Исследование масштабируемости параллельной реализации алгоритма AlFaMove для линейного программирования на кластерной вычислительной системе*

Н.А. Ольховский, Л.Б. Соколинский

Южно-Уральский государственный университет
(национальный исследовательский университет)

Работа посвящена параллельной реализации нового алгоритма линейного программирования, получившего название AlFaMove. Алгоритм 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 при поиске точки максимума целевой функции выбирается направление движения по грани многогранника, соответствующее максимальному увеличению значения целевой функции. Для вычисления направления движения используется итерационный алгоритм проекционного типа. Выполнена параллельная реализация алгоритма AlFaMove. Приведены результаты вычислительных экспериментов на кластерной вычислительной системе, демонстрирующие высокую масштабируемость предложенной реализации.

Ключевые слова: линейное программирование, AlFaMove, алгоритм движения по граням, параллельная реализация, кластерная вычислительная система, исследование масштабируемости.

1. 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 больших задач ЛП являются симплекс-метод [6] и метод 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

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

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 соответствие 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 многогранника 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 2 содержит теоретический базис, необходимый для описания метода поверхностного движения и его численной реализации. Раздел 3 посвящен описанию операции псевдопроекции, позволяющий найти вектор движения по оптимальному целевому пути для линейного многообразия, получаемого в результате пересечением гиперплоскостей. В разделе 4 дается формализованное описание алгоритма AlFaMove, представляющего собой численную реализацию метода поверхностного движения. Раздел 5 посвящен описанию параллельной версии алгоритма AlFaMove. В разделе 6 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.

2. Теоретический базис

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

$$\bar{\boldsymbol{x}} = \arg\max_{\boldsymbol{x}\in\mathbb{R}^n} \{\langle \boldsymbol{c}, \boldsymbol{x}\rangle \,|\, A\boldsymbol{x} \leqslant \boldsymbol{b}\}, \tag{1}$$

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

$$-\boldsymbol{x} \leqslant \boldsymbol{0}.$$

Линейная целевая функция задачи (1) имеет вид

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

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

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

$$\hat{H}_i = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_i, \boldsymbol{x}\rangle \leqslant b_i\}; \tag{2}$$

$$H_i = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_i, \boldsymbol{x}\rangle = b_i\}. \tag{3}$$

Определим допустимый многогранник

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

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

$$\forall \boldsymbol{x} \in H_i, \forall \lambda > 0 : \boldsymbol{x} + \lambda \boldsymbol{c} \notin \hat{H}_i. \tag{5}$$

Геометрический смысл этого определения состоит в том, что луч, исходящий в направлении вектора $\boldsymbol{c}$ 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:

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

Определим

$$\mathcal{I} = \{ i \in \{1, \ldots, m\} \,|\langle \boldsymbol{a}_i, \boldsymbol{c} \rangle > 0 \}, \tag{6}$$

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

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

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 $\hat{M}$[1]. Согласно утверждению 3 в [10] имеем

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

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

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

Ортогональной проекцией точки $\boldsymbol{v} \in \mathbb{R}^n$ на гиперплоскость H называется отображение $\boldsymbol{\pi}_H$, определяемое формулой

$$\boldsymbol{\pi}_H(\boldsymbol{v}) = \boldsymbol{v} - \frac{\langle \boldsymbol{a}, \boldsymbol{v} \rangle - b}{\|\boldsymbol{a}\|^2} \boldsymbol{a}. \tag{8}$$

Следующее утверждение дает способ вычисления оптимального целевого пути на гиперплоскости.

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

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

[1] Под 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.

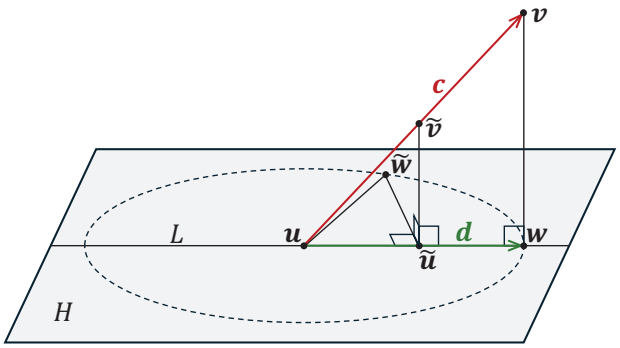

Рис. 1. Иллюстрация к доказательству предложения 1.
Пунктиром обозначена гиперокружность с радиусом $\|\boldsymbol{w}-\boldsymbol{u}\|$.

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

$$f(\boldsymbol{x}) = \langle \boldsymbol{c}, \boldsymbol{x} \rangle . \tag{10}$$

Пусть векторы $\boldsymbol{a}$ и $\boldsymbol{c}$ линейно независимы (не коллинеарны, и среди них нет нулевого вектора). Положим

$$\boldsymbol{v} = \boldsymbol{u} + \boldsymbol{c}. \tag{11}$$

Построим ортогональную проекцию $\boldsymbol{\pi}_H(\boldsymbol{v})$ точки $\boldsymbol{v}$ на гиперплоскость H:

$$\boldsymbol{w} = \boldsymbol{\pi}_H(\boldsymbol{v}). \tag{12}$$

Тогда вектор $\boldsymbol{d} = \boldsymbol{w} - \boldsymbol{u}$ однозначно задает направление максимального увеличения целевой функции $f(\boldsymbol{x})$, определяемой формулой (10).

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

$$\langle \boldsymbol{c}, \tilde{\boldsymbol{w}} \rangle \geqslant \langle \boldsymbol{c}, \boldsymbol{w} \rangle , \tag{13}$$

$\|\tilde{\boldsymbol{w}} - \boldsymbol{u}\| = \|\boldsymbol{w} - \boldsymbol{u}\|$ и $\tilde{\boldsymbol{w}} \neq \boldsymbol{w}$ (см. рис. 1). Здесь и далее $\|\cdot\|$ обозначает евклидову норму.

Вычислим $\langle \boldsymbol{c}, \boldsymbol{w} \rangle$. В соответствии с определением 2 ортогональная проекция $\boldsymbol{\pi}_H(\boldsymbol{v})$ точки $\boldsymbol{v}$ на гиперплоскость H, задаваемую формулой (9), вычисляется следующим образом:

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

Подставив вместо $\boldsymbol{v}$ правую часть формулы (11) получим

$$\boldsymbol{w} = \boldsymbol{u} + \boldsymbol{c} - \frac{\langle \boldsymbol{a}, \boldsymbol{c} \rangle}{\|\boldsymbol{a}\|^2} \boldsymbol{a}. \tag{14}$$

Используя (14), находим

$$\langle \boldsymbol{c}, \boldsymbol{w} \rangle = \langle \boldsymbol{c}, \boldsymbol{u} \rangle + \|\boldsymbol{c}\|^2 - \frac{\langle \boldsymbol{a}, \boldsymbol{c} \rangle^2}{\|\boldsymbol{a}\|^2}. \tag{15}$$

Поскольку $\boldsymbol{a}$ и $\boldsymbol{c}$ линейно независимы, в соответствии с неравенством Коши — Буняковского имеем

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

Отсюда следует

$$\|\boldsymbol{c}\|^2 - \frac{\langle \boldsymbol{a}, \boldsymbol{c} \rangle^2}{\|\boldsymbol{a}\|^2} > 0. \tag{16}$$

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

$$-1 \leqslant \delta < 1, \tag{17}$$

такое, что

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

Положим

$$\tilde{\boldsymbol{v}} = \boldsymbol{u} + \delta(\boldsymbol{v} - \boldsymbol{u}). \tag{18}$$

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

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

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

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

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

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

Очевидно

$$\tilde{\boldsymbol{w}} = (\tilde{\boldsymbol{w}} - \tilde{\boldsymbol{u}}) + \tilde{\boldsymbol{u}}. \tag{20}$$

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

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

Отсюда получаем

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

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

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

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

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

что противоречит (13). □

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

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

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

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

Require: $H_i = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_i, \boldsymbol{x} \rangle = b_i\}$; $\mathcal{J} \subseteq \{1, \ldots, m\}$; $\mathcal{J} \neq \emptyset$; $\bigcap\limits_{i \in \mathcal{J}} H_i \neq \emptyset$

function $\boldsymbol{\rho}_{\mathcal{J}}(\boldsymbol{x})$
  $k := 0$
  $\boldsymbol{x}_0 := \boldsymbol{x}$
  repeat
    $\Sigma := 0$
    for $i \in \mathcal{J}$ **do**
      $\Sigma := \Sigma + (\langle \boldsymbol{a}_i, \boldsymbol{x}_k \rangle - b_i)\boldsymbol{a}_i / \|\boldsymbol{a}_i\|^2$
    end for
    $\boldsymbol{x}_{(k+1)} := \boldsymbol{x}_k - \Sigma / |\mathcal{J}|$
    $\xi_{max} := 0$ ▷ Максимальная невязка
    for $i \in \mathcal{J}$ **do**
      $\xi_i := \| \langle \boldsymbol{a}_i, \boldsymbol{x}_{k+1} \rangle - b_i \|$
      if $\xi_i > \xi_{max}$ **then**
        $\xi_{max} := \xi_i$
      end if
    end for
    $k := k + 1$
  until $\xi_{max} < \epsilon_\xi$ ▷ Малый параметр $\epsilon_\xi > 0$
  return $\boldsymbol{x}_k$
end function

3. Операция псевдопроектирования на линейное многообразие

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

$$L = \bigcap_{i \in \mathcal{J}} H_i. \tag{23}$$

Обозначим k_L — размерность линейного многообразия L. При $k_L < n - 1$ многообразие L не является гиперплоскостью, и для определения вектора движения $\boldsymbol{d}$ 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 $\boldsymbol{d}$ с помощью операции псевдопроектирования [10]. Определим проекционное отображение $\boldsymbol{\varphi}(\cdot)$:

$$\boldsymbol{\varphi}(\boldsymbol{x}) = \frac{1}{|\mathcal{J}|} \sum_{i \in \mathcal{J}} \boldsymbol{\pi}_{H_i}(\boldsymbol{x}). \tag{24}$$

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

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

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

$$\boldsymbol{x}_k \to \tilde{\boldsymbol{x}} \in L.$$

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

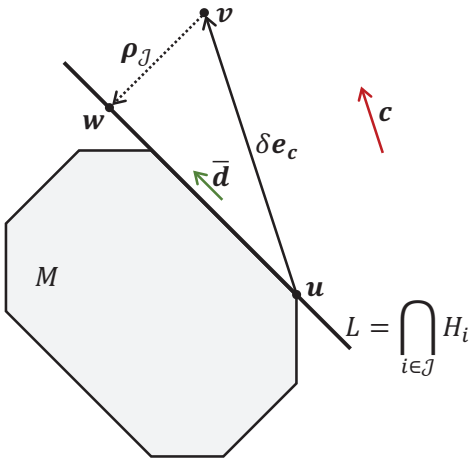

Рис. 2. Вычисление вектора направления движения $\bar{\boldsymbol{d}}$ по грани линейного многообразия L.

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

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

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 $\boldsymbol{x}_0$. Шаг 4 открывает итерационный цикл вычисления псевдопроекции. Шаги 5–8 для текущего приближения $\boldsymbol{x}_k$ вычисляют сумму из правой части формулы (24). Шаг 9 находит следующее приближение $\boldsymbol{x}_{k+1}$. 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 $\boldsymbol{x}_k$.

4. 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 $\boldsymbol{u}^{(0)} \in M \cap \Gamma(\hat{M})$ до точки $\bar{\boldsymbol{x}}$, 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 $\boldsymbol{D}(\boldsymbol{u})$ вычисления вектора движения $\bar{\boldsymbol{d}}$ по грани допустимого многогранника M из точки $\boldsymbol{u}$ в направлении максимального увеличения значения целевой функции. Процедура $\boldsymbol{D}(\boldsymbol{u})$ представлена в виде алгоритма 2. Схематично работа этого алгоритма изображена на рис. 2. 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 $\boldsymbol{u}$. На шаге 8 вектору направления движения $\bar{\boldsymbol{d}}$ в качестве начального значения присваивается нулевой вектор. На шаге 9 значению целевой функции f присваивается бесконечно малое число. На шаге 10 строится единичный вектор $\boldsymbol{e_c}$, сонаправленный с вектором $\boldsymbol{c}$. На шаге 11 строится точка $\boldsymbol{v}$ путем прибавления вектора

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

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

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

$\delta \boldsymbol{e_c}$ к вектору $\boldsymbol{u}$ (см. рис. 2). Здесь δ — «большой» положительный параметр: чем больше δ, тем точнее будет вычислен вектор направления движения $\bar{\boldsymbol{d}}$. Однако при увеличении параметра δ будет увеличиваться и время вычисления псевдопроекции $\boldsymbol{\rho}_{\mathcal{J}}(\boldsymbol{v})$ на шаге 13. Цикл **for** на шаге 12 перебирает все возможные комбинации индексов гиперплоскостей, проходящих через точку $\boldsymbol{u}$. Каждой такой комбинации $\mathcal{J}$ соответствует линейное многообразие $L = \bigcap_{i \in \mathcal{J}} H_i$, которое также проходит через точку $\boldsymbol{u}$. Шаг 13 вычисляет точку $\boldsymbol{w}$, выполняя псевдопроекцию точки $\boldsymbol{v}$ на линейное многообразие, соответствующее комбинации $\mathcal{J}$. На шаге 14 вычисляется возможный вектор движения $\boldsymbol{d}$ по текущему линейному многообразию в направлении максимального увеличения значений целевой функции. Шаг 15 вычисляет единичный вектор $\boldsymbol{e_d}$, сонаправленный с вектором $\boldsymbol{d}$. Шаг 16 проверяет, что малое движение из точки $\boldsymbol{u}$ в направлении $\boldsymbol{e_d}$ не выходит за границы допустимого многогранника. Шаг 17 проверяет, что значение целевой функции в точке $(\boldsymbol{u} + \boldsymbol{e_d})$ 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 $\bar{\boldsymbol{d}}$ (шаг 19). После того, как проверены все возможные комбинации, вектор $\bar{\boldsymbol{d}}$ возвращается в качестве результата (шаг 19). 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 $\boldsymbol{u}$ по поверхности допустимого многогранника локально 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.

Алгоритм 3 Алгоритм движения по граням AlFaMove

Require: $\hat{H}_i = \{\boldsymbol{x} \in \mathbb{R}^n | \langle \boldsymbol{a}_i, \boldsymbol{x} \rangle \leqslant b_i\}$; $M = \bigcap\limits_{i=1}^{m} \hat{H}_i$; $\hat{M} = \bigcap\limits_{i \in \mathcal{I}} \hat{H}_i$; $i \in \mathcal{I} \Leftrightarrow \langle \boldsymbol{a}_i, \boldsymbol{c} \rangle > 0$

1: **input** $\boldsymbol{u}_0$
2: **assert** $\boldsymbol{u}_0 \in M \cap \Gamma(\hat{M})$
3: $\boldsymbol{d}_0 := \boldsymbol{D}(\boldsymbol{u}_0)$
4: **assert** $\boldsymbol{d}_0 \neq \boldsymbol{0}$
5: $k := 0$
6: **repeat**
7: $\quad \boldsymbol{u}_{k+1} := \boldsymbol{u}_k + \mu(\boldsymbol{u}_k, \boldsymbol{d}_k, \epsilon_\mu) \cdot \boldsymbol{d}_k$ ▷ Малый параметр $\epsilon_\mu > 0$
8: $\quad \boldsymbol{d}_{k+1} := \boldsymbol{D}(\boldsymbol{u}_{k+1})$
9: $\quad k := k + 1$
10: **until** $\boldsymbol{d}_k = \boldsymbol{0}$
11: **output** $\boldsymbol{u}_k$ ▷ Решение задачи ЛП (1)
12: **stop**

На шаге 1 вводится начальное приближение $\boldsymbol{u}^{(0)}$. Это может быть произвольная граничная точка рецессивного многогранника $\hat{M}$, удовлетворяющая условию

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

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 $\boldsymbol{d}_0$. Для этого используется функция $\boldsymbol{D}(\cdot)$, реализованная в алгоритме 2. Функция $\boldsymbol{D}(\cdot)$ выдает вектор единичной длины, либо нулевой вектор. Предполагается, что $\boldsymbol{d}_0 \neq \boldsymbol{0}$[2]. Это условие проверяется на шаге 4. На шаге 5 счетчик итераций k устанавливается в значение ноль. Шаг 5 открывает цикл **repeat**, выполняющий движение по граням допустимого многогранника до тех пор, пока очередной вектор движения $\boldsymbol{d}_k$ не окажется нулевым вектором. Это означает, что последнее найденное приближение $\boldsymbol{u}_k$ является решением задачи ЛП (1). Шаг 7 в теле цикла вычисляет следующее приближение $\boldsymbol{u}_{k+1}$ путем прибавления к вектору $\boldsymbol{u}_k$ единичного вектора $\boldsymbol{d}_k$, умноженного на положительное число, вычисляемое функцией $\mu(\boldsymbol{u}_k, \boldsymbol{d}_k, \epsilon_\mu)$, удовлетворяющей следующим двум условиям:

$$\boldsymbol{u}_k + \mu(\boldsymbol{u}_k, \boldsymbol{d}_k, \epsilon_\mu) \cdot \boldsymbol{d}_k \in M;$$
$$\boldsymbol{u}_k + (\mu(\boldsymbol{u}_k, \boldsymbol{d}_k, \epsilon_\mu) + \epsilon_\mu) \cdot \boldsymbol{d}_k \notin M.$$

Здесь ϵ_μ — малый положительный параметр алгоритма. На практике эта функция может быть легко и эффективно реализовано с помощью метода дихотомии. Шаг 8 вычисляет вектор движения $\boldsymbol{d}_{k+1}$ для вновь найденного приближения $\boldsymbol{u}_{k+1}$. 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 $\boldsymbol{u}_k$ в качестве решения задачи ЛП (1). Шаг 12 завершает работу алгоритма AlFaMove.

5. 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

[2]Равенство вектора $\boldsymbol{d}_0$ нулевому вектору означает, что точка $\boldsymbol{u}_0$ принадлежит гиперплоскости, ортогональной целевому вектору $\boldsymbol{c}$.

Алгоритм 4 Параллельная версия алгоритма AlFaMove

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

решении больших задач ЛП она занимает более 90% процессорного времени. Это объясняется тем, что функция вычисления вектора движения $\boldsymbol{D}(\cdot)$, 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 $\boldsymbol{u}$. 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 $\mathcal{L}_{map} = \left[1, \ldots, 2^K\right]$, содержащий порядковые номера всех подмножеств множества $\mathcal{U}$, за исключением пустого. Здесь $K = 2^{|\mathcal{U}|}$. В качестве первого параметра фигурирует параметризованная функция

$$\mathrm{F}_{\boldsymbol{u}} : \left\{1, \ldots, 2^K\right\} \to \mathbb{R}^n \times \mathbb{R},$$

определенная следующим образом:

$$\begin{aligned} &\mathrm{F}_{\boldsymbol{u}}(j) = (\boldsymbol{d}_j, f_j); \\ &\boldsymbol{d}_j = \begin{cases} \boldsymbol{w} - \boldsymbol{u}, & \text{если } (\boldsymbol{u} + \tau \boldsymbol{d}/\|\boldsymbol{d}\|) \in M \wedge \langle \boldsymbol{c}, \boldsymbol{w} \rangle > \langle \boldsymbol{c}, \boldsymbol{u} \rangle; \\ \boldsymbol{0}, & \text{если } (\boldsymbol{u} + \tau \boldsymbol{d}/\|\boldsymbol{d}\|) \notin M \vee \langle \boldsymbol{c}, \boldsymbol{w} \rangle \leqslant \langle \boldsymbol{c}, \boldsymbol{u} \rangle; \end{cases} \\ &f_j = \begin{cases} \langle \boldsymbol{c}, \boldsymbol{w} \rangle, & \text{если } (\boldsymbol{u} + \tau \boldsymbol{d}/\|\boldsymbol{d}\|) \in M \wedge \langle \boldsymbol{c}, \boldsymbol{w} \rangle > \langle \boldsymbol{c}, \boldsymbol{u} \rangle; \\ -\infty, & \text{если } (\boldsymbol{u} + \tau \boldsymbol{d}/\|\boldsymbol{d}\|) \notin M \vee \langle \boldsymbol{c}, \boldsymbol{w} \rangle \leqslant \langle \boldsymbol{c}, \boldsymbol{u} \rangle, \end{cases} \end{aligned} \tag{26}$$

где

$$\boldsymbol{w} = \boldsymbol{\rho}_{\sigma(j)}(\boldsymbol{u} + \delta \boldsymbol{c}/\|\boldsymbol{c}\|). \tag{27}$$

Очевидно, что семантика функции $\mathrm{F}_{\boldsymbol{u}}(\cdot)$ однозначно определяется алгоритмом 2. Функция $\sigma(j)$, используемая в формуле (27), преобразует натуральное число j в двоичное представление, состоящее из K разрядов. Каждому разряду, содержащему единицу, сопоставляется соответствующая гиперплоскость из списка $\mathcal{U}$. Таким образом получается подмножество гиперплоскостей. Например, пусть $\mathcal{U} = [H_2, H_4, H_7, H_9]$ и $j = 5$. В этом случае $K = 2^4 - 1$. Функция $\sigma(\cdot)$ преобразует число 5 в двоичную запись из 4-х разрядов 0101 и возвращает в качестве результата подмножество гиперплоскостей $J = \{H_2, H_7\}$. Таким образом, функция высшего порядка $Map\,(\mathrm{F}_{\boldsymbol{u}}, \mathcal{L}_{map})$ преобразует список номеров подмножеств $\mathcal{L}_{map}$ в список пар $(\boldsymbol{d}_j, f_j)$:

$$Map\,(\mathrm{F}_{\boldsymbol{u}}, \mathcal{L}_{map}) = [\mathrm{F}_{\boldsymbol{u}}(1), \ldots, \mathrm{F}_{\boldsymbol{u}}\,(K)] = [(\boldsymbol{d}_1, f_1), \ldots, (\boldsymbol{d}_K, f_K)]\,.$$

Здесь $\boldsymbol{d}_j$ является вектором движения по соответствующей грани допустимого многогранника M, а f_j — значение целевой функции, которое при этом достигается.

Обозначим $\mathcal{L}_{reduce} = [(\boldsymbol{d}_1, f_1), \ldots, (\boldsymbol{d}_K, f_K)]$. Определим бинарную ассоциативную операцию

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

являющуюся первым параметром функции высшего порядка *Reduce*:

$$\left(\boldsymbol{d}', f'\right) \oplus \left(\boldsymbol{d}'', f''\right) = \begin{cases} \left(\boldsymbol{d}', f'\right), & \text{если } f' \geqslant f''; \\ \left(\boldsymbol{d}'', f''\right), & \text{если } f' < f''. \end{cases} \tag{28}$$

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

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

где

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

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 вводит исходные данные задачи ЛП и начальную точку $\boldsymbol{u}_0$. 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 $\boldsymbol{u}_k$ всем рабочим. На шаге 16 он получает от рабочих частичные результаты, которые на шаге 17 редуцируются в пару $(\boldsymbol{d}_k, f_k)$. Если на шаге 18 выполняется условие $\boldsymbol{d}_k = \boldsymbol{0}$, то решение найдено (мы предполагаем $\boldsymbol{d}_0 \neq \boldsymbol{0}$). В этом случае на шаге 19 мастер присваивает логической переменной *exit* значение **true**. Если $\boldsymbol{d}_k \neq \boldsymbol{0}$, то мастер на шаге 21 вычисляет следующее приближение $\boldsymbol{u}_{k+1}$, 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 $\boldsymbol{u}_k$ 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 вводит исходные данные задачи ЛП. Цикл **repeat/until** рабочего (шаги 3-26) соответствует циклу **repeat/until** мастера. На шаге 4 рабочий получает от мастера текущее приближение $\boldsymbol{u}_k$. Затем он формирует подсписок $\mathcal{L}_{map(l)}$ своих порядковых номеров подмножеств для последующей обработки (шаги 5–13). Для удобства программирования нумерация подмножеств начинается с нуля. Подсписки различных рабочих не пересекаются, и их объединение дает полный список порядковых номеров всех рассматриваемых подмножеств:

$$\mathcal{L}_{map} = \mathcal{L}_{map(0)} +\!\!+ \ldots +\!\!+ \mathcal{L}_{map(L-1)}. \tag{29}$$

Символ $+\!\!+$ здесь обозначает операцию конкатенации списков. На шаге 14 рабочий вызывает функцию высшего порядка *Map*, которая, в свою очередь, применяет параметризованную функцию $\mathrm{F}_{\boldsymbol{u}_k}$, определенную формулами (26), 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 $(\boldsymbol{d}_l, f_l)$ с помощью функции высшего порядка *Reduce*, которая последовательно применяет бинарную операцию $\oplus$, определенную по формуле (28), 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 **Bcast**, **Gather**, **RecvFromMaster** и **SendToMaster** обеспечивают неявную синхронизацию работы процесса–мастера и процессов–рабочих.

6. 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/BSF-Surface-movement-method`. С помощью этой программы мы исследовали масштабируемость алгоритма AlFaMove. Масштабные вычислительные эксперименты проводились на вычислительном кластере «Торнадо ЮУрГУ» [16], характеристики которого представлены в табл. 1. В качестве тестов

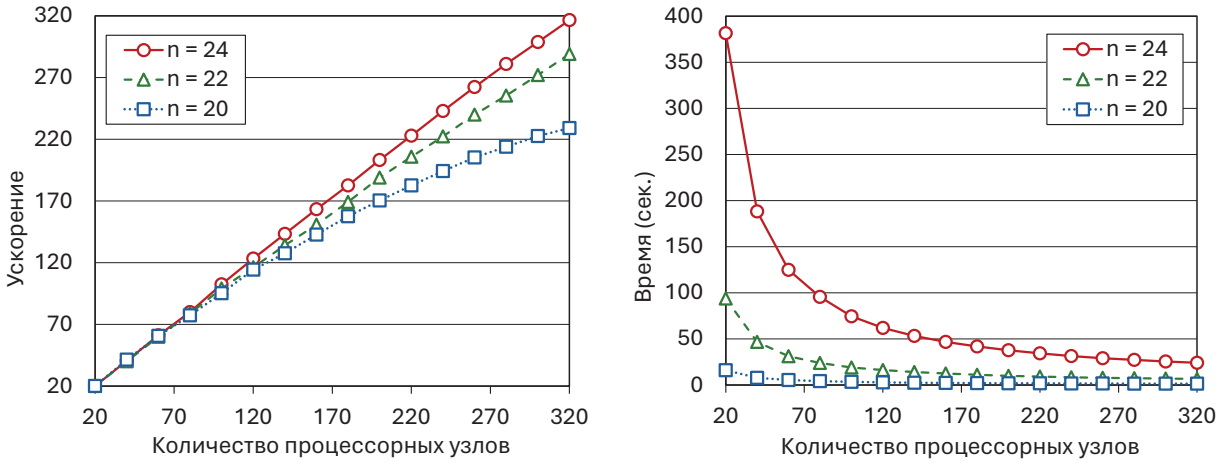

Рис. 3. Ускорение и временные затраты алгоритма AlFaMove.

мы использовали искусственные задачи, полученные с помощью генератора случайных задач ЛП FRaGenLP [17]. Все сгенерированные задачи доступны на GitHub по адресу `https://github.com/leonid-sokolinsky/Random-LP-Problems`. Верификация решений, выдаваемых алгоритмом AlFaMove, осуществлялась программой VaLiPro [18]. Была выполнена серия вычислительных экспериментов, в которой для задач ЛП различной размерности исследовалось ускорение и затраченное время в зависимости от количества используемых процессорных узлов кластера. Результаты этих экспериментов представлены на рис. 3. 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:

$$\alpha(L) = \frac{T(1)}{T(L)}. \tag{30}$$

Вычисления проводились для следующих размерностей: 20, 22 и 24. Число ограничений соответственно составило 41, 45 и 49. Списки $\mathcal{L}_{map}$ и $\mathcal{L}_{reduce}$. Для размерности $n = 24$ длина каждого из этих списков составила 16 777 215 элементов. Это — максимальный размер, допускаемый используемым компилятором. Следует отметить, что вычисления проводились с двойной точностью, при которой число с плавающей точкой занимает в оперативной памяти 64 бита.

Эксперименты показали, что алгоритм AlFaMove хорошо масштабируется даже на таких размерностях. Задачи всех размерностей показали на 80 процессорных узлах практически идеальное линейное ускорение (см. левый график). Для задачи размерности $n = 24$ эффективность распараллеливания составила цифру, близкую к 100%. В тоже

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

время правый график показывает, что с ростом размерности наблюдается экспоненциальный рост времени вычислений. Прогоны всех задач доступны на GitHub по адресу `https://github.com/leonid-sokolinsky/BSF-Surface-movement-method/tree/main/Runs`.

Мы не смогли протестировать алгоритм AlFaMove на задачах из репозитория Netlib-LP [19], так как во всех этих задачах количество гиперплоскостей, проходящих через начальную точку $\boldsymbol{u}_0$, превышало число 30, что соответствует количеству возможных комбинаций, равному 1 073 741 824. В целях преодоления «проклятия размерности» мы планируем в ближайшее время разработать и реализовать стохастический вариант алгоритма AlFaMove, который не будет требовать создания больших массивов.

Заключение

В статье предложен новый масштабируемый алгоритм линейного программирования, получивший название «AlFaMove». Алгоритм является численной реализацией метода поверхностного движения, ранее предложенного авторами. Ключевой особенностью этого метода является построение оптимального пути на поверхности допустимого многогранника от начальной точки к решению задачи линейного программирования. Под оптимальным путем понимается кратчайший 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, формирующие 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. Приведен алгоритм проекционного типа для построения псевдопроекции на линейные многообразия меньших размерностей, образуемые пересечением гиперплоскостей. 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 линейное ускорение близкое к оптимальному 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.

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. 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.

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. Соколинский Л.Б., Соколинская И.М. О валидации решений задач линейного программирования на кластерных вычислительных системах // Вычислительные методы и программирование. 2021. Т. 22, № 4. С. 252–261. DOI: 10.26089/NUMMET.V22R416.

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