbol{v}' \rangle}{\langle \boldsymbol{a}_{j^*}, \boldsymbol{d} \rangle} \boldsymbol{d}.$$

Отсюда

$$\langle \boldsymbol{a}_{j^*}, \boldsymbol{v}'' \rangle = \left\langle \boldsymbol{a}_{j^*}, \boldsymbol{v}' + \frac{b_{j^*} - \langle \boldsymbol{a}_{j^*}, \boldsymbol{v}' \rangle}{\langle \boldsymbol{a}_{j^*}, \boldsymbol{d} \rangle} \boldsymbol{d} \right\rangle = \langle \boldsymbol{a}_{j^*}, \boldsymbol{v}' \rangle + \frac{b_{j^*} - \langle \boldsymbol{a}_{j^*}, \boldsymbol{v}' \rangle}{\langle \boldsymbol{a}_{j^*}, \boldsymbol{d} \rangle} \langle \boldsymbol{a}_{j^*}, \boldsymbol{d} \rangle = b_{j^*},$$

то есть,

$$\langle \boldsymbol{a}_{j^*}, \boldsymbol{v}'' \rangle = b_{j^*}. \tag{31}$$

Таким образом, условие (8) также выполняется.

Осталось проверить выполнение условия (9) для $\boldsymbol{v}''$. Так как $\mathcal{V}'$ является базисным индексом для $\boldsymbol{v}'$, то в соответствии с (9) имеем

$$\operatorname{rank}\left(A_{\bar{I} \cup \mathcal{V}'}\right) = n.$$

Поскольку $A_{\bar{I} \cup \mathcal{V}'}$ является квадратной матрицей размера $n \times n$, отсюда получаем

$$\operatorname{rank}\left(A_{\bar{I} \cup \mathcal{V}' \setminus \{i^*\}}\right) = n - 1. \tag{32}$$

Предположим, что для $\boldsymbol{v}''$ не выполняется условие (9). В сочетании с (32) это означает, что

$$\operatorname{rank}\left(A_{\bar{I} \cup (\mathcal{V}' \setminus \{i^*\}) \cup \{j^*\}}\right) = n - 1.$$

Согласно (27) в этом случае

$$L_{i^*} = \bigcap_{j \in \bar{I} \cup (\mathcal{V}' \setminus \{i^*\}) \cup \{j^*\}} H_j.$$

Вместе с (28) отсюда следует

$$\forall j \in \bar{I} \cup \left(\mathcal{V}' \setminus \{i^*\}\right) \cup \{j^*\} : \ \langle \boldsymbol{a}_j, \boldsymbol{d} \rangle = 0.$$

В частности,

$$\langle \boldsymbol{a}_{j^*}, \boldsymbol{d} \rangle = 0.$$

Получили противоречие с (29).

Утверждение доказано. □

2. Алгоритм HAlEM

Данный раздел посвящен описанию алгоритма HAlEM, строящего из произвольной вершины $\boldsymbol{v}$ путь вдоль ребер допустимого многогранника M к вершине, в которой достигается максимум целевой функции задачи ЛП. Псевдокод HAlEM представлен в виде алгоритма 1. Поясним шаги алгоритма 1. На шаге 1 вводятся размерность пространства n, количество ограничений m, количество уравнений k в системе ограничений, матрица коэффициентов A и столбец свободных членов (правых частей) $\boldsymbol{b}$ системы ограничений, градиент $\boldsymbol{c}$ линейной целевой функции и координаты начальной вершины $\boldsymbol{v}_0$ допустимого многогранника M. Эти данные являются глобальными в том смысле, что они доступны всем подпрограммам-функциям, используемым в алгоритме 1. Предполагается, что ограничения задачи ЛП приведены к виду (1). Также предполагается, что среди уравнений нет избыточных, то есть $\operatorname{rank}(A_{\bar{I}}) = k$, где k — число уравнений. Допускается случай, когда $k = 0$. Шаги 2 и 3 формируют множество индексов уравнений $\bar{I}$ и множество индексов неравенств $\hat{I}$, составляющих систему ограничений (1). Шаг 4 с помощью функции BasisIndex формирует базисный индекс $\mathcal{V}_0 \subseteq \hat{I}$ для начальной вершины $\boldsymbol{v}_0$. Такой базисный индекс существует в силу (4). Псевдокод функции BasisIndex представлен в виде алгоритма 2. Шаг 5 устанавливает счетчик итераций t в значение 0. На шаге 6 логической переменной *exit* присваивается значение «ложь».

Основной цикл **repeat/until** (шаги 7–36) выполняет последовательные переходы от вершины к вершине по ребрам допустимого многогранника, пока не будет достигнута оптимальная вершина. Один переход соответствует одной итерации этого цикла.

Вложенный цикл **loop** (шаги 8–35) выполняет переход от вершины $\boldsymbol{v}_t$ к следующей вершине $\boldsymbol{v}_{t+1}$ с бóльшим значением целевой функции. На шаге 9 точке $\boldsymbol{v}^*$ присваиваются координаты текущей вершины $\boldsymbol{v}_t$.

Следующий за этим цикл **for all** (шаги 10–22) находит для текущей вершины $\boldsymbol{v}_t$ ребро с индексом i^*, ведущее к вершине $\boldsymbol{v}^*$ с наибольшим значением целевой функции среди всех ребер с индексами из $\mathcal{V}_t$. Прокомментируем шаги в теле этого цикла. Шаг 11 с помощью функции DirectionVector вычисляет направляющий вектор $\boldsymbol{d}$ текущей реберной прямой L_i в направлении увеличения целевой функции. Псевдокод функции DirectionVector представлен ниже в виде алгоритма 3. Оператор **if** на шаге 12 проверяет выполнение условия $\boldsymbol{d} = \boldsymbol{0}$. В этом случае на шаге 13 выполняется оператор **continue**, осуществляющий досрочный переход к следующей итерации цикла **for all**. В противном случае на шаге 15 с помощью функции BoundingHyperplaneIndex вычисляется индекс j гиперплоскости H_j, ограничивающей текущую реберную прямую L_i. Псевдокод функции BoundingHyperplaneIndex представлен ниже в виде алгоритма 4. Шаг 16 вычисляет конечную вершину $\boldsymbol{v}_{nex}$ текущего ребра в соответствии с формулой (25). Если в операторе **if** на шаге 17 значение целевой

функции в вершине $\boldsymbol{v}_{nex}$ оказывается больше значения целевой функции в вершине $\boldsymbol{v}^*$, то вершине $\boldsymbol{v}^*$ присваиваются координаты вершины $\boldsymbol{v}_{nex}$ (шаг 18), индекс i текущей реберной прямой сохраняется в переменной i^* (шаг 19), а индекс соответствующей ограничивающей гиперплоскости H_j сохраняется в переменной j^* (шаг 20). Шаг 21 закрывает оператор **if**. Шаг 22 заканчивает цикл **for all**.

Алгоритм 1 Алгоритм HAlEM

1: **input** n, m, k, A, $\boldsymbol{b}$, $\boldsymbol{c}$, $\boldsymbol{v}_0$
2: $\bar{I} := \{1, \ldots, k\}$
3: $\hat{I} := \{k+1, \ldots, m\}$
4: $\mathcal{V}_0 := \text{BasisIndex}(\boldsymbol{v}_0)$ // см. алгоритм 2
5: $t := 0$
6: $exit := \textbf{false}$
7: **repeat**
8: **loop**
9: $\boldsymbol{v}^* := \boldsymbol{v}_t$
10: **for all** $i \in \mathcal{V}_t$ **do**
11: $\boldsymbol{d} := \text{DirectionVector}(\boldsymbol{v}_t, \mathcal{V}_t \setminus \{i\})$ // см. алгоритм 3
12: **if** $\boldsymbol{d} = \boldsymbol{0}$ **then**
13: **continue**
14: **end if**
15: $j := \text{BoundingHyperplaneIndex}(\boldsymbol{v}_t, \boldsymbol{d})$ // см. алгоритм 4
16: $\boldsymbol{v}_{nex} := \boldsymbol{v}_t + \frac{b_j - \langle \boldsymbol{a}_j, \boldsymbol{v}_t \rangle}{\langle \boldsymbol{a}_j, \boldsymbol{d} \rangle} \boldsymbol{d}$
17: **if** $\langle \boldsymbol{c}, \boldsymbol{v}_{nex} \rangle > \langle \boldsymbol{c}, \boldsymbol{v}^* \rangle$ **then**
18: $\boldsymbol{v}^* := \boldsymbol{v}_{nex}$
19: $i^* := i$
20: $j^* := j$
21: **end if**
22: **end for**
23: **if** $\boldsymbol{v}^* \neq \boldsymbol{v}_t$ **then**
24: $\boldsymbol{v}_{t+1} := \boldsymbol{v}^*$
25: $\mathcal{V}_{t+1} := (\mathcal{V}_t \setminus \{i^*\}) \cup \{j^*\}$
26: $t := t + 1$
27: **break**
28: **end if**
29: $\mathcal{V}'_t := \text{ScrollBasisIndex}(\boldsymbol{v}_t, \mathcal{V}_t)$ // см. алгоритм 5
30: **if** $\mathcal{V}'_t = \mathcal{V}_t$ **then**
31: $exit := \textbf{true}$
32: **break**
33: **end if**
34: $\mathcal{V}_t := \mathcal{V}'_t$
35: **end loop**
36: **until** *exit*
37: **output** $\boldsymbol{v}_t$
38: **stop**

Если в операторе **if** на шаге 23 удалось получить вершину $\boldsymbol{v}^*$ с координатами, отличными от координат текущей вершины $\boldsymbol{v}_t$, то выполняются следующие действия. На шаге 24 эта вершина становится следующим приближением $\boldsymbol{v}_{t+1}$. На шаге 25 для новой вершины $\boldsymbol{v}_{t+1}$ вычисляется базисный индекс $\mathcal{V}_{t+1}$ по формуле (30). Шаг 26 увеличивает на единицу счетчик итераций t. На шаге 27 оператор **break** осуществляет выход из цикла **loop**, после чего выполняется следующая итерация основного цикла **repeat**/**until**.

Если условие оператора **if** на шаге 23 не выполняется, происходит переход к шагу 29. Эта ситуация возникает, когда базисный индекс $\mathcal{V}_t$ оказался «тупиковым»: в нем отсутствуют ребра, ведущие к увеличению целевой функции. В этом случае на шаге 29 выполняется «прокручивание» базисного индекса вершины $\boldsymbol{v}_t$ с помощью функции ScrollBasisIndex, псевдокод которой представлен в виде алгоритма 5. В результате получается обновленный базисный индекс $\mathcal{V}'_t \neq \mathcal{V}_t$. В этом случае оператор **if** (шаги 30–33) пропускается, и выполняется шаг 34, который заменяет предыдущий базисный индекс на новый. После этого повторяется итерация цикла **loop** для текущей вершины $\boldsymbol{v}_t$, но уже с обновленным базисным индексом.

Если обновить базисный индекс невозможно, функция ScrollBasisIndex возвращает исходный базисный индекс. Это означает, что оптимальная вершина уже достигнута. В этом случае выполняется условие в операторе **if** на шаге 30, что приводит к выполнению шага 31, который устанавливает логическую переменную *exit* в значение «истина». После этого оператор **break** на 32 шаге производит выход из цикла **loop**. Поскольку *exit* = **true**, оператор **until** на шаге 36 завершает выполнение основного цикла **repeat**/**until**. Шаг 37 выводит координаты оптимальной вершины. Шаг 38 завершает работу алгоритма HAlEM.

Псевдокод функции BasisIndex, используемой на шаге 4 алгоритма 1 для построения базисного индекса начальной вершины, представлен в виде алгоритма 2. В силу предполо-

Алгоритм 2 Построение базисного индекса для начальной вершины

```
 1: function BasisIndex(v)
 2:     V := ∅
 3:     for j = k + 1, ..., n do                    // k — число уравнений в системе (1)
 4:         for all i ∈ Î do
 5:             if ⟨a_i, v⟩ = b_i then
 6:                 if rank(A_{Ī∪V∪{i}}) = j then
 7:                     V := V ∪ {i}
 8:                     break
 9:                 end if
10:             end if
11:         end for
12:     end for
13:     return V
14: end function
```

жения (3) система ограничений (1) не содержит избыточных уравнений[4]. Поскольку любая точка допустимого многогранника M должна удовлетворять всем уравнениям из системы (1), любой матричный базис в любой вершине $\boldsymbol{v}$ допустимого многогранника M должен

[4]Если система ограничений содержит избыточные уравнения, их можно элиминировать, включая в первоначально пустую матрицу A строки коэффициентов уравнений, увеличивающих ее ранг, и игнорируя строки коэффициентов уравнений, при добавлении которых ранг матрицы A остается неизменным.

включать в себя коэффициенты всех уравнений. Для того, чтобы получился полный матричный базис в вершине $\boldsymbol{v}$, к матрице $A_{\bar{I}}$ необходимо добавить $n-k$ строк коэффициентов неравенств, соответствующих ограничивающим гиперплоскостям, проходящим через $\boldsymbol{v}$, так, чтобы ранг получившейся квадратной матрицы был равен размерности пространства n. Алгоритм 2 вычисляет базисный индекс, содержащий индексы соответствующих неравенств. На шаге 2 создается пустой базисный индекс $\mathcal{V}$. Внешний цикл **for** выполняет $n-k$ итераций, добавляя каждый раз в $\mathcal{V}$ новый индекс i из $\hat{I}$ так, чтобы ранг матрицы $A_{\bar{I}\cup\mathcal{V}\cup\{i\}}$ оставался полным. Это обеспечивается путем выполнения вложенного цикла **for all** (шаги 4–9), который для каждого i-того неравенства на шаге 5 проверяет условие $\langle \boldsymbol{a}_i, \boldsymbol{v}\rangle = b_i$. Если это условие выполняется, то граничная гиперплоскость, соответствующая i-тому неравенству, проходит через вершину $\boldsymbol{v}$. В этом случае на шаге 6 проверяется условие $\operatorname{rank}(A_{\bar{I}\cup\mathcal{V}\cup\{i\}}) = j$. Выполнение этого условия означает, что матрица $A_{\bar{I}\cup\mathcal{V}\cup\{i\}}$ имеет полностроч ный ранг. В этом случае шаг 7 добавляет индекс i в $\mathcal{V}$, после чего оператор **break** на шаге 8 выполняет досрочный выход из вложенного цикла **for all**. Процесс продолжается, пока не будут выполнены все итерации внешнего цикла **for**. В итоге получается полный базисный индекс $\mathcal{V}$, который на шаге 13 возвращается в качестве результата функции BasisIndex.

Псевдокод функции DirectionVector, используемой на шаге 11 алгоритма 1 для вычисления направляющего вектора реберной прямой с реберным индексом $\mathcal{E}$, представлен в виде алгоритма 3. Реализация этого алгоритма основывается на предложенном нами вычисли-

Алгоритм 3 Вычисление направляющего вектора

```
1:  function DirectionVector(v, E)
2:      e_c := c / ||c||
3:      z := v + δ e_c
4:      w := π_{Ī∪E}(z)
5:      if ||w − v|| = 0 then
6:          return 0
7:      end if
8:      d := w − v
9:      e_d := d / ||d||
10:     z' := v + δ e_d
11:     w' := π_{Ī∪E}(z')
12:     d' := w' − v
13:     return d'
14: end function
```

тельном методе, названным двух-факторной проекцией. Суть этого метода заключается в следующем. В контексте задачи ЛП (1) с помощью реберного индекса $\mathcal{E}$ задана реберная прямая $L_{\mathcal{E}}$, проходящая через вершину $\boldsymbol{v} \in M$. Необходимо найти направляющий вектор этой прямой, указывающий в сторону увеличения значения линейной целевой функции с градиентом $\boldsymbol{c}$. Для этого на шаге 2 алгоритма 3 вычисляется единичный вектор $\boldsymbol{e_c} = \boldsymbol{c}/\|\boldsymbol{c}\|$, сонаправленный с вектором $\boldsymbol{c}$. Шаг 3 находит точку $\boldsymbol{z} = \boldsymbol{v} + \delta\boldsymbol{e_c}$. Здесь δ — положительная вещественная константа, являющаяся параметром алгоритма. Шаг 4 с помощью формулы (11) вычисляет точку $\boldsymbol{w} = \boldsymbol{\pi}_{\bar{I}\cup\mathcal{E}}(\boldsymbol{z})$, являющуюся ортогональной проекцией точки $\boldsymbol{z}$ на прямую $L_{\mathcal{E}}$ (см. рис. 3). Ели на шаге 5 выясняется, что расстояние между точками $\boldsymbol{w}$ и $\boldsymbol{v}$ равно нулю, это означает, что градиент $\boldsymbol{c}$ целевой функции перпендикулярен прямой $L_{\mathcal{E}}$,

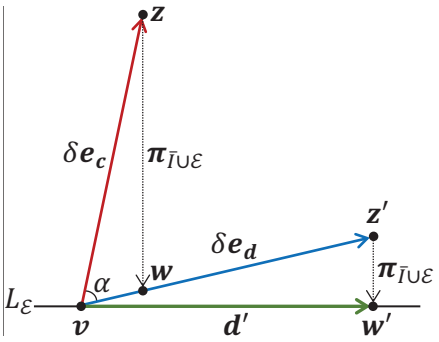

Рис. 3. Двух-факторная проекция

и она не может вести к увеличению целевой функции. В этом случае на шаге 6 функция DirectionVector в качестве результата возвращает нулевой вектор. В противном случае на шаге 8 получаем ненулевой вектор $\boldsymbol{d} = \boldsymbol{w} - \boldsymbol{v}$, направленный в сторону увеличения целевой функции. Точность вычисления координат точки $\boldsymbol{w}$ критически зависит от двух факторов: величины δ и угла α между векторами $\boldsymbol{c}$ и $\boldsymbol{d}$. Вычислительные эксперименты показали, что для достижения высокой точности на реальных задачах ЛП значение параметра δ должно находится в следующих пределах: $10^5 \leqslant \delta \leqslant 10^9$. Но этого оказывается недостаточно. Если величина угла α оказывается близкой к 90°, то наблюдается катастрофическая потеря точности для любых значений параметра δ. На рис. 3 такая потеря точности выражается в том, что точка $\boldsymbol{w}$ оказывается на некотором расстоянии от прямой $L_{\mathcal{E}}$. Для нейтрализации этого эффекта выполняется вторая фаза проектирования. На шаге 9 вычисляется единичный вектор $\boldsymbol{e_d} = \boldsymbol{d}/\|\boldsymbol{d}\|$, сонаправленный с вектором $\boldsymbol{d}$. Шаг 10 находит точку $\boldsymbol{z'} = \boldsymbol{v} + \delta\boldsymbol{e_d}$. Шаг 11 с помощью формулы (11) вычисляет точку $\boldsymbol{w'} = \boldsymbol{\pi}_{\bar{I}\cup\mathcal{E}}(\boldsymbol{z'})$, являющуюся ортогональной проекцией точки $\boldsymbol{z'}$ на прямую $L_{\mathcal{E}}$. Шаг 12 с высокой точностью вычисляет направляющий вектор $\boldsymbol{d'} = \boldsymbol{w'} - \boldsymbol{v}$ реберной прямой $L_{\mathcal{E}}$, указывающий в сторону увеличения целевой функции. Шаг 13 возвращает вектор $\boldsymbol{d'}$ в качестве результата функции DirectionVector.

Псевдокод функции BoundingHyperplaneIndex, используемой на шаге 15 алгоритма 1 для вычисления индекса j гиперплоскости H_j, ограничивающей реберную прямую L_i, представлен в виде алгоритма 4. Здесь $\boldsymbol{v}$ — вершина, через которую проходит реберная пря-

Алгоритм 4 Вычисление индекса ограничивающей гиперплоскости

```
1:  function BoundingHyperplaneIndex(v, d)
2:      γ_min := +∞                                  // см. формулы (17) и (18)
3:      for all j ∈ Î do
4:          if ⟨a_j, d⟩ > 0 then
5:              γ := (b_{j*} − ⟨a_{j*}, v⟩) / ⟨a_{j*}, d⟩
6:              if γ < γ_min then
7:                  γ_min := γ
8:                  j* := j
9:              end if
10:         end if
11:     end for
12:     return j*
13: end function
```

мая L_i, $\boldsymbol{d}$ — ее направляющий вектор. Алгоритм 4 в точности реализует вычисления по формуле (29) и не нуждается в пояснении. Заметим только, что ограничивающая гиперплоскость всегда существует в силу ограниченности допустимого многогранника M.

Следующий алгоритм реализует функцию «прокручивания» базисного индекса.

Алгоритм 5 Прокручивание базисного индекса для выхода из тупика

Require: $\boldsymbol{g} = (g_1, \ldots, g_n)$; $\boldsymbol{y} = (y_1, \ldots, y_n)$; $\boldsymbol{u} = (u_1, \ldots, u_m)$; $\mathcal{V} = [v_1, \ldots, v_{n-k}]$

function ScrollBasisIndex($\boldsymbol{v}, \mathcal{V}$)
  repeat
    $\boldsymbol{g} := \boldsymbol{c} A^{-1}_{\bar{I} \cup \mathcal{V}}$
    $\boldsymbol{u} := \boldsymbol{0}$
    for $i = 1, \ldots, n-k$ **do**
      $u_{v_i} := g_{k+i}$
    end for
    $i^* := 0$
    for $i = 1, \ldots, m$ **do**
      if $u_i < 0$ **then**
        $i^* := i$
        break
      end if
    end for
    if $i^* = 0$ **then**
      return $\mathcal{V}$
    end if
    for $j = 1, \ldots, n-k$ **do**
      if $v_j = i^*$ **then**
        for $i = 1, \ldots, n$ **do**
          $y_i := -A^{-1}_{\bar{I} \cup \mathcal{V}}[i, k+j]$
        end for
      end if
    end for
    $\lambda_{min} := +\infty$
    for all $j \in \hat{I}$ **do**
      if $\langle \boldsymbol{a}_j, \boldsymbol{y} \rangle > 0$ **then**
        $\lambda := \frac{b_j - \langle \boldsymbol{a}_j, \boldsymbol{v} \rangle}{\langle \boldsymbol{a}_j, \boldsymbol{y} \rangle}$
        if $\lambda < \lambda_{min}$ **then**
          $\lambda_{min} := \lambda$
          $j^* := j$
        end if
      end if
    end for
    $\mathcal{V} := (\mathcal{V} \setminus \{i^*\}) \cup \{j^*\}$
  until $\lambda_{min} > 0$
  return $\mathcal{V}$
end function

В некоторых случаях алгоритм HAlEM, находясь в вершине $\boldsymbol{v}_t$, может попасть на шаге 29 в «тупиковую» ситуацию, когда в базисном индексе $\mathcal{V}_t$ отсутствуют ребра, ведущие к увеличению целевой функции. С помощью комбинаторного перебора всевозможных сочетаний ограничивающих гиперплоскостей, проходящих через $\boldsymbol{v}_t$, всегда можно найти ребро, ведущее к увеличению целевой функции (если такое ребро не существует, мы уже находимся в оптимальной вершине). Этот метод был нами реализован в алгоритме AlEM [16]. Однако алгоритм AlEM имеет неприемлемо высокую экспоненциальную вычислительную сложность для многих практических задач ЛП. Вместо этого в алгоритме HAlEM на шаге 29 мы используем функцию ScrollBasisIndex, которая «прокручивает» в вершине $\boldsymbol{v}_t$ различные варианты базисного индекса $\mathcal{V}_t$ до тех пор, пока не будет найдена комбинация, в которой есть ребро, ведущее к увеличению целевой функции. Функция ScrollBasisIndex, представленная в виде алгоритма 5, для этой цели использует подход, применяемый в симплекс-методе, и детально описанный в разделе 11.1 монографии [19]. При написании алгоритма 5, реализующего функцию ScrollBasisIndex, мы сохранили обозначения, использованные в [19], за исключением следующих: вершине x_0 соответствует $\boldsymbol{v}_t$; матрице A_0 соответствует матрица $A_{\bar{I}\cup\mathcal{V}}$; столбцу b_0 соответствует $\boldsymbol{b}_{\bar{I}\cup\mathcal{V}}$. Для своей работы алгоритм 5 использует три вспомогательных вектора: $\boldsymbol{g}, \boldsymbol{y} \in \mathbb{R}^n$ и $\boldsymbol{u} \in \mathbb{R}^m$. Основной цикл **repeat/until** (шаги 2–32) осуществляет «прокрутку» базисных индексов, пока не будет найден вариант с ребром, ведущим к увеличению целевой функции. Шаг 3 вычисляет вспомогательный вектор $\boldsymbol{g}$. Шаги 4–7 заполняют координаты вектора $\boldsymbol{u}$ из двойственной задачи. В отличии от [19], мы заполняем нулями координаты $\boldsymbol{u}$, соответствующие уравнениям. На шагах 8–14 находим индекс i^*, соответствующий первой отрицательной координате вектора $\boldsymbol{u}$. Если в $\boldsymbol{u}$ таких координат нет, то мы находимся в оптимальной вершине. В этом случае функция ScrollBasisIndex на шаге 16 возвращает исходный базисный индекс. В противном случае на шагах 18–24 вычисляются координаты направляющего вектора $\boldsymbol{y}$ для i^*-го ребра. Шаги 25–34 вычисляют индекс j^* ограничивающей гиперплоскости подобно тому, как это делалось в алгоритме 4. Шаг 35 обновляет («прокручивает») базисный индекс $\mathcal{V}$, удаляя из него i^* и добавляя j^*. Процесс продолжается до тех пор, пока не будет получен вариант с $\lambda_{min} > 0$, означающий, что j^*-тое ребро ведет к увеличению целевой функции.

Следует отметить, что в редких случаях могут встретиться вырожденные задачи ЛП, для которых через определенное число итераций цикла **repeat/until** мы можем получить базисный индекс, который уже встречался ранее. В этом случае произойдет зацикливание алгоритма 5. Это характерно и для оригинального симплекс-метода, разработанного Данцигом. Для таких случаев предложены различные техники [11], позволяющие выйти из зацикливания. Однако обсуждение этих вопросов выходит за рамки настоящей статьи. Для невырожденных задач ЛП алгоритм HAlEM за конечное число шагов гарантированно сходится к оптимальной вершине.

3. Параллельная версия алгоритма HAlEM

Алгоритм 1 перебирает в цикле **for all** (шаги 10–22) $n-k$ ребер базисного индекса $\mathcal{V}_t$ с целью нахождения ребра, ведущего к вершине с наибольшим значением целевой функции. Этот цикл поддается эффективному распараллеливанию, что мы реализовали в параллельной версии алгоритма HAlEM.

Параллельная версия алгоритма HAlEM основана на модели параллельных вычислений BSF [20], ориентированной на кластерные вычислительные системы. Модель BSF использу-

ет схему распараллеливания «мастер–рабочие» и требует представление алгоритма в виде операций над списками с использованием функций высшего порядка Map и Reduce.

Функция высшего порядка Map(F, *MapList*) выполняет функцию F для всех элементов списка *MapList*, в результате чего формируется список *ReduceList*. Функция высшего порядка Reduce($\oplus$, *ReduceList*) попарно выполняет ассоциативную бинарную операцию $\oplus$ для всех элементов списка *ReduceList*, в результате чего получается один элемент.

В данном случае список *MapList* имеет длину $n-k$ и включает в себя все элементы базисного индекса $\mathcal{V}_t$, взятые в определенном порядке:

$$MapList = [i_1, \ldots, i_{n-k}], \tag{33}$$

где $\{i_1, \ldots, i_{n-k}\} = \mathcal{V}_t$. Семантика функции F_t определяется алгоритмом 6, шаги которого соответствуют шагам 11– 16 алгоритма 1.

Алгоритм 6 Функция F_t

1: **function** $\mathrm{F}_t(i)$
2: $\quad \boldsymbol{d} := \mathrm{DirectionVector}(\boldsymbol{v}_t, \mathcal{V}_t \setminus \{i\})$ // см. алгоритм 3
3: $\quad$ **if** $\boldsymbol{d} = \mathbf{0}$ **then**
4: $\quad\quad$ **return** $(\boldsymbol{v}_t, i, 0)$
5: $\quad$ **end if**
6: $\quad j := \mathrm{BoundingHyperplaneIndex}(\boldsymbol{v}_t, \boldsymbol{d})$ // см. алгоритм 4
7: $\quad \boldsymbol{v}^* := \boldsymbol{v}_t + \frac{b_j - \langle \boldsymbol{a}_j, \boldsymbol{v}_t \rangle}{\langle \boldsymbol{a}_j, \boldsymbol{d} \rangle} \boldsymbol{d}$
8: $\quad$ **return** $(\boldsymbol{v}^*, i, j)$
9: **end function**

Список *ReduceList*, получаемый в результате вызова функции высшего порядка Map(F_t, *MapList*), будет выглядеть следующим образом:

$$ReduceList = \left[(\boldsymbol{v}_1^*, i_1^*, j_1^*), \ldots, (\boldsymbol{v}_{n-k}^*, i_{n-k}^*, j_{n-k}^*)\right],$$

где для любого $l \in \{1, \ldots, n-k\}$ точка $\boldsymbol{v}_l^*$ является смежной вершиной по отношению к вершине $\boldsymbol{v}_t$, i_l^* — индекс соответствующего ребра, j_l^* — индекс гиперплоскости, ограничивающей соответствующую реберную прямую, за исключением случая, когда $j_l^* = 0$. В этом случае $\boldsymbol{v}_l^* = \boldsymbol{v}_t$.

Бинарная операция $\oplus$ над элементами списка *ReduceList* определяется следующей формулой:

$$\left(\boldsymbol{v}', i', j'\right) \oplus \left(\boldsymbol{v}'', i'', j''\right) = \begin{cases} \left(\boldsymbol{v}', i', j'\right), & \textbf{if } \langle \boldsymbol{c}, \boldsymbol{v}' \rangle > \langle \boldsymbol{c}, \boldsymbol{v}'' \rangle, \\ \left(\boldsymbol{v}'', i'', j''\right), & \textbf{if } \langle \boldsymbol{c}, \boldsymbol{v}' \rangle \leqslant \langle \boldsymbol{c}, \boldsymbol{v}'' \rangle, \end{cases} \tag{34}$$

семантика которой соответствует шагам 17–21 алгоритма 1.

Псевдокод параллельной версии алгоритма HAlEM представлен в виде алгоритма 7. Распараллеливанию подвергается цикл **for all** (шаги 10–22) последовательного алгоритма 1. Параллельные вычисления организуются по схеме «мастер–рабочие» и включают в себя $L+1$ процесс: один процесс–мастер и L процессов–рабочих, где $L \leqslant n-k$. Для простоты мы будем предполагать, что $n-k$ кратно L.

Процесс–мастер (далее — просто «мастер») выполняет последовательную часть алгоритма 7 и организует выполнение параллельной части процессами–рабочими (далее — про-

Алгоритм 7 Параллельная версия алгоритма HAlEM

	мастер		***l*-тый рабочий** $(l = 1, \ldots, L)$
1:	**input** n, m, k, A, $\boldsymbol{b}$, $\boldsymbol{c}$, $\boldsymbol{v}_0$	1:	**input** n, m, k, A, $\boldsymbol{b}$, $\boldsymbol{c}$
2:	$\bar{I} := \{1, \ldots, k\}$	2:	$L :=$ **NumberOfWorkers**
3:	$\hat{I} := \{k+1, \ldots, m\}$	3:	$l :=$ **MyNumber**
4:	$\mathcal{V}_0 := \mathrm{BasisIndex}(\boldsymbol{v}_0)$	4:	
5:	$t := 0$	5:	
6:	$exit :=$ **false**	6:	
7:	**repeat**	7:	**repeat**
8:	**loop**	8:	
9:	**Broadcast**$(\boldsymbol{v}_t, \mathcal{V}_t)$	9:	**Recv**$(\boldsymbol{v}_t, \mathcal{V}_t)$ // $\mathcal{V}_t = \{i_1, \ldots, i_{n-k}\}$
10:		10:	$K := \frac{n-k}{L}$
11:		11:	$MapList_l := \left[i_{1+(l-1)K}, \ldots, i_{lK}\right]$
12:		12:	$ReduceList_l := \mathrm{Map}(\mathrm{F}_t, MapList_l)$
13:		13:	$(\boldsymbol{v}^*, i^*, j^*) := \mathrm{Reduce}(\oplus, ReduceList_l)$
14:	**Gather**$(ReduceList^*)$	14:	**Send**$(\boldsymbol{v}^*, i^*, j^*)$
15:	$(\boldsymbol{v}^*, i^*, j^*) := \mathrm{Reduce}(\oplus, ReduceList^*)$	15:	
16:	**if** $\boldsymbol{v}^* \neq \boldsymbol{v}_t$ **then**	16:	
17:	$\boldsymbol{v}_{t+1} := \boldsymbol{v}^*$	17:	
18:	$\mathcal{V}_{t+1} := (\mathcal{V}_t \setminus \{i^*\}) \cup \{j^*\}$	18:	
19:	$t := t + 1$	19:	
20:	**break**	20:	
21:	**end if**	21:	
22:	$\mathcal{V}'_t := \mathrm{ScrollBasisIndex}(\boldsymbol{v}_t, \mathcal{V}_t)$	22:	
23:	**if** $\mathcal{V}'_t = \mathcal{V}_t$ **then**	23:	
24:	$exit :=$ **true**	24:	
25:	**break**	25:	
26:	**end if**	26:	
27:	$\mathcal{V}_t := \mathcal{V}'_t$	27:	
28:	**end loop**	28:	
29:	**Broadcast**$(exit)$	29:	**Recv**$(exit)$
30:	**until** $exit$	30:	**until** $exit$
31:	**output** $\boldsymbol{v}_t$	31:	
32:	**stop**	32:	**stop**

сто «рабочие»). Шаги 1–8 мастера соответствуют шагам 1–8 последовательного алгоритма 1. На шаге 9 мастер с помощью системной функции **Broadcast** рассылает всем рабочим координаты текущей вершины $\boldsymbol{v}_t$ и соответствующий базисный индекс $\mathcal{V}_t$. На шаге 14 мастер с помощью системной функции **Gather** собирает результаты всех рабочих, представленных в виде троек вида $(\boldsymbol{v}^*, i^*, j^*)$, и организует их в единый список $ReduceList^*$. На шаге 15 мастер с помощью функции высшего порядка Reduce редуцирует список $ReduceList^*$ к одной тройке $(\boldsymbol{v}^*, i^*, j^*)$ путем попарного применения бинарной операции $\oplus$ по формуле (34). Далее мастер выполняет шаги 16–28, соответствующие шагам 23–35 последовательного ал-

горитма 1. На шаге 29 мастер с помощью системной функции **Broadcast** рассылает всем рабочим значение логической переменной *exit*, в зависимости от которого рабочие продолжают или заканчивают свою работу. Финальные шаги 30–32 мастера соответствуют финальным шагам 36–38 последовательного алгоритма 1.

Все рабочие выполняют одну и ту же последовательность шагов, но над разными данными. На шаге 1 каждый рабочий вводит исходные данные задачи ЛП. На шаге 2 с помощью системной функции **NumberOfWorkers** рабочий получает общее число рабочих L, задействованных в вычислениях. На шаге 3 с помощью системной функции **MyNumber** рабочий получает свой номер l. Далее рабочий входит в вычислительный цикл **repeat**/**until** (шаги 7–30). На шаге 9 рабочий с помощью системной функции **Recv** ожидает получения от мастера координат текущей вершины $\boldsymbol{v}_t$ и элементов текущего базисного индекса $\mathcal{V}_t$. На шаге 10 рабочий вычисляет длину K той части списка *MapList*, которую ему необходимо обработать. На шаге 11 рабочий строит свою часть списка $MapList_l$. На шаге 12 с помощью функции высшего порядка Map рабочий вычисляет список $ReduceList_l$. На шаге 13 рабочий с помощью функции высшего порядка Reduce редуцирует список $ReduceList_l$ к одной тройке $(\boldsymbol{v}^*, i^*, j^*)$ путем попарного применения бинарной операции $\oplus$ по формуле (34). На шаге 14 с помощью системной функции **Send** рабочий отсылает полученную тройку мастеру. После этого рабочий на шаге 29 с помощью системной функции **Recv** ожидает получения от мастера значения логической переменной *exit*. Если на шаге 30 переменная *exit* принимает значение **false**, рабочий переходит к следующей итерации вычислительного цикла **repeat**/**until**. В противном случает рабочий заканчивает свою работу.

Отметим, что с помощью системных функций **Recv** и **Gather** осуществляется неявная синхронизация параллельных процессов.

4. Реализация и вычислительные эксперименты

Мы реализовали параллельную версию алгоритма HAlEM на языке C++ с использованием программного BSF-каркаса [21], базирующегося на модели параллельных вычислений BSF [20]. BSF-каркас инкапсулирует все аспекты, связанные с распараллеливанием программы на основе библиотеки MPI. Исходные коды параллельной реализации свободно доступны в репозитории GitVerse по адресу `https://gitverse.ru/sokolinsky/HAlEM`. С использованием этой реализации были проведены вычислительные эксперименты по исследованию масштабируемости параллельной версии алгоритма HAlEM на различных задачах ЛП. Все эксперименты проводились на суперкомпьютере «Торнадо ЮУрГУ» [22], характеристики которого представлены в табл. 1. Для сборки программы использовался

Таблица 1. Характеристики суперкомпьютера «Торнадо ЮУрГУ»

Параметр	Значение
Количество процессорных узлов	480
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

Таблица 2. Тестирование параллельного алгоритма HAlEM на задачах Netlib-LP

Задача	m	n	d_M	d_M/n	δ	K_{peak}	$K_{\max}$	$T_{K_{\max}}$	T_1	α	ϵ
1	2	3	4	5	6	7	8	9	10	11	12
adlittle	153	97	82	0.85	0	6	5	0.55	2.22	4.04	0.81
afiro	59	32	24	0.75	2.5×10^{-16}	2	2	0.0015	0.0027	1.8	0.90
agg	651	163	127	0.78	0	10	6	2.57	10.40	3.84	0.67
agg2	818	302	242	0.80	7.4×10^{-16}	20	8	93.4	616.2	6.6	0.82
beaconfd	435	262	122	0.47	2.2×10^{-16}	10	6	9.18	31.62	3.44	0.57
blend	157	83	40	0.48	1.2×10^{-16}	3	3	0.35	0.52	1.5	0.51
israel	316	142	142	1	1.2×10^{-15}	11	7	5.46	29.09	5.33	0.76
kb2	93	41	25	0.61	0	2	2	0.022	0.030	1.34	0.67
recipe	366	180	92	0.51	0	7	6	2.07	6.94	3.35	0.56
sc105	207	103	58	0.56	0	4	4	0.13	0.32	2.47	0.62
sc50a	97	48	28	0.58	0	2	2	0.008	0.009	1.2	0.62
sc50b	96	48	28	0.58	2×10^{-16}	2	2	0.005	0.007	1.3	0.65
scagr7	269	140	56	0.4	0	4	4	0.81	2.13	2.6	0.66
share2b	175	79	66	0.83	1.5×10^{-15}	5	3	0.18	0.43	2.34	0.78
stocfor1	228	111	48	0.43	1.1×10^{-14}	4	4	0.18	0.43	2.32	0.58

компилятор g++, распространяемый в составе пакета компиляторов GCC 10, и библиотека Intel MPI 5.0. Компиляция выполнялась с опцией оптимизации O3. Задачи 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.

В первой серии экспериментов мы исследовали ускорение и эффективность распараллеливания алгоритма HAlEM на эталонных задачах ЛП из репозитория Netlib-LP [23], доступного по адресу `https://netlib.org/lp/data`. В качестве начальной вершины $\boldsymbol{v}_0$ всегда выбиралась точка $\mathbf{0}$, если она являлась вершиной допустимого многогранника M. В противном случае начальная вершина вычислялась с помощью программы VeRSAl (Vertex Retrieve by Simplex Algorithm), написанной на языке C++, исходные тексты которой свободно доступны в репозитории GitVerse по адресу `https://gitverse.ru/sokolinsky/VeRSAl`. Программа VeRSAl строит на основе исходной задачи ЛП расширенную задачу ЛП в соответствии с методом, описанными в [19] (см. «Общий случай», стр. 199). Точка $\mathbf{0}$ по построению является вершиной допустимого многогранника для расширенной задачи ЛП. Программа VeRSAl решает эту задачу с помощью стандартного симплекс-метода, в результате чего получается вершина допустимого многогранника для исходной задачи ЛП.

Результаты экспериментов с эталонными задачами из репозитория Netlib-LP представлены в табл. 2. В столбце 1 перечислены имена задач из репозитория Netlib-LP, использованные для тестирования параллельного алгоритма HAlEM. Файлы со спецификациями этих задач в формате MPS [24] доступны в репозитории GitVerse по адресу `https://gitverse.ru/sokolinsky/Set-of-LP-Problems/content/master/NetLib-LP`. В этом же репозитории в формате MatrixMarket [25] сохранены координаты начальных вершин для всех указанных задач. В столбце 2 указано количество ограничений m для соответствующей задачи ЛП, включая неравенства, уравнения (если они присутствуют) и ограничения вида (2). В столбце 3 указано количество переменных n (размерность пространства решений). Столбец 4 содержит размерность d_M допустимого многогранника M, совпадающую с размерностью его аффинной оболочки: $d_M = \dim(M) = \dim(\text{aff}(M))$. Если

система ограничений не содержит избыточных уравнений и неявных равенств[5], то размерность допустимого многогранника M может быть вычислена по следующей формуле:

$$\dim(M) = n - k,$$

где k — количество уравнений в системе ограничений. В соответствии с этим количество уравнений k в системе ограничений задачи ЛП из табл. 2 может быть вычислено по формуле

$$k = n - d_M. \tag{35}$$

Столбец 5 содержит отношение размерности d_M допустимого многогранника к размерности n пространства решений. В столбце 6 указана относительная погрешность максимума целевой функции, вычисленного с помощью алгоритма HAlEM, в сравнении с «точным» значением, приведенным в работе Коха [26]:

$$\delta = \left| \frac{F_{Koch} - F_{HAlEM}}{F_{Koch}} \right|.$$

Здесь F_{HAlEM} — значение, вычисленное алгоритмом HAlEM, F_{Koch} — значение, указанное в работе [26]. Практически во всех случаях относительная погрешность оказалась в пределах машинного нуля (машинного эпсилон) [27], равного 2.2×10^{-16} для типа double (64 бита). Столь высокая точность вычислений была достигнута благодаря использованию метода двух-факторной проекции в алгоритме 3.

В столбце 7 приведена верхняя теоретическая граница K_{peak} масштабируемости алгоритма HAlEM, которая указывает максимальное число процессорных узлов, доступных для распараллеливания при использовании BSF-каркаса. Величина K_{peak} вычисляется следующим образом. Алгоритм 7 организует параллельные вычисления путем деления списка *MapList* на равные части по числу рабочих. Обработка этих подсписков выполняется рабочими на шаге 12 независимо друг от друга. Согласно (33) список *MapList* имеет длину $n - k$. Из (35) следует, что длина списка *MapList* равна d_M. Это означает, что для параллельной обработки списка *MapList* нельзя использовать более d_M рабочих. Учитывая, что при прогоне задачи на каждом процессорном узле запускалось 12 рабочих MPI-процессов, приходим к выводу, что

$$K_{\text{peak}} = \lfloor d_M/12 \rfloor .$$

Столбец 8 содержит реальную границу масштабируемости $K_{\max}$, определенную для каждой задачи ЛП в ходе вычислительных экспериментов. Под реальной границей масштабируемости параллельного алгоритма понимается число процессорных узлов вычислительного кластера, при превышении которого время решения задачи перестает сокращаться. В соответствии с характеристиками вычислительной платформы, использованной при проведении экспериментов (см. табл. 1), число процессорных ядер, задействованных для решения задачи на $K_{\max}$ узлах будет равно $K_{\max} \times 12$. В столбце 9 указано время $T_{K_{\max}}$ решения задачи ЛП на $K_{\max}$ узлах[6]. В столбце 10 указано время T_1 решения этой же задачи на одном процессорном узле. Значения ускорения, достигнутые на границе масштабируемости,

[5]Неравенство $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \leqslant b$ в системе ограничений $A\boldsymbol{x} \leqslant \boldsymbol{b}$ является неявным равенством, если $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b$ для всех $\boldsymbol{x}$, удовлетворяющих $A\boldsymbol{x} \leqslant \boldsymbol{b}$.

[6]Время везде указано в секундах.

представлены в столбце 11. Ускорение вычислялось по формуле

$$\alpha = \frac{T_1}{T_{K_{max}}}.$$

В столбце 12 содержится параллельная эффективность, достигаемая на границе масштабируемости. Параллельная эффективность вычислялась по формуле

$$\epsilon = \frac{T_1}{K_{max} \cdot T_{K_{max}}}.$$

Анализ полученных результатов позволяет сделать несколько выводов. Первый вывод касается границы масштабируемости $K_{\max}$. Для допустимых многогранников малой размерности ($d_M < 70$) теоретическая граница масштабируемости совпадает с реальной:

$$K_{\text{peak}} = K_{\max}.$$

Для допустимых многогранников средней размерности ($70 \leqslant d_M \leqslant 100$) теоретическая и реальная границы масштабируемости различаются не более чем на 1:

$$K_{\text{peak}} - K_{\max} \leqslant 1.$$

В случае допустимых многогранников большой размерности ($d_M > 100$) реальная граница масштабируемости может быть существенно меньше теоретической:

$$K_{\max} \ll K_{\text{peak}}.$$

Это связано с тем, что при использовании большого количества MPI-процессов для решения задачи ЛП методом HAlEM накладные расходы на организацию параллельного выполнения могут полностью нивелировать ускорение, получаемое при добавлении новых процессорных узлов.

Второй вывод связан с корреляцией между величиной d_M/n и эффективностью распараллеливания ϵ. Столбчатая диаграмма, представленная на рис. 4, показывает, что большому значению d_M/n соответствует высокая параллельная эффективность ϵ, и наоборот, низкая параллельная эффективность ϵ характерна для задач ЛП с небольшим значением d_M/n. Из этого ряда выпадает задача israel, которая характерна тем, что у нее единственной в системе ограничений отсутствуют уравнения, то есть размерность пространства решений совпадает с размерностью допустимого многогранника. Также отметим, что во всех случаях параллельная эффективность не опускалась ниже 51%, что является неплохим показателем для численного алгоритма.

Для сравнения разработанной параллельной версии алгоритма HAlEM с симплекс-методом мы использовали программу Simplex, исходные тексты которой на языке C++ свободно доступны по адресу `https://gitverse.ru/sokolinsky/Simplex`. Программа Simplex реализует параллельную версию классического симплекс-метода, описанного в [19] (стр. 197–198). Используя эту реализацию, мы решили все упомянутые задачи из репозитория Netlib-LP на той же вычислительной платформе, на которой исследовали HAlEM. Для всех задач ЛП программа Simplex запускалась на двух процессорных узлах по 12 MPI-процессов на узел. Использование большего количества процессорных узлов приводило к деградации ускорения программы Simplex. Результаты сравнения приведены в табл. 3. Эксперименты показали, что метод HAlEM обладает существенно большим

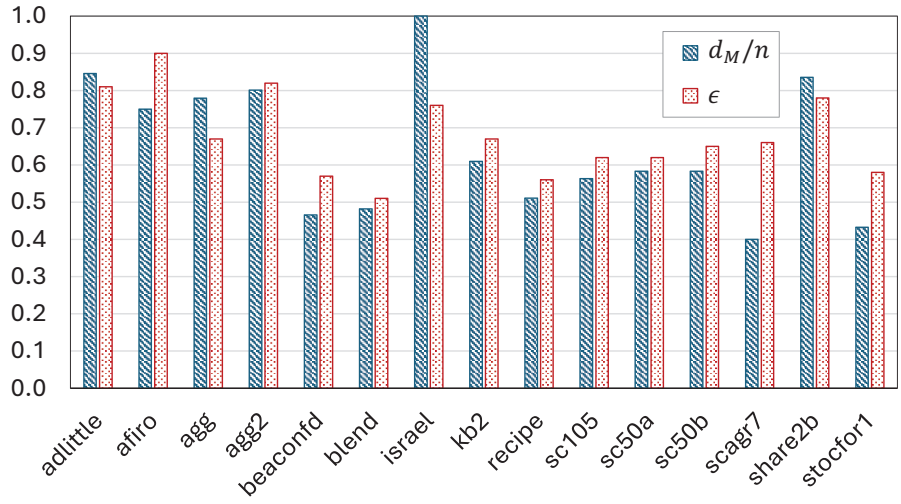

Рис. 4. Корреляция между d_M/n и параллельной эффективностью ϵ

Таблица 3. Сравнение метода HAlEM с симплекс-методом на задачах Netlib-LP

Задача	Процессорных узлов		Число итераций		Время (сек)		Относительная погрешность	
	HAlEM	Simplex	HAlEM	Simplex	HAlEM	Simplex	HAlEM	Simplex
adlittle	5	2	67	59	0.55	0.12	0	3.9×10^{-15}
afiro	2	2	3	4	0.0015	0.0009	2.5×10^{-16}	1.2×10^{-16}
agg	6	2	23	63	2.57	1.06	0	0
agg2	8	2	99	132	93.4	19.3	7.4×10^{-16}	0
beaconfd	6	2	15	23	9.18	2.07	2.2×10^{-16}	2.2×10^{-16}
blend	3	2	35	70	0.35	0.09	1.2×10^{-16}	4.7×10^{-14}
israel	7	2	146	160	5.46	1.26	1.2×10^{-15}	3.9×10^{-15}
kb2	2	2	23	76	0.02	0.02	0	4×10^{-15}
recipe	7	2	12	17	2.07	0.41	0	0
sc105	4	2	13	13	0.13	0.03	0	0
sc50a	2	2	7	7	0.008	0.002	0	0
sc50b	2	2	5	5	0.005	0.002	2×10^{-16}	0
scagr7	4	2	30	30	0.81	0.21	0	1.8×10^{-15}
share2b	3	2	27	33	0.18	0.05	1.5×10^{-15}	2.3×10^{-15}
stocfor1	4	2	12	22	0.18	0.07	1.1×10^{-14}	1.1×10^{-14}

ресурсом параллелизма, чем симплекс-метод. По количеству итераций оба метода имеют близкие показатели, однако симплекс-метод превосходит HAlEM по быстродействию на всех исследованных задачах ЛП. При этом разница в быстродействии находится в пределах одного порядка, что не является катастрофическим. Оба метода демонстрируют высокую точность вычислений на границе машинного нуля для типа double (64 бит). Однако существуют классы задач ЛП, на которых метод HAlEM показывает более высокую эффективность, чем симплекс-метод. В качестве примера можно привести циклические многогранники. Используя пример циклического многогранника $C_4(8)$ из [28] (стр. 29–30), мы сконструировали задачу ЛП zieglerC4_8, исходные данные которой в формате MPS можно найти по адресу `https://gitverse.ru/sokolinsky/Set-of-LP-Problems/content/master/Miscellaneous-LP`. Алгоритм HAlEM решает эту задачу за одну итерацию, в то время как программе Simplex для этого требуется 11 итераций.

В финальной серии экспериментов с параллельной версией алгоритма HAlEM мы исследовали зависимость ускорения и параллельной эффективности от размера решаемой задачи. С этой целью мы сконструировали специальную параметризованную задачу ЛП «гиперкуб с отсеченной вершиной», для которой размерность n пространства решений является параметром. Ограничения этой задачи содержат $2n+1$ неравенств следующего вида:

$$\begin{array}{rcl} x_1 & \leqslant & 200, \\ x_2 & \leqslant & 200, \\ & \vdots & \\ x_n & \leqslant & 200, \\ x_1 + x_2 + \cdots + x_n & \leqslant & 200(n-1)+100, \\ x_1 \geqslant 0, \quad x_2 \geqslant 0, \quad \cdots \quad , x_n \geqslant 0. & & \end{array} \tag{36}$$

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

$$F_{\max}(n) = 100(n^2 + n - 1). \tag{37}$$

Очевидно, что точка $\mathbf{0}$ является вершиной допустимого многогранника системы ограничений (36). Для произвольного n эта задача может быть получена в формате MatrixMarket [25] с помощью генератора FRaGenLP [29], если в качестве количества случайных неравенств задать 0. Исходные коды генератора FRaGenLP на языке C++ свободно доступны по адресу `https://gitverse.ru/sokolinsky/FRaGenLP`. Сгенерированные задачи для различных n доступны по адресу `https://gitverse.ru/sokolinsky/Set-of-LP-Problems/content/master/Tcube` под именами `lp_tcube<s>K<h>`, где в качестве `<s>` и `<h>` указаны две цифры, задающие размерность задачи: $n = \mathsf{sh}00$. В экспериментах мы использовали три задачи: tcube0K2, tcube0K3 и tcube0K4, информация о которых приведена в табл. 4. Семантика столбцов такая же, как в табл. 2, за исключением столбца $F_{\max}$, содержащего

Таблица 4. Тестирование HAlEM на задачах Tcube

Задача	m	n	$F_{\max}$	K_{peak}	$K_{\max}$	$T_{K_{\max}}$	T_1	α	ϵ
tcube0K2	401	200	4019900	16	16	16.1	136	8.5	0.53
tcube0K3	601	300	9029900	24	24	92.5	1126	12.2	0.51
tcube0K4	801	400	16039900	33	24	387	5681	14.7	0.61

максимальное значение целевой функции. Относительная погрешность для всех трех задач была равна нулю. Соответствующие графики ускорения и параллельной эффективности приведены на рис. 5. Отметим, что, как и в предыдущих экспериментах, на каждом процессорном узле запускалось 12 MPI-процессов. В качестве начальной вершины всегда выбиралась точка $\mathbf{0}$. Эксперименты показали, что для задач размерностей 200 и 300 теоретическая граница масштабируемости K_{peak} совпала с реальной $K_{\max}$. Однако при увеличении размерности задачи до 400, начиная с 25 узлов, накладные расходы на распараллеливание стали превалировать над ускорением. В этом случае реальная граница масштабируемости

оказалась заметно меньше теоретической. Отметим, что и в этой серии экспериментов параллельная эффективность на реальной границе масштабируемости не опускалась ниже 51%, что можно считать хорошим результатом.

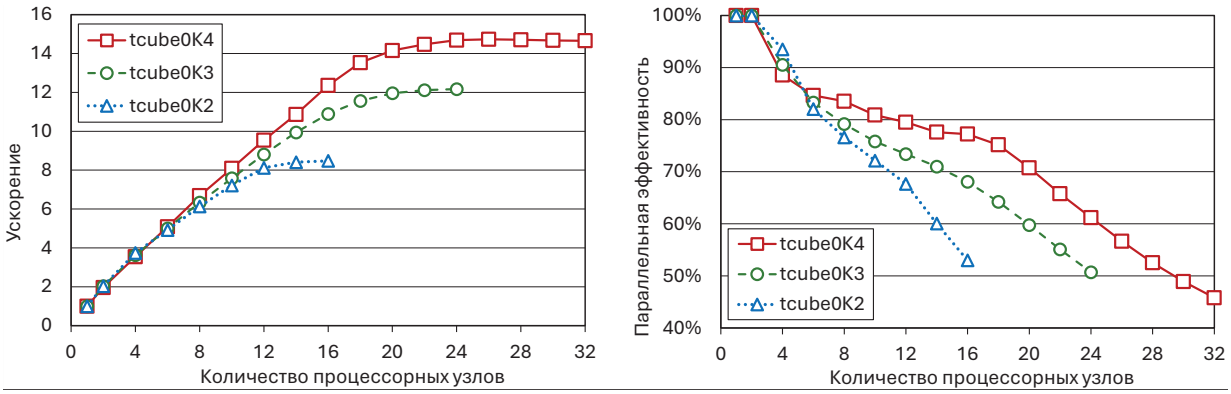

Рис. 5. Ускорение и параллельная эффективность алгоритма HAlEM

Заключение

В данной работе представлен новый проекционный алгоритм HAlEM для решения задач линейного программирования, который объединяет сильные стороны разработанного ранее авторами алгоритма AlEM (проекционный подход) и классического симплекс-метода (обновление матричного базиса). Основная идея HAlEM заключается в построении пути к оптимальной вершине путем последовательного перехода по ребрам допустимого многогранника, причем на каждом шаге для определения направления движения используется эффективная процедура двух-факторной ортогональной проекции, а для переключения на новое ребро применяется механизм ротации базисных индексов, аналогичный симплекс-методу. Такой гибридный подход позволил преодолеть главные недостатки предшествующих алгоритмов: избежать экспоненциального комбинаторного перебора (в отличие от AlFaMove и AlEM) и проблем, связанных с низкой скоростью сходимости фейеровских процессов при вычислении проекций (в отличие от апекс-метода).

Для предложенного алгоритма разработана параллельная версия, ориентированная на кластерные вычислительные системы. Распараллеливание, основанное на BSF-модели и схеме «мастер-рабочие», заключается в одновременной обработке всех ребер текущей вершины, что позволяет значительно сократить время выполнения итерации.

Проведенные вычислительные эксперименты на задачах из репозитория Netlib-LP и на серии специально сконструированных задач подтвердили работоспособность и эффективность алгоритма HAlEM. Основные выводы по результатам исследования можно сформулировать следующим образом. Благодаря использованию метода двух-факторной проекции алгоритм HAlEM демонстрирует высокую точность вычисления оптимального значения целевой функции, находящуюся в пределах машинного нуля для типа double (64 бита). Алгоритм HAlEM имеет ресурс параллелизма, существенно превосходящий симплекс-метод, что позволяет эффективно задействовать значительно большее количество процессорных узлов. Хотя по абсолютному времени счета на малом числе узлов HAlEM уступает симплекс-методу (разница в пределах одного порядка), на некоторых задачах (например, на циклических многогранниках) HAlEM может находить решение за меньшее число итераций.

Исследование масштабируемости показало, что граница эффективного распараллеливания алгоритма HAlEM напрямую зависит от размерности допустимого многогранника. Для задач с небольшой и средней размерностью теоретическая граница масштабируемости совпадает с реальной. При увеличении размерности накладные расходы на коммуникацию могут ограничивать рост ускорения, однако параллельная эффективность на реальной границе масштабируемости остается выше 50% во всех случаях, что является хорошим показателем для численного алгоритма.

Дальнейшие исследования планируется направить на разработку искусственной нейронной сети, которая позволит идентифицировать ребра допустимого многогранника быстрее, чем это делает алгоритм HAlEM с помощью двух-факторной проекции. Подобная модернизация позволит алгоритму HAlEM составить конкуренцию симплекс-методу по быстродействию на реальных задачах.

Обозначения

n	число переменных в системе ограничений (размерность пространства)
m	число ограничений
k	количество уравнений в системе ограничений
$\mathbb{R}^n$	вещественное евклидово пространство размерности n
$\langle\cdot,\cdot\rangle$	скалярное произведение двух векторов
$\|\cdot\|$	евклидова норма
$\bar{I}$	множество индексов ограничений в виде уравнений: $\bar{I} = \{1,\ldots,k\}$
$\hat{I}$	множество индексов ограничений в виде неравенств: $\hat{I} = \{k+1,\ldots,m\}$
A_J	матрица коэффициентов ограничений с индексами из $J \subseteq \{1,\ldots,m\}$
$\boldsymbol{b}_J$	столбец правых частей ограничений с индексами из $J \subseteq \{1,\ldots,m\}$
$\boldsymbol{a}_i$	i-тая строка матрицы A_J $(i \in J)$
P_i	полупространство, определяемое формулой $\langle \boldsymbol{a}_i, \boldsymbol{x}\rangle \leqslant b_i$ $\left(i \in \hat{I}\right)$
H_i	гиперплоскость, определяемая формулой $\langle \boldsymbol{a}_i, \boldsymbol{x}\rangle = b_i$ $(i \in \{1,\ldots,m\})$
M	допустимый многогранник (область допустимых решений)
$\operatorname{rank}(A)$	ранг матрицы A
$\operatorname{aff}(X)$	аффинная оболочка множества X
$\dim(X)$	размерность множества X: $\dim(X) = \dim(\operatorname{aff}(X))$
$\boldsymbol{\pi}_J(\boldsymbol{x})$	ортогональная проекция точки $\boldsymbol{x}$ на подпространство $\bigcap\limits_{i\in J} H_i$

Литература

1. Pan P.-Q. Linear Programming Computation. Berlin, Heidelberg: Springer, 2014. 747 p. DOI: 10.1007/978-3-642-40754-3.
2. Bixby R.E. Solving Real-World Linear Programs: A Decade and More of Progress // Operations Research. 2002. Vol. 50, no. 1. P. 3–15. DOI: 10.1287/opre.50.1.3.17780.
3. Rasmussen S. Production Economics. The Basic Theory of Production Optimisation. 2nd ed. Berlin, Heidelberg: Springer, 2013. XII, 292 p. Springer Texts in Business and Economics. DOI: 10.1007/978-3-642-30200-8.
4. 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. P. 269–292. DOI: 10.1007/978-3-030-28565-4_23.

5. 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.
6. 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.
7. Dantzig G.B., Thapa M.N. Linear Programming 2: Theory and Extensions. Springer Series in Operations Research and Financial Engineering. New York, NY, USA: Springer, 2003. XXVI, 448 p. DOI: 10.1007/b97283.
8. Tolla P. A Survey of Some Linear Programming Methods // Concepts of Combinatorial Optimization / ed. by V.T. Paschos. 2nd ed. Hoboken, NJ, USA: John Wiley, Sons, 2014. Chap. 7. P. 157–188. DOI: 10.1002/9781119005216.ch7.
9. Mamalis B., Pantziou G. Advances in the Parallelization of the Simplex Method // Algorithms, Probability, Networks, and Games. Lecture Notes in Computer Science, vol. 9295 / ed. by C. Zaroliagis, G. Pantziou, S. Kontogiannis. Cham: Springer, 2015. P. 281–307. DOI: 10.1007/978-3-319-24024-4_17.
10. Im H., Wolkowicz H. Revisiting degeneracy, strict feasibility, stability, in linear programming // European Journal of Operational Research. 2023. Vol. 310, no. 2. P. 495–510. DOI: 10.1016/j.ejor.2023.03.021.
11. Gass S.I., Vinjamuri S. Cycling in linear programming problems // Computers and Operations Research. 2004. Vol. 31, no. 2. P. 303–311. DOI: 10.1016/S0305-0548(02)00226-5.
12. Hall J., McKinnon K. The simplest examples where the simplex method cycles and conditions where EXPAND fails to prevent cycling // Mathematical Programming, Series B. 2004. Vol. 100, no. 1. P. 133–150. DOI: 10.1007/s10107-003-0488-1.
13. 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.
14. 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.
15. 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.
16. Жулев А.Э., Соколинский Л.Б. О поиске оптимальной вершины на многограннике допустимых решений задачи линейного программирования // Вычислительные методы и программирование. 2026. Т. 27, № 1. С. 1–18. DOI: 10.26089/NumMet.v27r101.
17. Murty K.G. Computational and Algorithmic Linear Algebra and n-Dimensional Geometry. Singapore: World Scientific Publishing Company, 2014. 480 p. DOI: 10.1142/8261.
18. Стренг Г. Линейная алгебра и ее применения. Москва: Мир, 1980. 455 с.

19. Схрейвер А. Теория линейного и целочисленного программирования: в 2 т. Т. 1. Москва: Мир, 1991. 360 с.

20. Sokolinsky L.B. BSF: A parallel computation model for scalability estimation of iterative numerical algorithms on cluster computing systems // Journal of Parallel and Distributed Computing. 2021. Vol. 149. P. 193–206. DOI: 10.1016/j.jpdc.2020.12.009.

21. Sokolinsky L.B. BSF-skeleton: A Template for Parallelization of Iterative Numerical Algorithms on Cluster Computing Systems // MethodsX. 2021. Vol. 8. Article number 101437. DOI: 10.1016/j.mex.2021.101437.

22. 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. Cham: Springer, 2022. P. 43–55. DOI: 10.1007/978-3-031-11623-0_4.

23. Gay D.M. Electronic mail distribution of linear programming test problems // Mathematical Programming Society COAL Bulletin. 1985. Vol. 13. P. 10–12.

24. Муртаф Б. Современное линейное программирование. М.: Мир, 1984. 224 с.

25. 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.

26. Koch T. The final NETLIB-LP results // Operations Research Letters. 2004. Vol. 32, no. 2. P. 138–142. DOI: 10.1016/S0167-6377(03)00094-4.

27. Quarteroni A., Sacco R., Saleri F. Numerical Mathematics. Vol. 37. 2nd ed. Berlin, Heidelberg: Springer, 2007. XVIII, 657 p. Texts in Applied Mathematics. DOI: 10.1007/b98885.

28. Циглер Г.М. Теория многогранников. М.: МЦНМО, 2014. 568 с.

29. 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.

Соколинский Леонид Борисович, д.ф.-м.н., профессор, заведующий кафедрой системного программирования, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

Жулев Александр Эдуардович, аспирант кафедры системного программирования, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

Соколинская Ирина Михайловна, к.ф.-м.н., доцент кафедры прикладной математики и программирования, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

DOI: 10.14529/cmse260101

ON PROJECTION METHOD OF LINEAR PROGRAMMING

© 2026 L.B. Sokolinsky, A.E. Zhulev, I.M. Sokolinskaya

South Ural State University (pr. Lenina 76, Chelyabinsk, 454080 Russia)

E-mail: leonid.sokolinsky@susu.ru, zhulevae@susu.ru, irina.sokolinskaya@susu.ru

Received: 14.11.2025

The paper addresses the problem of developing efficient projection-type methods for linear programming (LP). A new hybrid projection algorithm called HAlEM (Hybrid Along Edges Movement) is proposed, combining ideas from the projection approach and the simplex method. The algorithm starts from an arbitrary vertex of the feasible solution polytope and moves along its edges toward the optimal vertex. An original two-factor projection method is used to compute the direction of movement, ensuring high computational accuracy for any LP problem. The main advantage of HAlEM over previous projection algorithms (AlFaMove, AlEM) is its avoidance of exhaustive combinatorial enumeration of all possible combinations of hyperplanes by employing a matrix basis update technique borrowed from the simplex method. This allows one to circumvent exponential computational complexity. The paper presents a parallel version of the algorithm based on the BSF parallel computing model and a master-worker scheme, enabling an efficient implementation for cluster-type supercomputer systems. Computational experiments were conducted on benchmark problems from the Netlib-LP repository and on a series of parameterized problems. The experimental results demonstrate that HAlEM, unlike the simplex method, possesses a higher degree of parallelism, allowing the efficient use of up to several dozen processor nodes, and exhibits good scalability with parallel efficiency not falling below 51%. Furthermore, the algorithm provides high computational accuracy at the level of machine epsilon.

Keywords: linear programming, HAlEM algorithm, projection method, parallel implementation, MPI, cluster computing system, scalability evaluation, Netlb-LP.

FOR CITATION

Sokolinsky L.B., Zhulev A.E., Sokolinskaya I.M. On Projection Method of Linear Programming. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2026. Vol. 15, no. 1. P. 5–37. (in Russian) DOI: 10.14529/cmse260101.

References

1. Pan P.-Q. Linear Programming Computation. Berlin, Heidelberg: Springer, 2014. 747 p. DOI: 10.1007/978-3-642-40754-3.
2. Bixby R.E. Solving Real-World Linear Programs: A Decade and More of Progress. Operations Research. 2002. Vol. 50, no. 1. P. 3–15. DOI: 10.1287/opre.50.1.3.17780.
3. Rasmussen S. Production Economics. The Basic Theory of Production Optimisation. 2nd ed. Berlin, Heidelberg: Springer, 2013. XII, 292 p. Springer Texts in Business and Economics. DOI: 10.1007/978-3-642-30200-8.
4. 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. P. 269–292. DOI: 10.1007/978-3-030-28565-4_23.

5. 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.

6. 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.

7. Dantzig G.B., Thapa M.N. Linear Programming 2: Theory and Extensions. Springer Series in Operations Research and Financial Engineering. New York, NY, USA: Springer, 2003. XXVI, 448 p. DOI: 10.1007/b97283.

8. Tolla P. A Survey of Some Linear Programming Methods. Concepts of Combinatorial Optimization / ed. by V.T. Paschos. 2nd ed. Hoboken, NJ, USA: John Wiley, Sons, 2014. Chap. 7. P. 157–188. DOI: 10.1002/9781119005216.ch7.

9. Mamalis B., Pantziou G. Advances in the Parallelization of the Simplex Method. Algorithms, Probability, Networks, and Games. Lecture Notes in Computer Science, vol. 9295 / ed. by C. Zaroliagis, G. Pantziou, S. Kontogiannis. Cham: Springer, 2015. P. 281–307. DOI: 10.1007/978-3-319-24024-4_17.

10. Im H., Wolkowicz H. Revisiting degeneracy, strict feasibility, stability, in linear programming. European Journal of Operational Research. 2023. Vol. 310, no. 2. P. 495–510. DOI: 10.1016/j.ejor.2023.03.021.

11. Gass S.I., Vinjamuri S. Cycling in linear programming problems. Computers and Operations Research. 2004. Vol. 31, no. 2. P. 303–311. DOI: 10.1016/S0305-0548(02)00226-5.

12. Hall J., McKinnon K. The simplest examples where the simplex method cycles and conditions where EXPAND fails to prevent cycling. Mathematical Programming, Series B. 2004. Vol. 100, no. 1. P. 133–150. DOI: 10.1007/s10107-003-0488-1.

13. Sokolinsky L.B., Sokolinskaya I.M. Apex Method: A New Scalable Iterative Method for Linear Programming. Mathematics. 2023. Vol. 11, no. 7. P. 1–28. DOI: 10.3390/MATH11071654.

14. Olkhovsky N.A., Sokolinsky L.B. Surface Movement Method for Linear Programming. Lobachevskii Journal of Mathematics. 2024. Vol. 45, no. 10. P. 5061–5079. DOI: 10.1134/S1995080224605745.

15. Olkhovsky N.A., Sokolinsky L.B. AlFaMove: Scalable Implementation of Surface Movement Method for Cluster Computing Systems. Supercomputing Frontiers and Innovations. 2024. Vol. 11, no. 3. P. 4–26. DOI: 10.14529/jsfi240301.

16. Zhulev A.E., Sokolinsky L.B. Finding optimal vertex on polytope of feasible solutions to linear programming problem. Numerical Methods and Programming. 2026. Vol. 27, no. 1. P. 1–18. (in Russian) DOI: 10.26089/NumMet.v27r101.

17. Murty K.G. Computational and Algorithmic Linear Algebra and n-Dimensional Geometry. Singapore: World Scientific Publishing Company, 2014. 480 p. DOI: 10.1142/8261.

18. Strang G. Linear Algebra and Its Applications. 2nd. Academic Press, 1980. 425 p.

19. Schrijver A. Theory of Linear and Integer Programming. Chichester, New York, Brisbane, Toronto, Singapore: Wiley, Sons, 1998. 484 p.

20. Sokolinsky L.B. BSF: A parallel computation model for scalability estimation of iterative numerical algorithms on cluster computing systems. Journal of Parallel and Distributed Computing. 2021. Vol. 149. P. 193–206. DOI: 10.1016/j.jpdc.2020.12.009.

21. Sokolinsky L.B. BSF-skeleton: A Template for Parallelization of Iterative Numerical Algorithms on Cluster Computing Systems. MethodsX. 2021. Vol. 8. Article number 101437. DOI: 10.1016/j.mex.2021.101437.

22. 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. Cham: Springer, 2022. P. 43–55. DOI: 10.1007/978-3-031-11623-0_4.

23. Gay D.M. Electronic mail distribution of linear programming test problems. Mathematical Programming Society COAL Bulletin. 1985. Vol. 13. P. 10–12.

24. Murtagh B.A. Advanced linear programming: computation and practice. New York, London: McGraw-Hill, 1981. xii, 202 p.

25. 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.

26. Koch T. The final NETLIB-LP results. Operations Research Letters. 2004. Vol. 32, no. 2. P. 138–142. DOI: 10.1016/S0167-6377(03)00094-4.

27. Quarteroni A., Sacco R., Saleri F. Numerical Mathematics. Vol. 37. 2nd ed. Berlin, Heidelberg: Springer, 2007. XVIII, 657 p. Texts in Applied Mathematics. DOI: 10.1007/b98885.

28. Ziegler G.M. Lectures on Polytopes. Vol. 152. New York, NY: Springer New York, 1995. XI, 370 p. Graduate Texts in Mathematics. DOI: 10.1007/978-1-4613-8431-1.

29. Sokolinsky L.B., Sokolinskaya I.M. FRaGenLP: A Generator of Random Linear Programming Problems for Cluster Computing Systems. Parallel Computational Technologies. PCT 2021. Communications in Computer and Information Science, vol. 1437 / ed. by L. Sokolinsky, M. Zymbler. Cham: Springer, 2021. P. 164–177. DOI: 10.1007/978-3-030-81691-9_12.