

ЧИСЛЕННАЯ РЕАЛИЗАЦИЯ МЕТОДА ПОВЕРХНОСТНОГО ДВИЖЕНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ

© 2024 Л.Б. Соколинский, Н.А. Ольховский, И.М. Соколинская

Южно-Уральский государственный университет

(454080 Челябинск, пр. им. В.И. Ленина, д. 76)

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

Поступила в редакцию: 10.06.2024

Работа посвящена численной реализации нового метода линейного программирования, получившего название «метод поверхностного движения». В основе реализации лежит оригинальный алгоритм AlFaMove, который строит на поверхности допустимого многогранника оптимальный целевой путь от произвольной граничной точки до точки, являющейся решением задачи линейного программирования. Оптимальность пути заключается в том, что направление движения по грани многогранника соответствует максимальному увеличению значения целевой функции. Для вычисления оптимального направления движения используется метод, базирующийся на операции построения псевдопроекции на линейное многообразие. Операция псевдопроекции обобщает понятие ортогональной проекции и реализуется с помощью итерационного алгоритма проекционного типа. Доказано, что в случае линейного многообразия, образуемого путем пересечения гиперплоскостей, псевдопроекция совпадает с ортогональной проекцией. Также доказано, что в случае линейного многообразия метод на основе псевдопроектирования вычисляет вектор движения в направлении максимального увеличения целевой функции. Выполнена параллельная реализация алгоритма AlFaMove. Приведены результаты вычислительных экспериментов на кластерной вычислительной системе, демонстрирующие высокую масштабируемость предложенной численной реализации.

Ключевые слова: линейное программирование, метод поверхностного движения, численная реализация, алгоритм AlFaMove, параллельная реализация, кластерная вычислительная система, исследование масштабируемости.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

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. X, № Y. С. Z1–Z2. DOI: 10.14529/cmse240Y0Z.

Введение

Эпоха больших данных и индустрия 4.0 породили задачи линейного программирования (ЛП) сверхбольших размерностей, включающих в себя миллионы переменных и миллионы ограничений [1–4]. Во многих случаях объектом линейного программирования являются задачи, связанные с оптимизацией нестационарных процессов [5]. В нестационарных задачах ЛП целевая функция и/или ограничения изменяются в течение вычислительного процесса. Также среди этого класса задач встречаются приложения, в которых необходимо выполнять оптимизацию в режиме реального времени. Для решения таких задач необходимы масштабируемые методы и параллельные алгоритмы линейного программирования.

Один из стандартных подходов к решению нестационарных задач оптимизации состоит в том, чтобы рассматривать каждое изменение как появление новой задачи оптимизации, которую необходимо решать с нуля [5]. Однако такой подход часто непрактичен, поскольку решение проблемы с нуля без повторного использования информации из прошлого может

занять слишком много времени. Таким образом, желательно иметь алгоритм оптимизации, способный непрерывно адаптировать решение к изменяющейся среде, повторно используя информацию, полученную в прошлом. Этот подход применим для процессов реального времени, если алгоритм достаточно быстро отслеживает траекторию движения оптимальной точки. В случае больших задач ЛП последнее требует разработки масштабируемых методов и параллельных алгоритмов ЛП.

До настоящего времени наиболее популярными методами решения задач ЛП являются симплекс-метод [6] и методы внутренних точек [7]. Эти методы способны решать задачи с десятками тысяч переменных и ограничений. Однако масштабируемость параллельных алгоритмов, основанных на симплекс-методе, в общем случае ограничивается 16–32 процессорными узлами [8]. Что касается алгоритмов внутренних точек, они не поддаются эффективному распараллеливанию в общем случае. Это ограничивает применение указанных методов для решения сверхбольших нестационарных задач ЛП в режиме реального времени. В соответствии с этим задача разработки масштабируемых методов и эффективных параллельных алгоритмов ЛП для кластерных вычислительных систем остается актуальной.

В недавней работе [9] было дано теоретическое описание нового метода ЛП — метода поверхностного движения, строящего на поверхности допустимого многогранника¹ оптимальный целевой путь к решению задачи ЛП. Под оптимальным целевым путем понимается путь по поверхности допустимого многогранника в направлении наибольшего увеличения значений целевой функции. Однако предложенный в этой статье алгоритм 1 на шаге 15 требует нахождения на границе гипердиска точки с максимальным значением целевой функции. При этом не приводится численный алгоритм, позволяющий выполнить этот шаг. В этой статье мы приводим и исследуем алгоритм AlFaMove, устраняющий допущенный пробел.

Статья организована следующим образом. Раздел 1 содержит теоретический базис, необходимый для описания метода поверхностного движения и его численной реализации. Раздел 2 посвящен описанию операции псевдопроекции, позволяющий найти вектор движения по оптимальному целевому пути для линейного многообразия, получаемого в результате пересечением гиперплоскостей. В разделе 3 дается формализованное описание алгоритма AlFaMove, представляющего собой численную реализацию метода поверхностного движения. Раздел 4 посвящен описанию параллельной версии алгоритма AlFaMove. В разделе 5 представлены информация о программной реализации алгоритма AlFaMove и результаты экспериментов на кластерной вычислительной системе по исследованию его масштабируемости. В заключении суммируются полученные результаты и намечаются направления дальнейших исследований.

1. Теоретический базис

Данный раздел содержит необходимый теоретический базис, используемый для описания алгоритма движения по граням AlFaMove. Рассмотрим задачу ЛП в следующем виде:

$$\bar{x} = \arg \max_{x \in \mathbb{R}^n} \{ \langle c, x \rangle \mid Ax \leq b \}, \quad (1)$$

где $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$, $m > 1$, $c \neq 0$. Здесь $\langle \cdot, \cdot \rangle$ обозначает скалярное произведение двух векторов. Мы предполагаем, что ограничение $x \geq 0$ также включено в матричное

¹ Допустимый многогранник — область допустимых решений задачи линейного программирования.

неравенство $A\mathbf{x} \leq \mathbf{b}$ в форме

$$-\mathbf{x} \leq \mathbf{0}.$$

Линейная целевая функция задачи (1) имеет вид

$$f(\mathbf{x}) = \langle \mathbf{c}, \mathbf{x} \rangle.$$

Вектор \mathbf{c} в данном случае является градиентом целевой функции $f(\mathbf{x})$.

Пусть $\mathbf{a}_i \in \mathbb{R}^n$ обозначает вектор, представляющий i -тую строку матрицы A . Мы предполагаем, что $\mathbf{a}_i \neq \mathbf{0}$ для всех $i \in \{1, \dots, m\}$. Обозначим через \hat{H}_i замкнутое полупространство, определяемое неравенством $\langle \mathbf{a}_i, \mathbf{x} \rangle \leq b_i$, а через H_i — ограничивающую его гиперплоскость:

$$\hat{H}_i = \{\mathbf{x} \in \mathbb{R}^n \mid \langle \mathbf{a}_i, \mathbf{x} \rangle \leq b_i\}; \quad (2)$$

$$H_i = \{\mathbf{x} \in \mathbb{R}^n \mid \langle \mathbf{a}_i, \mathbf{x} \rangle = b_i\}. \quad (3)$$

Определим допустимый многогранник

$$M = \bigcap_{i=1}^m \hat{H}_i, \quad (4)$$

представляющий множество допустимых точек задачи ЛП (1). Заметим, что M в этом случае будет замкнутым выпуклым множеством. Мы будем предполагать, что множество M является ограниченным и $M \neq \emptyset$, то есть задача ЛП (1) имеет решение.

Дадим определение рецессивного полупространства [10].

Определение 1. Полупространство \hat{H}_i называется рецессивным, если

$$\forall \mathbf{x} \in H_i, \forall \lambda > 0 : \mathbf{x} + \lambda \mathbf{c} \notin \hat{H}_i. \quad (5)$$

Геометрический смысл этого определения состоит в том, что луч, исходящий в направлении вектора \mathbf{c} из любой точки гиперплоскости, ограничивающей рецессивное полупространство, не имеет общих точек с этим полупространством, за исключением начальной. Известно [10], что следующее условие является необходимым и достаточным для того, чтобы полупространство \hat{H}_i было рецессивным:

$$\langle \mathbf{a}_i, \mathbf{c} \rangle > 0.$$

Определим

$$\mathcal{I} = \{i \in \{1, \dots, m\} \mid \langle \mathbf{a}_i, \mathbf{c} \rangle > 0\}, \quad (6)$$

то есть \mathcal{I} представляет множество индексов, для которых полупространство \hat{H}_i является рецессивным. Поскольку допустимый многогранник M представляет собой ограниченное множество, имеем

$$\mathcal{I} \neq \emptyset.$$

Положим

$$\hat{M} = \bigcap_{i \in \mathcal{I}} \hat{H}_i. \quad (7)$$

Очевидно, что \hat{M} является выпуклым, замкнутым, неограниченным многогранником. Будем называть его рецессивным. Из (4) и (6) следует

$$M \subset \hat{M}.$$

Обозначим через $\Gamma(M)$ множество граничных точек допустимого многогранника M , а через $\Gamma(\hat{M})$ — множество граничных точек рецессивного многогранника \hat{M}^2 . Согласно утверждению 3 в [10] имеем

$$\bar{x} \in \Gamma(\hat{M}),$$

то есть решение задачи ЛП (1) лежит на границе рецессивного многогранника \hat{M} .

Следуя [11], дадим определение ортогональной проекции на гиперплоскость.

Определение 2. Пусть в пространстве \mathbb{R}^n имеется гиперплоскость

$$H = \{x \in \mathbb{R}^n | \langle a, x \rangle = b\}.$$

Ортогональная проекция $\pi_H(v)$ точки $v \in \mathbb{R}^n$ на гиперплоскость H определяется формулой

$$\pi_H(v) = v - \frac{\langle a, v \rangle - b}{\|a\|^2} a. \quad (8)$$

Следующее утверждение дает способ вычисления оптимального пути на гиперплоскости.

Утверждение 1. Пусть в пространстве \mathbb{R}^n задана гиперплоскость H с нормалью $a \in \mathbb{R}^n$, проходящая через точку $u \in \mathbb{R}^n$:

$$H = \{x \in \mathbb{R}^n | \langle a, x \rangle = \langle a, u \rangle\}. \quad (9)$$

Пусть задана линейная функция $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$ с градиентом $c \in \mathbb{R}^n$:

$$f(x) = \langle c, x \rangle. \quad (10)$$

Пусть векторы a и c линейно независимы (не коллинеарны, и среди них нет нулевого вектора). Положим

$$v = u + c. \quad (11)$$

Построим ортогональную проекцию $\pi_H(v)$ точки v на гиперплоскость H :

$$w = \pi_H(v). \quad (12)$$

Тогда вектор $d = w - u$ однозначно задает направление максимального увеличения линейной функции $f(x)$, определяемой формулой (10).

Доказательство. Предположим противное, то есть существует точка $\tilde{w} \in H$ такая, что

$$\langle c, \tilde{w} \rangle \geq \langle c, w \rangle, \quad (13)$$

$\|\tilde{w} - u\| = \|w - u\|$ и $\tilde{w} \neq w$ (см. рис. 1). Здесь и далее $\|\cdot\|$ обозначает евклидову норму. Вычислим $\langle c, w \rangle$. В соответствии с определением 2 ортогональная проекция $\pi_H(v)$ точки v на гиперплоскость H , задаваемую формулой (9), вычисляется следующим образом:

$$w = v - \frac{\langle a, v - u \rangle}{\|a\|^2} a.$$

²Под граничной точкой множества $\hat{M} \subset \mathbb{R}^n$ понимается точка в \mathbb{R}^n , для которой любая открытая ее окрестность в \mathbb{R}^n имеет непустое пересечение как с множеством \hat{M} , так и с его дополнением.

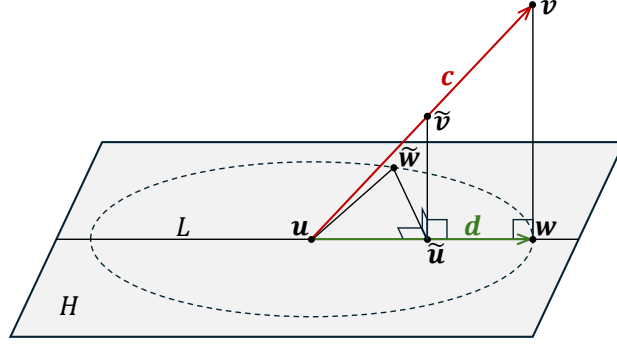


Рис. 1. Иллюстрация к доказательству предложения 1.
Пунктиром обозначена гиперокружность с радиусом $\|\mathbf{w} - \mathbf{u}\|$.

Подставив вместо \mathbf{v} правую часть формулы (11), получим

$$\mathbf{w} = \mathbf{u} + \mathbf{c} - \frac{\langle \mathbf{a}, \mathbf{c} \rangle}{\|\mathbf{a}\|^2} \mathbf{a}. \quad (14)$$

Используя (14), находим

$$\langle \mathbf{c}, \mathbf{w} \rangle = \langle \mathbf{c}, \mathbf{u} \rangle + \|\mathbf{c}\|^2 - \frac{\langle \mathbf{a}, \mathbf{c} \rangle^2}{\|\mathbf{a}\|^2}. \quad (15)$$

Поскольку \mathbf{a} и \mathbf{c} линейно независимы, в соответствии с неравенством Коши–Буняковского имеем

$$\langle \mathbf{a}, \mathbf{c} \rangle^2 < \|\mathbf{a}\|^2 \cdot \|\mathbf{c}\|^2.$$

Отсюда следует

$$\|\mathbf{c}\|^2 - \frac{\langle \mathbf{a}, \mathbf{c} \rangle^2}{\|\mathbf{a}\|^2} > 0. \quad (16)$$

Теперь вычислим $\langle \mathbf{c}, \tilde{\mathbf{w}} \rangle$. Определим $\tilde{\mathbf{u}} = \pi_L(\tilde{\mathbf{w}})$ — ортогональная проекция точки $\tilde{\mathbf{w}}$ на прямую L , проходящую через точки \mathbf{u} и \mathbf{w} . По построению существует число $\delta \in \mathbb{R}$, удовлетворяющее условию

$$-1 \leq \delta < 1, \quad (17)$$

такое, что

$$\tilde{\mathbf{u}} = \mathbf{u} + \delta(\mathbf{w} - \mathbf{u}).$$

Положим

$$\tilde{\mathbf{v}} = \mathbf{u} + \delta(\mathbf{v} - \mathbf{u}). \quad (18)$$

Тогда точка $\tilde{\mathbf{u}}$ является ортогональной проекцией точки $\tilde{\mathbf{v}}$ на гиперплоскость H , определяемую формулой (9), и может быть вычислена следующим образом:

$$\tilde{\mathbf{u}} = \tilde{\mathbf{v}} - \frac{\langle \mathbf{a}, \tilde{\mathbf{v}} - \mathbf{u} \rangle}{\|\mathbf{a}\|^2} \mathbf{a}.$$

Подставив вместо $\tilde{\mathbf{v}}$ правую часть формулы (18), откуда получим

$$\tilde{\mathbf{u}} = \mathbf{u} + \delta \left(\mathbf{v} - \mathbf{u} - \frac{\langle \mathbf{a}, \mathbf{v} - \mathbf{u} \rangle}{\|\mathbf{a}\|^2} \mathbf{a} \right).$$

Используя (11), произведем замену $\mathbf{v} - \mathbf{u} = \mathbf{c}$:

$$\tilde{\mathbf{u}} = \mathbf{u} + \delta \left(\mathbf{c} - \frac{\langle \mathbf{a}, \mathbf{c} \rangle}{\|\mathbf{a}\|^2} \mathbf{a} \right). \quad (19)$$

Очевидно,

$$\tilde{\mathbf{w}} = (\tilde{\mathbf{w}} - \tilde{\mathbf{u}}) + \tilde{\mathbf{u}}. \quad (20)$$

Заменим второе слагаемое в (20) на правую часть формулы (19):

$$\tilde{\mathbf{w}} = (\tilde{\mathbf{w}} - \tilde{\mathbf{u}}) + \mathbf{u} + \delta \left(\mathbf{c} - \frac{\langle \mathbf{a}, \mathbf{c} \rangle}{\|\mathbf{a}\|^2} \mathbf{a} \right).$$

Отсюда получаем

$$\langle \mathbf{c}, \tilde{\mathbf{w}} \rangle = \langle \mathbf{c}, \tilde{\mathbf{w}} - \tilde{\mathbf{u}} \rangle + \langle \mathbf{c}, \mathbf{u} \rangle + \delta \left(\|\mathbf{c}\|^2 - \frac{\langle \mathbf{a}, \mathbf{c} \rangle^2}{\|\mathbf{a}\|^2} \right).$$

По построению вектор $\tilde{\mathbf{w}} - \tilde{\mathbf{u}}$ ортогонален вектору $\mathbf{v} - \mathbf{u} = \mathbf{c}$. Поэтому $\langle \mathbf{c}, \tilde{\mathbf{w}} - \tilde{\mathbf{u}} \rangle = 0$. Таким образом, последняя формула преобразуется к виду

$$\langle \mathbf{c}, \tilde{\mathbf{w}} \rangle = \langle \mathbf{c}, \mathbf{u} \rangle + \delta \left(\|\mathbf{c}\|^2 - \frac{\langle \mathbf{a}, \mathbf{c} \rangle^2}{\|\mathbf{a}\|^2} \right). \quad (21)$$

Сопоставляя (15) и (21), с учетом (16) и (17) получаем

$$\langle \mathbf{c}, \tilde{\mathbf{w}} \rangle < \langle \mathbf{c}, \mathbf{w} \rangle,$$

что противоречит (13). □

Возвращаясь к задаче ЛП (1), можно сказать следующее. Пусть $\mathbf{u} \in M \cap \Gamma(\hat{M})$ и существует единственная рецессивная гиперплоскость $H_{i'}$ ($i' \in \mathcal{I}$) такая, что $\mathbf{u} \in H_{i'}$. В этом случае вектор \mathbf{d} , определяющий направление оптимального целевого пути из точки \mathbf{u} , в соответствии с утверждением 1 вычисляется по формуле

$$\mathbf{d} = \mathbf{c} - \frac{\langle \mathbf{a}_{i'}, \mathbf{u} + \mathbf{c} \rangle - b_{i'}}{\|\mathbf{a}_{i'}\|^2} \mathbf{a}_{i'}. \quad (22)$$

В следующем разделе мы рассмотрим общий случай, когда через точку \mathbf{u} проходит несколько гиперплоскостей.

2. Операция псевдопроектирования на линейное многообразие

Пусть $\mathcal{J} \subseteq \{1, \dots, m\}$, $\mathcal{J} \neq \emptyset$, и $\bigcap_{i \in \mathcal{J}} H_i \neq \emptyset$. В этом случае множество индексов \mathcal{J} определяет в пространстве \mathbb{R}^n линейное многообразие

$$L = \bigcap_{i \in \mathcal{J}} H_i. \quad (23)$$

Обозначим k_L — размерность линейного многообразия L . При $k_L < n - 1$ многообразие L не является гиперплоскостью, и для определения вектора движения \mathbf{d} по этому многообразию в направлении максимального увеличения целевой функции нельзя применить

формулу (22), так как такое линейное многообразие невозможно задать одним линейным уравнением в пространстве \mathbb{R}^n . Однако мы можем найти указанный вектор \mathbf{d} с помощью операции псевдопроектирования [10]. Определим проекционное отображение $\varphi(\cdot)$:

$$\varphi(\mathbf{x}) = \frac{1}{|\mathcal{J}|} \sum_{i \in \mathcal{J}} \pi_{H_i}(\mathbf{x}). \quad (24)$$

Известно [12], что отображение $\varphi(\mathbf{x})$ является непрерывным L -фейеровским отображением, и последовательность точек

$$\left\{ \mathbf{x}_k = \varphi^k(\mathbf{x}_0) \right\}_{k=1}^{\infty}, \quad (25)$$

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

$$\mathbf{x}_k \rightarrow \tilde{\mathbf{x}} \in L.$$

Теперь мы готовы дать определение псевдопроекции на линейное многообразие, образуемое пересечением гиперплоскостей.

Определение 3. Пусть $\mathcal{J} \subseteq \{1, \dots, m\}$, $\mathcal{J} \neq \emptyset$, $\bigcap_{i \in \mathcal{J}} H_i \neq \emptyset$, $\varphi(\cdot)$ — проекционное отображение, определяемое формулой (24). Псевдопроекцией $\rho_{\mathcal{J}}(\mathbf{x})$ точки $\mathbf{x} \in \mathbb{R}^n$ на линейное многообразие $L = \bigcap_{i \in \mathcal{J}} H_i$ называется предельная точка последовательности (25):

$$\lim_{k \rightarrow \infty} \left\| \rho_{\mathcal{J}}(\mathbf{x}) - \varphi^k(\mathbf{x}) \right\| = 0.$$

Важным свойством псевдопроекции на линейное многообразие является то, что в этом случае псевдопроекция совпадает с ортогональной проекцией. Для доказательства этого факта нам понадобится следующая лемма.

Лемма 1. Пусть в пространстве \mathbb{R}^n заданы гиперплоскость $H_{i'}$ и линейное многообразие L , принадлежащее этой гиперплоскости:

$$\begin{aligned} H_{i'} &= \{ \mathbf{x} \in \mathbb{R}^n \mid \langle \mathbf{a}_{i'}, \mathbf{x} \rangle = b_{i'} \}; \\ L &= \bigcap_{i \in \mathcal{J}} H_i; \\ i' &\in \mathcal{J}. \end{aligned}$$

Обозначим через P линейное многообразие, являющееся ортогональным дополнением к L :

$$P = L^{\perp}. \quad (26)$$

Тогда для любой точки \mathbf{v} , принадлежащей линейному многообразию P , ее ортогональная проекция $\pi_{H_{i'}}(\mathbf{v})$ на гиперплоскость $H_{i'}$ также будет принадлежать линейному многообразию P :

$$\forall \mathbf{v} \in P : \pi_{H_{i'}}(\mathbf{v}) \in P.$$

Доказательство. Обозначим \mathbf{p} — ортогональная проекция точки \mathbf{v} на гиперплоскость $H_{i'}$:

$$\mathbf{p} = \pi_{H_{i'}}(\mathbf{v}). \quad (27)$$

Без ограничения общности мы можем считать, что $\mathbf{p} \in L$ (см. рис 2). Предположим, что

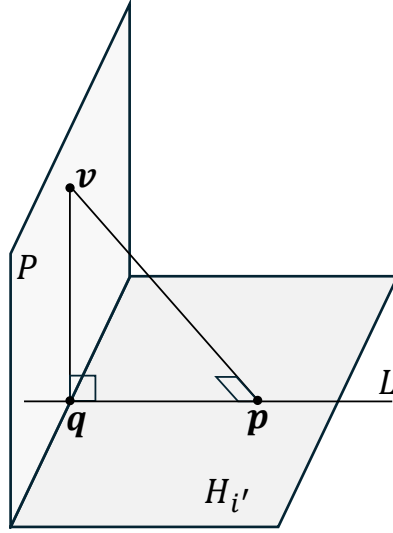


Рис. 2. Иллюстрация к лемме 1.

точка \mathbf{p} не принадлежит линейному многообразию P . Обозначим \mathbf{q} — точка пересечения линейного многообразия L с его ортогональным дополнением P :

$$\mathbf{q} = L \cap P.$$

Поскольку P является ортогональным дополнением к линейному многообразию L , точка \mathbf{q} является ортогональной проекцией точки \mathbf{p} на линейное многообразие P :

$$\mathbf{q} = \pi_P(\mathbf{p}). \quad (28)$$

Рассмотрим треугольник $\triangle(\mathbf{v}, \mathbf{p}, \mathbf{q})$. В силу (27) угол $\angle \mathbf{p}$ с вершиной в точке \mathbf{p} является прямым. В соответствии с (28) угол $\angle \mathbf{q}$ с вершиной в точке \mathbf{q} также является прямым. Но это возможно только в случае, когда $\mathbf{p} = \mathbf{q}$, то есть

$$\mathbf{p} \in P.$$

Лемма доказана. □

Следующее утверждение доказывает, что псевдопроекция на линейное многообразие совпадает с ортогональной проекцией.

Утверждение 2. Пусть выполняются следующие условия:

$$\mathcal{J} \subseteq \{1, \dots, m\}, \quad (29)$$

$$L = \bigcap_{i \in \mathcal{J}} H_i, \quad L \neq \emptyset; \quad (30)$$

где $H_i = \{\mathbf{x} \in \mathbb{R}^n \mid \langle \mathbf{a}_i, \mathbf{x} \rangle = b_i\}$. Обозначим $\pi_L(\mathbf{x})$ — ортогональная проекция точки $\mathbf{x} \in \mathbb{R}^n$ на линейное многообразие L . Тогда

$$\rho_L(\mathbf{x}) = \pi_L(\mathbf{x}),$$

то есть, псевдопроекция на линейное многообразие L совпадает с ортогональной проекцией.

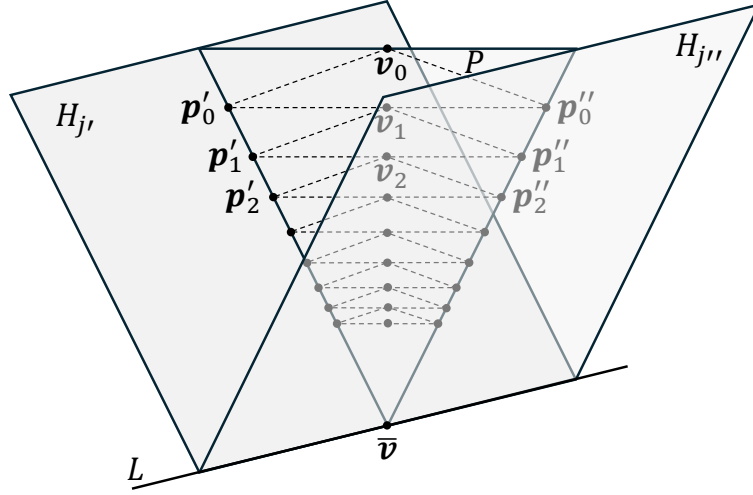


Рис. 3. Иллюстрация к доказательству утверждения 2 при $n = 3$.

Доказательство. Зафиксируем произвольную точку $\mathbf{v}_0 \in \mathbb{R}^n$. Рассмотрим линейное многообразие P , содержащее точку \mathbf{v}_0 , и являющееся ортогональным дополнением к линейному многообразию L :

$$\mathbf{v}_0 \in P = L^\perp. \quad (31)$$

Обозначим через $\bar{\mathbf{v}}$ точку пересечения линейного многообразия L с его ортогональным дополнением P :

$$L \cap P = \{\bar{\mathbf{v}}\}$$

(см. рис. 3). Построим ортогональную проекцию точки \mathbf{v}_0 на гиперплоскость H_j для произвольного $j \in \mathcal{J}$:

$$\mathbf{p}_0 = \pi_{H_j}(\mathbf{v}_0).$$

В соответствии с леммой 1 имеем

$$\pi_{H_j}(\mathbf{v}_0) \in P.$$

Отсюда и из (24) следует, что

$$\mathbf{v}_1 = \varphi(\mathbf{v}_0) \in P.$$

Это означает, что последовательность

$$\left\{ \mathbf{v}_k = \varphi^k(\mathbf{v}_0) \right\}_{k=1}^{\infty}$$

сходится к точке $\bar{\mathbf{v}}$ пересечения линейного многообразия L с линейным многообразием P , то есть, $\rho_L(\mathbf{v}_0) = \bar{\mathbf{v}}$. С другой стороны, в силу (31) имеем $\pi_L(\mathbf{v}_0) = \bar{\mathbf{v}}$. Следовательно

$$\forall \mathbf{x} \in \mathbb{R}^n : \rho_L(\mathbf{x}) = \pi_L(\mathbf{x}).$$

Утверждение доказано. □

Процедура приближенного вычисления псевдопроекции на линейное многообразие представлена в виде алгоритма 1. Дадим краткий комментарий по шагам этого алгоритма. На шаге 2 счетчик итераций k устанавливается в значение ноль. Шаг 3 задает начальное приближение \mathbf{x}_0 . Шаг 4 открывает итерационный цикл вычисления псевдопроекции. Шаги 5–8 для текущего приближения \mathbf{x}_k вычисляют сумму из правой части формулы (24).

Алгоритм 1 Вычисление псевдопроекции $\rho_{\mathcal{J}}(\mathbf{x})$

Require: $H_i = \{\mathbf{x} \in \mathbb{R}^n \mid \langle \mathbf{a}_i, \mathbf{x} \rangle = b_i\}$; $\mathcal{J} \subseteq \{1, \dots, m\}$; $\mathcal{J} \neq \emptyset$; $\bigcap_{i \in \mathcal{J}} H_i \neq \emptyset$

```
1: function  $\rho_{\mathcal{J}}(\mathbf{x})$ 
2:    $k := 0$ 
3:    $\mathbf{x}_0 := \mathbf{x}$ 
4:   repeat
5:      $\Sigma := 0$ 
6:     for  $i \in \mathcal{J}$  do
7:        $\Sigma := \Sigma + (\langle \mathbf{a}_i, \mathbf{x}_k \rangle - b_i) \mathbf{a}_i / \|\mathbf{a}_i\|^2$ 
8:     end for
9:      $\mathbf{x}_{(k+1)} := \mathbf{x}_k - \Sigma / |\mathcal{J}|$ 
10:     $\xi_{max} := 0$  ▷ Максимальная невязка
11:    for  $i \in \mathcal{J}$  do
12:       $\xi_i := \|\langle \mathbf{a}_i, \mathbf{x}_{k+1} \rangle - b_i\|$ 
13:      if  $\xi_i > \xi_{max}$  then
14:         $\xi_{max} := \xi_i$ 
15:      end if
16:    end for
17:     $k := k + 1$ 
18:  until  $\xi_{max} < \epsilon_{\xi}$  ▷ Малый параметр  $\epsilon_{\xi} > 0$ 
19:  return  $\mathbf{x}_k$ 
20: end function
```

Шаг 9 находит следующее приближение \mathbf{x}_{k+1} . Шаги 10–16 определяют максимальную невязку ξ для нового приближения относительно всех гиперплоскостей H_i , участвующих в вычислении. На шаге 17 счетчик итераций увеличивается на единицу. Шаг 18 проверяет условие выхода из итерационного цикла. Шаг 19 возвращает в качестве результата полученное приближение \mathbf{x}_k .

3. Алгоритм движения по граням AlFaMove

В этом разделе мы опишем новый алгоритм AlFaMove (Along Faces Movement), представляющий собой численную реализацию метода поверхностного движения [9]. Алгоритм AlFaMove строит на поверхности допустимого многогранника путь из произвольной граничной точки $\mathbf{u}_0 \in M \cap \Gamma(\dot{M})$ до точки $\bar{\mathbf{x}}$, являющейся решением задачи ЛП (1). Перемещение по граням допустимого многогранника происходит в направлении наибольшего увеличения значения целевой функции. Путь, построенный в результате такого движения, называется оптимальным целевым путем.

В основе алгоритма AlFaMove лежит процедура $\mathbf{D}(\mathbf{u})$ вычисления вектора движения $\bar{\mathbf{d}}$ по грани допустимого многогранника M из точки \mathbf{u} в направлении максимального увеличения значения целевой функции. Процедура $\mathbf{D}(\mathbf{u})$ представлена в виде алгоритма 2. Схематично работа этого алгоритма изображена на рис. 4. Дадим краткий комментарий по шагам алгоритма 2. Шаги 2–7 строят множество \mathcal{U} , включающее в себя индексы всех гиперплоскостей H_i , проходящих через точку \mathbf{u} . На шаге 8 вектору направления движения $\bar{\mathbf{d}}$ в качестве начального значения присваивается нулевой вектор. На шаге 9 значению целевой

Алгоритм 2 Вычисление вектора движения $\bar{d} = D(u)$

Require: $H_i = \{x \in \mathbb{R}^n \mid \langle a_i, x \rangle = b_i\}; u \in \Gamma(M)$

```
1: function  $D(u)$ 
2:    $\mathcal{U} := \emptyset$   $\triangleright \mathcal{U}$  — множество индексов гиперплоскостей  $H_i$ , проходящих через точку  $u$ 
3:   for  $i = 1 \dots m$  do
4:     if  $\langle a_i, u \rangle = b_i$  then
5:        $\mathcal{U} := \mathcal{U} \cup \{i\}$ 
6:     end if
7:   end for
8:    $\bar{d} := 0$ 
9:    $f := -\infty$   $\triangleright f$  — значение целевой функции  $f(x) = \langle c, x \rangle$ 
10:   $e_c := c / \|c\|$ 
11:   $v := u + \delta e_c$   $\triangleright$  Большой параметр  $\delta > 0$ 
12:  for  $\mathcal{J} \in \mathcal{P}(\mathcal{U}) \setminus \emptyset$  do  $\triangleright \mathcal{P}(\mathcal{U})$  — множество всех подмножеств множества  $\mathcal{U}$ 
13:     $w := \rho_{\mathcal{J}}(v)$ 
14:     $d := w - u$ 
15:     $e_d := d / \|d\|$ 
16:    if  $(u + \tau e_d) \in M$  then  $\triangleright$  Малый параметр  $\tau > 0$ 
17:      if  $\langle c, u + \tau e_d \rangle > f$  then
18:         $f := \langle c, u + \tau e_d \rangle$ 
19:         $\bar{d} := d$ 
20:      end if
21:    end if
22:  end for
23:  return  $\bar{d}$ 
24: end function
```

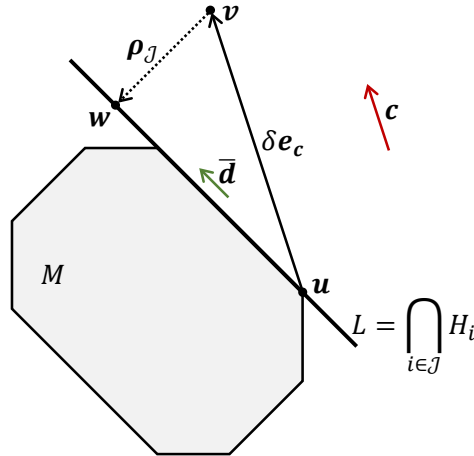


Рис. 4. Вычисление вектора направления движения \bar{d} по грани линейного многообразия L .

функции f присваивается бесконечно малое число³. На шаге 10 строится единичный вектор e_c , сонаправленный с вектором c . На шаге 11 строится точка v путем прибавления вектора

³В случае 64-разрядной арифметики с плавающей точкой, бесконечно малое число равно $-1 \cdot 10^{308}$

$\delta \mathbf{e}_c$ к вектору \mathbf{u} (см. рис. 4). Здесь δ — «большой» положительный параметр: чем больше δ , тем точнее будет вычислен вектор направления движения $\bar{\mathbf{d}}$. Однако при увеличении параметра δ будет увеличиваться и время вычисления псевдопроекции $\rho_{\mathcal{J}}(\mathbf{v})$ на шаге 13. Цикл **for** на шаге 12 перебирает все возможные комбинации индексов гиперплоскостей, проходящих через точку \mathbf{u} . Каждой такой комбинации \mathcal{J} соответствует линейное многообразие $L = \bigcap_{i \in \mathcal{J}} H_i$, которое также проходит через точку \mathbf{u} . Шаг 13 вычисляет точку \mathbf{w} , выполняя псевдопроекцию точки \mathbf{v} на линейное многообразие, соответствующее комбинации \mathcal{J} . На шаге 14 вычисляется возможный вектор движения \mathbf{d} по текущему линейному многообразию в направлении максимального увеличения значений целевой функции. Шаг 15 вычисляет единичный вектор \mathbf{e}_d , сонаправленный с вектором \mathbf{d} . Шаг 16 проверяет, что малое движение из точки \mathbf{u} в направлении \mathbf{d} не выходит за границы допустимого многогранника. Шаг 17, в свою очередь, проверяет, что значение целевой функции в точке $(\mathbf{u} + \mathbf{e}_d)$ превышает максимальное значение, полученное на предыдущих итерациях цикла **for** (шаг 12). В этом случае это значение запоминается как максимальное (шаг 18), а найденное направление присваивается вектору $\bar{\mathbf{d}}$ (шаг 19). После того, как проверены все возможные комбинации, вектор $\bar{\mathbf{d}}$ возвращается в качестве результата (шаг 23). Если ни одна комбинация не прошла проверку на шагах 16–17, то в качестве результата будет возвращен нулевой вектор. Это означает, что любое движение из точки \mathbf{u} по поверхности допустимого многогранника не приводит к увеличению значения целевой функции.

Теперь мы готовы описать алгоритм движения по граням AlFaMove, решающий задачу ЛП (1). За основу мы берем алгоритм 1 из работы [9]. Реализация алгоритма AlFaMove на псевдокоде представлена в виде алгоритма 3. Прокомментируем шаги этого алгоритма. На шаге 1 вводится начальное приближение $\mathbf{u}^{(0)}$. Это может быть произвольная граничная точка рецессивного многогранника \hat{M} , удовлетворяющая условию

$$\mathbf{u}_0 \in M \cap \Gamma(\hat{M}).$$

Это условие проверяется на шаге 2. Для получения подходящего начального приближения может применяться алгоритм, реализующий стадию Quest апекс-метода [10]. Шаг 3 вычисляет начальный вектор движения \mathbf{d}_0 . Для этого используется функция $\mathbf{D}(\cdot)$, реа-

Алгоритм 3 Алгоритм движения по граням AlFaMove

Require: $\hat{H}_i = \{\mathbf{x} \in \mathbb{R}^n \mid \langle \mathbf{a}_i, \mathbf{x} \rangle \leq b_i\}$; $M = \bigcap_{i=1}^m \hat{H}_i$; $\hat{M} = \bigcap_{i \in \mathcal{I}} \hat{H}_i$; $i \in \mathcal{I} \Leftrightarrow \langle \mathbf{a}_i, \mathbf{c} \rangle > 0$

```

1: input  $\mathbf{u}_0$ 
2: assert  $\mathbf{u}_0 \in M \cap \Gamma(\hat{M})$ 
3:  $\mathbf{d}_0 := \mathbf{D}(\mathbf{u}_0)$ 
4: assert  $\mathbf{d}_0 \neq \mathbf{0}$ 
5:  $k := 0$ 
6: repeat
7:    $\mathbf{u}_{k+1} := \mu(\mathbf{u}_k, \mathbf{d}_k)$ 
8:    $\mathbf{d}_{k+1} := \mathbf{D}(\mathbf{u}_{k+1})$ 
9:    $k := k + 1$ 
10: until  $\mathbf{d}_k = \mathbf{0}$ 
11: output  $\mathbf{u}_k$ 
12: stop
```

▷ Решение задачи ЛП (1)

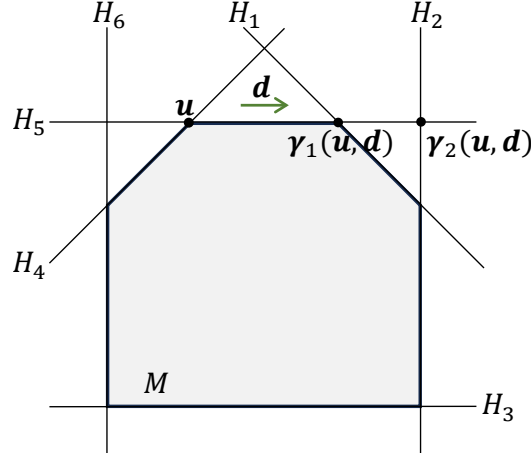


Рис. 5. Действие функции μ :
 $\mu(u, d) = \gamma_1(u, d)$.

лизованная в алгоритме 2. Предполагается, что $d_0 \neq 0^4$. Это условие контролируется на шаге 4. На шаге 5 счетчик итераций k устанавливается в значение ноль. Шаг 6 открывает цикл **repeat**, выполняющий движение по граням допустимого многогранника до тех пор, пока очередной вектор движения d_k не окажется нулевым вектором. Это означает, что последнее найденное приближение u_k является решением задачи ЛП (1). Шаг 7 в теле цикла вычисляет следующее приближение u_{k+1} с помощью вектор-функции μ (определение см. ниже). Шаг 8 вычисляет вектор движения d_{k+1} для вновь найденного приближения u_{k+1} . Шаг 9 увеличивает счетчик итераций k на единицу. Если последний вектор движения равен нулевому вектору, то на шаге 10 происходит выход из цикла **repeat/until**, после чего на шаге 11 выводятся координаты точки u_k в качестве решения задачи ЛП (1). Шаг 12 завершает работу алгоритма AlFaMove.

Вектор-функция $\mu(\cdot)$, используемая на шаге 7, определяется следующим образом. Обозначим

$$\mathcal{Q} = \{i \in \{1, \dots, m\} \mid \langle a_i, u \rangle < b_i \wedge \langle a_i, d \rangle > 0\}. \quad (32)$$

Тогда

$$\mu(u, d) = \arg \min_{i \in \mathcal{Q}} \{\|u - x\| \mid x = \gamma_i(u, d)\}. \quad (33)$$

Здесь $\gamma_i(u, d)$ обозначает вектор-функцию, вычисляющую косоугольную проекцию точки u на гиперплоскость H_i по направлению d :

$$\gamma_i(u, d) = u - \frac{\langle a_i, u \rangle - b_i}{\langle a_i, d \rangle} d.$$

Действие функции μ проиллюстрировано на рис. 5. В соответствии с рисунком гиперплоскости H_4 и H_5 не удовлетворяют неравенству $\langle a_i, u \rangle < b_i$ в (32). Гиперплоскости H_3 и H_6 не удовлетворяют неравенству $\langle a_i, d \rangle > 0$ в (32). Таким образом, $\mathcal{Q} = \{1, 2\}$. Так как $\|u - \gamma_1(u, d)\| < \|u - \gamma_2(u, d)\|$, то $\mu(u, d) = \gamma_1(u, d)$.

Сходимость алгоритма 3 к решению задачи ЛП (1) за конечное число итераций обеспечивается следующей теоремой.

⁴Равенство вектора d_0 нулевому вектору означает, что точка u_0 является решением задачи ЛП (1).

Теорема 1. (Сходимость алгоритма AlFaMove) Пусть допустимый многогранник M задачи ЛП (1) является ограниченным непустым множеством. Пусть $\bar{\mathbf{x}}$ — решение задачи ЛП (1). Тогда последовательность приближений $\{\mathbf{u}_k\}_{k=1}^K$, генерируемых алгоритмом 3, является конечной ($K < +\infty$), и $\langle \mathbf{c}, \mathbf{u}_K \rangle = \langle \mathbf{c}, \bar{\mathbf{x}} \rangle$, то есть \mathbf{u}_K является решением задачи ЛП (1).

Доказательство. Обозначим через $\mathbf{d}_{AlFaMove}$ вектор \mathbf{d}_{k+1} , вычисляемый на шаге 8 алгоритма 3. В соответствии с шагами 13, 14 алгоритма 2 имеем

$$\mathbf{d}_{AlFaMove} = \rho_{\mathcal{J}}(\mathbf{v}) - \mathbf{u}.$$

Согласно утверждению 2 отсюда следует

$$\mathbf{d}_{AlFaMove} = \pi_{\mathcal{J}}(\mathbf{v}) - \mathbf{u}, \quad (34)$$

где $\pi_{\mathcal{J}}(\mathbf{v})$ обозначает ортогональную проекцию точки \mathbf{v} на линейное многообразие $L = \bigcap_{i \in \mathcal{J}} H_i$. В соответствии с утверждением 1 это означает, что вектор $\mathbf{d}_{AlFaMove}$ однозначно задает направление максимального увеличения целевой функции $f(\mathbf{x})$ задачи (1). А это, в свою очередь, означает, что алгоритм 3 является численной реализацией метода поверхностного движения [9], то есть последовательности приближений $\{\mathbf{u}_{AlFaMove}^k\}$ и $\{\mathbf{u}_{SMM}^k\}$, генерируемые алгоритмом 3 из настоящей статьи и алгоритмом 1 из [9] соответственно, совпадают. Таким образом сходимость алгоритма AlFaMove непосредственно следует из сходимости алгоритма поверхностного движения, гарантируемой теоремой 1 из статьи [9]. \square

4. Параллельная версия алгоритма AlFaMove

Наиболее ресурсоемкой операцией, используемой в алгоритме AlFaMove (алгоритм 3), является операция вычисления вектора движения, выполняемая в цикле на шаге 8. При решении больших задач ЛП она занимает более 90% процессорного времени. Это объясняется тем, что функция вычисления вектора движения $\mathbf{D}(\cdot)$, реализованная в виде алгоритма 2, использует в цикле на шаге 13 операцию псевдопроектирования, заключающуюся в последовательном применении отображения $\varphi(\cdot)$, задаваемого формулой (24), к исходной точке (см. алгоритм 1, реализующий вычисление псевдопроекции). Известно, что в случае больших задач ЛП проекционный метод может потребовать значительных временных затрат [13]. Кроме того, следует отметить, что алгоритм 2 в цикле на шаге 12 перебирает все непустые подмножества множества \mathcal{U} , включающего в себя индексы гиперплоскостей, проходящих через точку \mathbf{u} . Если, например, через точку проходит 30 гиперплоскостей, то у нас получится $2^{30} - 1 = 1\,073\,741\,823$ непустых подмножества. Перебор такого количества подмножеств потребует суперкомпьютерных мощностей. Потому мы разработали параллельную версию алгоритма AlFaMove, представленную в виде алгоритма 4.

Параллельный алгоритм 4 построен на основе модели параллельных вычислений BSF [14], ориентированной на кластерные вычислительные системы. Модель BSF использует схему распараллеливания «мастер–рабочие» и требует представление алгоритма в виде операций над списками с использованием функций высшего порядка *Map* и *Reduce*.

В качестве второго параметра функции высшего порядка *Map* в алгоритме 4 используется список $\mathcal{L}_{map} = [1, \dots, K]$, содержащий порядковые номера всех подмножеств множества \mathcal{U} , за исключением пустого. Здесь $K = 2^{|\mathcal{U}|} - 1$. В качестве первого параметра

Алгоритм 4 Параллельная версия алгоритма AlFaMove

мастер	l -тый рабочий ($l = 0, \dots, L - 1$)
1: input $n, m, A, \mathbf{b}, \mathbf{u}_0$	1: input $n, m, A, \mathbf{b}, \mathbf{c}$
2: $k := 0$	2:
3: repeat	3: repeat
4: Broadcast \mathbf{u}_k	4: RecvFromMaster \mathbf{u}_k
5:	5: $\mathcal{U} := []$
6:	6: for $i = 1 \dots m$ do
7:	7: if $\langle \mathbf{a}_i, \mathbf{u}_k \rangle = b_i$ then
8:	8: $\mathcal{U} := \mathcal{U} \uplus [i]$
9:	9: end if
10:	10: end for
11:	11: $K := 2^{ \mathcal{U} } - 1$
12:	12: $L := \text{NumberOfWorkers}$
13:	13: $\mathcal{L}_{\text{map}(l)} := [lK/L, \dots, (l+1)K/L - 1]$
14:	14: $\mathcal{L}_{\text{reduce}(l)} := \text{Map}(\mathbf{F}_{\mathbf{u}_k}, \mathcal{L}_{\text{map}(l)})$
15:	15: $(\mathbf{d}_l, f_l) := \text{Reduce}(\oplus, \mathcal{L}_{\text{reduce}(l)})$
16:	16: SendToMaster (\mathbf{d}_l, f_l)
17:	17:
18: if $\mathbf{d}_k = \mathbf{0}$ then	18:
19: $\text{exit} := \text{true}$	19:
20: else	20:
21: $\mathbf{u}_{k+1} := \boldsymbol{\mu}(\mathbf{u}_k, \mathbf{d}_k)$	21:
22: $k := k + 1$	22:
23: $\text{exit} := \text{false}$	23:
24: end if	24:
25: Broadcast exit	25: RecvFromMaster exit
26: until exit	26: until exit
27: output \mathbf{u}_k, f_k	27:
28: stop	28: stop

фигурирует параметризованная функция

$$\mathbf{F}_{\mathbf{u}} : \{1, \dots, K\} \rightarrow \mathbb{R}^n \times \mathbb{R},$$

определенная следующим образом:

$$\begin{aligned}
 \mathbf{F}_{\mathbf{u}}(j) &= (\mathbf{d}_j, f_j); \\
 \mathbf{d}_j &= \begin{cases} \mathbf{e}_d, & \text{if } (\mathbf{u} + \tau \mathbf{e}_d) \in M \wedge \langle \mathbf{c}, \mathbf{w} \rangle > \langle \mathbf{c}, \mathbf{u} \rangle; \\ \mathbf{0}, & \text{if } (\mathbf{u} + \tau \mathbf{e}_d) \notin M \vee \langle \mathbf{c}, \mathbf{w} \rangle \leq \langle \mathbf{c}, \mathbf{u} \rangle; \end{cases} \\
 f_j &= \begin{cases} \langle \mathbf{c}, \mathbf{u} + \mathbf{e}_d \rangle, & \text{if } (\mathbf{u} + \tau \mathbf{e}_d) \in M \wedge \langle \mathbf{c}, \mathbf{w} \rangle > \langle \mathbf{c}, \mathbf{u} \rangle; \\ -\infty, & \text{if } (\mathbf{u} + \tau \mathbf{e}_d) \notin M \vee \langle \mathbf{c}, \mathbf{w} \rangle \leq \langle \mathbf{c}, \mathbf{u} \rangle, \end{cases}
 \end{aligned} \tag{35}$$

где

$$\mathbf{w} = \rho_{\sigma(j)}(\mathbf{u} + \delta \mathbf{c} / \|\mathbf{c}\|) \quad (36)$$

и

$$\mathbf{e}_d = \frac{\mathbf{w} - \mathbf{u}}{\|\mathbf{w} - \mathbf{u}\|}.$$

Семантика функции $F_{\mathbf{u}}(\cdot)$ однозначно определяется алгоритмом 2. Функция $\sigma(\cdot)$, используемая в формуле (36), отображает натуральное число $j \in \{1, \dots, K\}$ в j -тое подмножество множества элементов списка \mathcal{U} следующим образом. Число j преобразуется в двоичное представление, состоящее из $|\mathcal{U}|$ разрядов. Каждому разряду соответствует индекс гиперплоскости из списка \mathcal{U} в естественном порядке. Если разряд содержит единицу, то соответствующий индекс входит в подмножество $\sigma(j)$, если — ноль, то не входит. Например, пусть через точку \mathbf{u} проходят гиперплоскости H_2, H_4, H_7, H_9 . В этом случае $\mathcal{U} = [2, 4, 7, 9]$ и $K = 2^4 - 1 = 15$, то есть из множества элементов списка \mathcal{U} может быть сформировано 15 различных непустых подмножеств. Найдем, например, пятое подмножество. Функция $\sigma(\cdot)$ преобразует число 5 в двоичную запись из 4-х разрядов 0101 и возвращает в качестве результата подмножество $\{4, 9\}$. Таким образом функция высшего порядка $Map(F_{\mathbf{u}}, \mathcal{L}_{map})$ преобразует список \mathcal{L}_{map} номеров подмножеств в список пар (\mathbf{d}_j, f_j) :

$$Map(F_{\mathbf{u}}, \mathcal{L}_{map}) = [F_{\mathbf{u}}(1), \dots, F_{\mathbf{u}}(K)] = [(\mathbf{d}_1, f_1), \dots, (\mathbf{d}_K, f_K)].$$

Здесь \mathbf{d}_j ($j = 1, \dots, K$) является единичным вектором движения, а f_j — значение целевой функции, которое достигается в точке $\mathbf{u} + \mathbf{d}_j$.

Обозначим через \mathcal{L}_{reduce} список пар, генерируемый функцией высшего порядка Map :

$$\mathcal{L}_{reduce} = Map(F_{\mathbf{u}}, \mathcal{L}_{map}) = [(\mathbf{d}_1, f_1), \dots, (\mathbf{d}_K, f_K)].$$

Определим бинарную ассоциативную операцию

$$\oplus : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^n \times \mathbb{R},$$

являющуюся первым параметром функции высшего порядка $Reduce$:

$$(\mathbf{d}', f') \oplus (\mathbf{d}'', f'') = \begin{cases} (\mathbf{d}', f'), & \text{если } f' \geq f''; \\ (\mathbf{d}'', f''), & \text{если } f' < f''. \end{cases} \quad (37)$$

Функция высшего порядка $Reduce$ редуцирует список пар \mathcal{L}_{reduce} к одной паре путем последовательного применения операции \oplus ко всем элементам списка:

$$Reduce(\oplus, \mathcal{L}_{reduce}) = (\mathbf{d}_1, f_1) \oplus \dots \oplus (\mathbf{d}_K, f_K) = (\mathbf{d}_{j'}, f_{j'}),$$

где согласно (37)

$$j' = \arg \max_{1 \leq j \leq K} f_j.$$

Параллельная работа алгоритма 4 организована по схеме «мастер–рабочие» и включает в себя $L + 1$ процесс: один процесс–мастер и L процессов–рабочих. Процесс–мастер осуществляет общее управление вычислениями, распределяет работу между процессами–рабочими, получает от них результаты и формирует итоговый результат. Для простоты

будем предполагать, что количество подмножеств K кратно количеству рабочих L . На шаге 1 мастер вводит исходные данные задачи ЛП и начальную точку \mathbf{u}_0 . На шаге 2 мастер устанавливает счетчик итераций k в значение ноль. Шаги 3–26 реализуют основной цикл **repeat/until**, вычисляющий решение задачи ЛП (1). На шаге 4 мастер рассылает текущее приближение \mathbf{u}_k всем рабочим. На шаге 16 он получает от рабочих частичные результаты, которые на шаге 17 редуцируются в пару (\mathbf{d}_k, f_k) . Если на шаге 18 выполняется условие $\mathbf{d}_k = \mathbf{0}$, то решение найдено (мы предполагаем $\mathbf{d}_0 \neq \mathbf{0}$). В этом случае на шаге 19 мастер присваивает логической переменной *exit* значение **true**. Если $\mathbf{d}_k \neq \mathbf{0}$, то мастер на шаге 21 вычисляет следующее приближение \mathbf{u}_{k+1} , увеличивает на шаге 22 счетчик итераций k на единицу и на шаге 23 присваивает логической переменной *exit* значение **false**. На шаге 25 мастер рассылает всем рабочим значение логической переменной *exit*. Если логическая переменная *exit* принимает значение «истина», цикл **repeat/until** завершается на шаге 26. На шаге 27 мастер выводит в качестве результата последнее приближение \mathbf{u}_k и оптимальное значение целевой функции f_k . Шаг 28 завершает работу процесса–мастера.

Все рабочие выполняют один и тот же код, но над различными данными. На шаге 1 l -тый рабочий ($l = 1, \dots, L$) вводит исходные данные задачи ЛП. Цикл **repeat/until** рабочего (шаги 3–26) соответствует циклу **repeat/until** мастера. На шаге 4 l -тый рабочий получает от мастера текущее приближение \mathbf{u}_k . Затем он формирует подсписок $\mathcal{L}_{map(l)}$ своих порядковых номеров подмножеств для последующей обработки (шаги 5–13). Подписки различных рабочих не пересекаются:

$$l' \neq l'' \Leftrightarrow \mathcal{L}_{map(l')} \neq \mathcal{L}_{map(l'')}, \quad (38)$$

а их конкатенация дает полный список:

$$\mathcal{L}_{map} = \mathcal{L}_{map(0)} \# \dots \# \mathcal{L}_{map(L-1)}. \quad (39)$$

На шаге 14 рабочий вызывает функцию высшего порядка *Map*, которая, в свою очередь, применяет параметризованную функцию $F_{\mathbf{u}_k}$, определенную формулами (35), ко всем элементам подсписка $\mathcal{L}_{map(l)}$, формируя на выходе подсписок пар $\mathcal{L}_{reduce(l)}$. Этот подсписок на шаге 15 редуцируется рабочим в единственную пару (\mathbf{d}_l, f_l) с помощью функции высшего порядка *Reduce*, которая последовательно применяет бинарную операцию \oplus , определенную по формуле (37), ко всем элементам подсписка $\mathcal{L}_{reduce(l)}$. Результат пересылается мастеру на шаге 16. На шаге 25 рабочий получает от мастера значение логической переменной *exit*. Если эта переменная принимает значение **true**, то рабочий процесс завершается. В противном случае продолжает выполняться цикл **repeat/until**. Операторы обмена **Broadcast**, **Gather**, **RecvFromMaster** и **SendToMaster** обеспечивают неявную синхронизацию работы процесса–мастера и процессов–рабочих.

5. Вычислительные эксперименты

Мы реализовали параллельную версию алгоритма AlFaMove на языке C++ с использованием программного BSF-каркаса [15], базирующегося на модели параллельных вычислений BSF [14]. BSF-каркас инкапсулирует все аспекты, связанные с распараллеливанием программы на основе библиотеки MPI. Исходные коды параллельной реализации алгоритма AlFaMove решения задач ЛП свободно доступны в репозитории GitHub по адресу <https://github.com/leonid-sokolinsky/AlFaMove>. Разработанная программа была протестирована на большом количестве задач ЛП, взятых из различных источников. Все эти

Таблица 1. Характеристики кластера «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

задачи в формате MTX [16] доступны по адресу <https://github.com/leonid-sokolinsky/Set-of-LP-Problems>. В качестве тестов мы также использовали искусственные задачи, полученные с помощью генератора случайных задач ЛП FRaGenLP [17]. Эти задачи доступны по адресу <https://github.com/leonid-sokolinsky/Set-of-LP-Problems/tree/main/Rnd-LP>. Мы не смогли протестировать реализацию AlFaMove на задачах из репозитория Netlib-LP [18], так как во всех этих задачах количество гиперплоскостей, проходящих через начальную точку \mathbf{u}_0 , превышало число 30, что соответствует количеству возможных комбинаций, равному 1 073 741 824. Массивы таких размеров не допускаются доступными нам компиляторами C++.

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

$$\left\{ \begin{array}{ll} x_1 & \leq 200 \\ & x_2 \leq 200 \\ & \vdots \\ & x_n \leq 200 \\ x_1 + x_2 + \dots + x_n & \leq 200(n - 1) + 100 \\ x_1 \geq 0, x_2 \geq 0, \dots, x_n \geq 0. \end{array} \right. \quad (40)$$

Градиент целевой функции задается вектором

$$\mathbf{c} = (1, 2, \dots, n). \quad (41)$$

Задача предполагает нахождение максимума целевой функции и имеет единственное решение в точке $(100, 200, \dots, 200)$ со значением целевой функции, равным $100(n^2 + n - 1)$. Для произвольного n эта задача может быть получена в формате MTX с помощью генератора FRaGenLP, если в качестве количества случайных неравенств задать 0. Эти задачи ЛП для различных n доступны по адресу <https://github.com/leonid-sokolinsky/Set-of-LP-Problems/tree/main/Rnd-LP> под именами `lp_rnd<n>-0`, где в качестве `<n>` указана размерность пространства. Вычислительные эксперименты проводились на суперкомпьютере «Торнадо ЮУрГУ» [19], характеристики которого представлены в табл. 1. Все вычисления выполнялись с двойной точностью, при которой число с плавающей точкой занимает в оперативной памяти 64 бита.

В первой серии экспериментов исследовалась зависимость ускорения и эффективности распараллеливания алгоритма AlFaMove от количества используемых процессорных узлов

кластера при решении задач «гиперкуб с отсеченной вершиной». Результаты этих экспериментов представлены на рис. 6. В данном контексте ускорение $\alpha(L)$ определялось как отношение времени $T(1)$ решения задачи на конфигурации с узлом-мастером и единственным узлом-рабочим ко времени $T(L)$ решения той же задачи на конфигурации с узлом-мастером и L узлами-рабочими:

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

Эффективность распараллеливания $\beta(L)$ вычислялась по формуле

$$\beta(L) = \frac{T(1)}{L \cdot T(L)}.$$

Вычисления проводились для размерностей 16, 18 и 20. Число ограничений соответственно составило 33, 37 и 41. В качестве начальной точки всегда выбиралась вершина допустимого многогранника со следующими координатами:

$$x_1 = 0, \quad \dots \quad x_{n/2} = 0, \quad x_{n/2+1} = 200, \quad \dots \quad x_n = 200. \quad (42)$$

Эксперименты продемонстрировали хорошую масштабируемость алгоритма AlFaMove на задачах «гиперкуб с отсеченной вершиной», начиная с размерности $n = 18$. В этом случае алгоритм демонстрировал ускорение, близкое к линейному. На меньших размерностях затраты на обмены и латентность начинают превалировать над вычислительными затратами, что приводит к резкому уменьшению границы масштабируемости алгоритма⁵. Для $n = 16$ эта граница оказалась равной 180 узлам. Эксперименты также показали, что с увеличением размерности задачи эффективность распараллеливания на малом количестве процессорных узлов (меньше 120) падает. Однако, при большем количестве процессорных узлов наблюдается обратная тенденция. Так, для размерности $n = 16$ эффективность распараллеливания на 20 процессорных узлах составила 61%, после чего снизилась до 23% на 220 узлах. При этом, для $n = 20$ эффективность распараллеливания оказалась равной 50% и 40% соответственно.

⁵Под границей масштабируемости понимается максимальное количество процессорных узлов, на котором наблюдается рост ускорения.

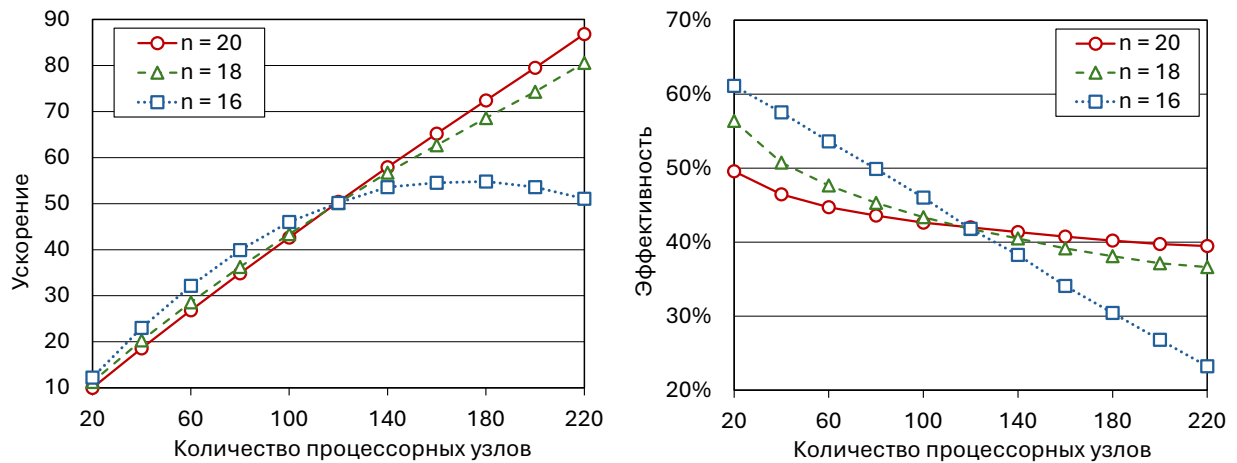


Рис. 6. Ускорение и эффективность распараллеливания алгоритма AlFaMove.

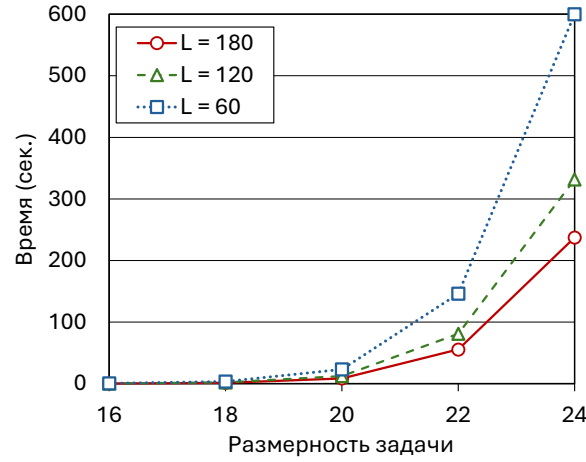


Рис. 7. Зависимость времени работы AlFaMove от размерности задачи на различных многопроцессорных конфигурациях (L — количество рабочих узлов).

В следующей серии экспериментов исследовалась зависимость времени вычислений от размерности задачи «гиперкуб с отсеченной вершиной» на различных многопроцессорных конфигурациях с количеством рабочих узлов $L = 60$, $L = 120$ и $L = 180$. Результаты этих экспериментов представлены на рис. 7. Размерность варьировалась от $n = 16$ до $n = 24$ с шагом 2. Для размерности $n = 24$ каждый из списков \mathcal{L}_{map} и \mathcal{L}_{reduce} включал в себя 16 777 215 элементов. Это — максимальный размер, допускаемый используемым компилятором. В качестве начальной точки всегда выбиралась вершина допустимого многогранника с координатами (42). На всех исследуемых конфигурациях эксперименты показали экспоненциальный рост времени вычислений с увеличением размерности задачи. Однако, конфигурации с большим количеством рабочих узлов демонстрировали существенно меньшее время работы алгоритма AlFaMove.

В третьей серии экспериментов мы исследовали поведение алгоритма AlFaMove на кубе Кле-Минти (Klee-Minty), представляющем собой параметризованную задачу ЛП с размерностью пространства в качестве параметра. Область допустимых решений этой задачи представляет собой гиперкуб с перекошенными углами и может быть задана следующей системой неравенств:

$$\begin{cases} x_1 & \leq 5 \\ 4x_1 + x_2 & \leq 25 \\ 8x_1 + 4x_2 + x_3 & \leq 125 \\ & \vdots \\ 2^n x_1 + 2^{n-1} x_2 + \dots + 4x_{n-1} + x_n & \leq 5^n \\ x_1 \geq 0, x_2 \geq 0, \dots, x_n \geq 0. \end{cases}$$

Градиент целевой функции задается вектором

$$\mathbf{c} = (2^{n-1}, 2^{n-2}, \dots, 2, 1).$$

Задача предполагает нахождение максимума целевой функции и имеет единственное решение в точке $(0, \dots, 0, 5^n)$ со значением целевой функции, равным 5^n . Кле и Минти в статье [20] показали, что в случае, когда начальной вершиной является центр координат, классический симплекс-метод при решении этой задачи посещает все 2^n вершин, делая

$2^n - 1$ итераций. Известно, что большинство оптимизационных алгоритмов демонстрируют плохую производительность применительно к кубу Кле-Минти. Мы применили алгоритм AlFaMove для решения кубов Кле-Минти размерности от 5 до 9. Результаты экспериментов, приведенные в табл. 2, показывают, что алгоритм AlFaMove во всех случаях находил решение за $2n - 1$ итераций, в то время, как симплекс-метод совершал $2^n - 1$ итераций.

Таблица 2. Эксперименты с кубами Кле-Минти.

Размер- ность	AlFaMove				Simplex
	Граница масшт-ти	Время (сек.)	Относит. погреш-ть	Кол-во итераций	Кол-во итераций
5	10	0.2	$0.9 \cdot 10^{-12}$	9	31
6	15	2	$0.2 \cdot 10^{-12}$	11	63
7	20	13	$0.8 \cdot 10^{-11}$	13	127
8	25	126	$0.8 \cdot 10^{-11}$	15	255
9	30	1445	$0.2 \cdot 10^{-10}$	17	511

Относительная погрешность вычислялась по формуле

$$\delta = \left| \frac{f_{exact} - f_{approx}}{f_{exact}} \right|,$$

где f_{exact} — точное максимальное значение целевой функции, f_{approx} — значение, вычисленное алгоритмом AlFaMove. Подсчет итераций симплекс-метода производился с помощью онлайн калькулятора, доступного по адресу <https://www.pmc calculators.com/simplex-method-calculator>. Эксперименты также показали, что порог масштабируемости алгоритм AlFaMove на кубах Кле-Минти рос линейно с ростом размерности. При этом одновременно наблюдался экспоненциальный рост времени вычислений.

Заключение

В статье представлен алгоритм AlFaMove, являющийся численной реализацией метода поверхностного движения для решения задач линейного программирования. Ключевой особенностью этого метода является построение оптимального пути на поверхности допустимого многогранника от начальной точки к решению задачи линейного программирования. Под оптимальным путем понимается путь по поверхности допустимой области в направлении максимального увеличения значений целевой функции. Практическая значимость предложенного метода состоит в том, что он открывает возможность применения искусственных нейронных сетей прямого распространения для решения нестационарных многомерных задач линейного программирования в режиме реального времени.

Теоретической основой алгоритма AlFaMove является операция построения псевдопроекции на линейные многообразия, которые формируют грани допустимого многогранника. Псевдопроекция реализуется на основе фейеровского процесса и является обобщением понятий ортогональной проекции на линейное многообразие и метрической проекции на выпуклое множество. В случае грани, образуемой гиперплоскостью, псевдопроекция сводится к ортогональной проекции. Доказано, что путь по гиперплоскости, построенный с помощью градиента целевой функции и ортогональной проекции, является оптимальным. При-

веден алгоритм проекционного типа для построения псевдопроекции на линейные многообразия меньших размерностей, образуемые пересечением гиперплоскостей. Доказано, что в этом случае псевдопроекция совпадает с ортогональной проекцией. Представлено формализованное описание алгоритма AlFaMove, строящего оптимальный путь на поверхности допустимого многогранника. В основе алгоритма AlFaMove лежит процедура вычисления вектора движения по грани допустимого многогранника из точки текущего приближения в направлении максимального увеличения значения целевой функции. Дано формализованное описание этой процедуры.

Алгоритмы проекционного типа характеризуются низкой скоростью сходимости, зависящей от углов между гиперплоскостями, образующими линейное многообразие. Также отмечено, что при вычислении вектора движения возникает переборная задача комбинаторного типа, имеющая высокую пространственную и временную сложность. Представлена параллельная версия алгоритма AlFaMove, ориентированная на кластерные вычислительные системы. Параллельная версия реализована на языке C++ с использованием программного BSF-каркаса, основанного на модели параллельных вычислений BSF. Проведены эксперименты по исследованию масштабируемости алгоритма AlFaMove на кластерной вычислительной системе. Вычислительные эксперименты показали, что задача линейного программирования с 24 переменными и 49 ограничениями демонстрирует ускорение близкое к линейному на 320 процессорных узлах кластера. Задачи большей размерности приводили к ошибке компилятора, связанной с превышением максимального допустимого размера массивов.

В качестве направлений дальнейших исследований выделим следующие. Мы планируем разработать новый, более эффективный метод построения пути на поверхности допустимого многогранника, приводящего к решению задачи линейного программирования. Основная идея состоит в сокращении перебираемых комбинаций гиперплоскостей при определении направления движения. Это может быть достигнуто, если мы ограничимся перемещениями только по ребрам многогранника (линейным многообразиям размерности один). Проблема пространственной сложности может быть решена путем перехода к стохастическим методам выбора направления движения.

Исследование выполнено при финансовой поддержке РНФ (проект № 23-21-00356).

Литература

1. Optimization in Large Scale Problems: Industry 4.0 and Society 5.0 Applications / ed. by M. Fathi, M. Khakifirooz, P.M. Pardalos. Cham, Switzerland: Springer, 2019. XI, 340 p. DOI: 10.1007/978-3-030-28565-4.
2. Kopanos G.M., Puigjaner L. Solving Large-Scale Production Scheduling and Planning in the Process Industries. Cham, Switzerland: Springer, 2019. 1–291 p. DOI: 10.1007/978-3-030-01183-3.
3. Schlenkrich M., Parragh S.N. Solving large scale industrial production scheduling problems with complex constraints: an overview of the state-of-the-art // 4th International Conference on Industry 4.0 and Smart Manufacturing. Procedia Computer Science. Vol. 217 / ed. by F. Longo, M. Affenzeller, A. Padovano, W. Shen. Elsevier, 2023. P. 1028–1037. DOI: 10.1016/J.PROCS.2022.12.301.

4. Соколинская И.М., Соколинский Л.Б. О решении задачи линейного программирования в эпоху больших данных // Параллельные вычислительные технологии (ПаВТ'2017). Короткие статьи и описания плакатов. Челябинск: Издательский центр ЮУрГУ, 2017. С. 471—484. URL: <http://omega.sp.susu.ru/pavt2017/short/014.pdf>.
5. Branke J. Optimization in Dynamic Environments // Evolutionary Optimization in Dynamic Environments. Genetic Algorithms and Evolutionary Computation, vol. 3. Boston, MA: Springer, 2002. P. 13–29. DOI: 10.1007/978-1-4615-0911-0_2.
6. Dantzig G.B. Linear programming and extensions. Princeton, N.J.: Princeton university press, 1998. 656 p.
7. Зоркальцев В.И., Мокрый И.В. Алгоритмы внутренних точек в линейной оптимизации // Сибирский журнал индустриальной математики. 2018. Т. 21, 1 (73). С. 11–20. DOI: 10.17377/sibjim.2018.21.102.
8. 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. City: Cham: Springer, 2015. P. 281–307. DOI: 10.1007/978-3-319-24024-4_17.
9. 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.
10. 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.
11. Мальцев А.И. Основы линейной алгебры. Москва: Наука. Главная редакция физико-математической литературы, 1970. 402 с.
12. Васин В.В., Ерёмин И.И. Операторы и итерационные процессы фейеровского типа. Теория и приложения. Екатеринбург: УрО РАН, 2005. 211 с.
13. Gould N.I. How good are projection methods for convex feasibility problems? // Computational Optimization and Applications. 2008. Vol. 40, no. 1. P. 1–12. DOI: 10.1007/S10589-007-9073-5.
14. 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.
15. 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.
16. 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>.

17. 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.
18. Gay D.M. Electronic mail distribution of linear programming test problems // Mathematical Programming Society COAL Bulletin. 1985. Vol. 13. P. 10–12.
19. Dolganina N., Ivanova E., Bilenko R., Rekachinsky A. HPC Resources of South Ural State University // Parallel Computational Technologies. PCT 2022. Communications in Computer and Information Science, vol. 1618 / ed. by L. Sokolinsky, M. Zymbler. City: Cham: Springer, 2022. P. 43–55. DOI: 10.1007/978-3-031-11623-0_4.
20. Klee V., Minty G.J. How good is the simplex algorithm? // Inequalities - III. Proceedings of the Third Symposium on Inequalities Held at the University of California, Los Angeles, Sept. 1-9, 1969 / ed. by O. Shisha. New York-London: Academic Press, 1972. P. 159–175.

Ольховский Николай Александрович, аспирант, кафедра системного программирования, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

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

DOI: 10.14529/cmse240Y0Z

NUMERICAL IMPLEMENTATION OF SURFACE MOVEMENT METHOD IN LINEAR PROGRAMMING

© 2024 L.B. Sokolinsky, N.A. Olkhovsky, I.M. Sokolinskaya

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

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

Received: 10.06.2024

The article is devoted to the numerical implementation of a new linear programming method called the surface movement method. The implementation is based on the AlFaMove algorithm, which builds, on the surface of the feasible polytope, the optimal objective path from an arbitrary boundary point to a point that is a solution to the linear programming problem. The optimality of the path lies in the fact that the direction of movement along the face of the polytope corresponds to the maximum increase in the value of the objective function. To calculate the optimal direction of movement, a method based on the operation of pseudoprojection onto a linear manifold is used. The pseudoprojection operation is a generalization of the notion of orthogonal projection. Pseudo projection is implemented using an iterative projection-type algorithm. The proposition is proved that the pseudoprojection coincides with the orthogonal projection in the case of a linear manifold that is the intersection of hyperplanes. It is also proved that, in the case of a linear manifold, the pseudoprojection method calculates a movement vector that coincides with the direction of maximum increase of the objective function. A parallel implementation of the AlFaMove algorithm is performed. The results of computational experiments on a cluster computing system are presented, which demonstrate the high scalability of the proposed numerical implementation.

Keywords: linear programming, surface movement method, numerical implementation, AlFaMove algorithm, parallel implementation, cluster computing system, scalability evaluation.

FOR CITATION

Sokolinsky L.B., Olkhovsky N.A., Sokolinskaya I.M. Numerical Implementation of Surface Movement Method in Linear Programming. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2024. Vol. X, no. Y. P. Z1–Z2. (in Russian) DOI: 10.14529/cmse240Y0Z.

This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 4.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.

References

1. Optimization in Large Scale Problems: Industry 4.0 and Society 5.0 Applications / ed. by M. Fathi, M. Khakifirooz, P.M. Pardalos. Cham, Switzerland: Springer, 2019. XI, 340 p. DOI: 10.1007/978-3-030-28565-4.
2. Kopanos G.M., Puigjaner L. Solving Large-Scale Production Scheduling and Planning in the Process Industries. Cham, Switzerland: Springer, 2019. 1–291 p. DOI: 10.1007/978-3-030-01183-3.
3. Schlenkrich M., Parragh S.N. Solving large scale industrial production scheduling problems with complex constraints: an overview of the state-of-the-art. 4th International Conference on Industry 4.0 and Smart Manufacturing. Procedia Computer Science. Vol. 217 / ed. by F. Longo, M. Affenzeller, A. Padovano, W. Shen. Elsevier, 2023. P. 1028–1037. DOI: 10.1016/J.PROCS.2022.12.301.
4. Sokolinskaya I.M., Sokolinsky L.B. On the Solution of Linear Programming Problems in the Age of Big Data. Parallel Computational Technologies. PCT 2017. Communications in Computer and Information Science, vol. 753. / ed. by L. Sokolinsky, M. Zymbler. Cham, Switzerland: Springer, 2017. P. 86–100. DOI: 10.1007/978-3-319-67035-5_7.
5. Branke J. Optimization in Dynamic Environments. Evolutionary Optimization in Dynamic Environments. Genetic Algorithms and Evolutionary Computation, vol. 3. Boston, MA: Springer, 2002. P. 13–29. DOI: 10.1007/978-1-4615-0911-0_2.
6. Dantzig G.B. Linear programming and extensions. Princeton, N.J.: Princeton university press, 1998. 656 p.
7. Zorkaltsev V., Mokryi I. Interior point algorithms in linear optimization. Journal of applied and industrial mathematics. 2018. Vol. 12, no. 1. P. 191–199. DOI: 10.1134/S1990478918010179.
8. 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. City: Cham: Springer, 2015. P. 281–307. DOI: 10.1007/978-3-319-24024-4_17.
9. Olkhovsky N.A., Sokolinsky L.B. Surface Movement Method for Linear Programming. 2024. DOI: 10.48550/arXiv.2404.12640. arXiv: 2404.12640.
10. Sokolinsky L.B., Sokolinskaya I.M. Apex Method: A New Scalable Iterative Method for Linear Programming. Mathematics. 2023. T. 11, № 7. C. 1654. DOI: 10.3390/MATH11071654.

11. Maltsev A. The basics of linear algebra. Moskow: Science. The main editorial office of the phys-math literature, 1970. 402 p. (in Russian).
12. Vasin V.V., Eremin I.I. Operators and Iterative Processes of Fejér Type. Theory and Applications. Berlin, New York: Walter de Gruyter, 2009. 155 p. Inverse and III-Posed Problems Series. DOI: 10.1515/9783110218190.
13. Gould N.I. How good are projection methods for convex feasibility problems?. Computational Optimization and Applications. 2008. Vol. 40, no. 1. P. 1–12. DOI: 10.1007/S10589-007-9073-5.
14. 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.
15. 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.
16. 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>.
17. 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. City: Cham: Springer, 2021. P. 164–177. DOI: 10.1007/978-3-030-81691-9_12.
18. Gay D.M. Electronic mail distribution of linear programming test problems. Mathematical Programming Society COAL Bulletin. 1985. Vol. 13. P. 10–12.
19. Dolganina N., Ivanova E., Bilenko R., Rekachinsky A. HPC Resources of South Ural State University. Parallel Computational Technologies. PCT 2022. Communications in Computer and Information Science, vol. 1618 / ed. by L. Sokolinsky, M. Zymbler. City: Cham: Springer, 2022. P. 43–55. DOI: 10.1007/978-3-031-11623-0_4.
20. Klee V., Minty G.J. How good is the simplex algorithm?. Inequalities - III. Proceedings of the Third Symposium on Inequalities Held at the University of California, Los Angeles, Sept. 1-9, 1969 / ed. by O. Shisha. New York-London: Academic Press, 1972. P. 159–175.