Декомпозиционный алгоритм для задач линейного программирования, не имеющих блочной структуры, в среде Everest

В.В. Волошинов

***Аннотация* — В статье описывается декомпозиционный метод решения задач линейного программирования общего вида, когда в матрице ограничений нельзя или затруднительно выделить блочную структуру, используемую классическими декомпозиционными алгоритмами Данцига-Вульфа или Бендерса. Вместо этого применяется произвольное разделение матрицы ограничений на некоторое количество подматриц, соответствующих группам строк. Как и в алгоритме Данцига-Вульфа, вычислительный метод соответствует поиску максимума вогнутой функции, зависящей от двойственных переменных, субградиентным методом. Решение исходной задачи проводится итеративно, включая этапы решения наборов независимых подзадач меньшей размерности, чем у исходной. Выбор числа подматриц (и подзадач) зависит от размерности исходной задачи и вычислительной мощности распределенной среды, где при возможно параллельное решение указанных независимых подзадач. Конкретно, предлагается использовать сервисы решения задач оптимизации, развернутые на платформе Everest, everest.distcomp.org. Для программирования алгоритма расчетов и обмена данными с решателями, подключенными к Everest, можно использовать систему AMPLX, использующую алгебраический язык оптимизационного моделирования AMPL, ampl.com. Вместо коммерческого транслятора AMPL возможно применять интерпретатор языка Python и свободно доступный пакет Pyomo, pyomo.org, обеспечивающий взаимодействие с AMPL-совместимыми решателями. Программная реализация алгоритма в распределенной среде сервисов оптимизация Everest позволяет получить заметное ускорение при решении масштабных задач.**

***Ключевые слова* — линейное программирование, лагранжевая релаксация, ленточная декомпозиция, распределенные вычисления, платформа Everest.**

I. Введение

Решение задач оптимизации большой размерности является одной из важных областей развития декомпозиционных алгоритмов и технологий распределенных вычислений.

Здесь описывается декомпозиционная схема решения задач линейного программирования. Рассматривается общий случай, когда в матрице ограничений нельзя или затруднительно выделить блочную структуру, используемую классическими декомпозиционными алгоритмами Данцига-Вульфа или Бендерса [1], [2]. Вместо этого предлагается использовать, вообще говоря, произвольную ленточную строчную («горизонтальную») или столбцовую («вертикальную») декомпозицию матрицы ограничений на некоторое количество подматриц. Количество этих подматриц зависит от размерности исходной задачи и вычислительной мощности распределенной среды, где происходит выполнение алгоритма. Конкретно предлагается использовать сервисы решения задач оптимизации, развернутые на платформе Everest, everest.distcomp.org, [3]. Для программирования алгоритма расчетов и обмена данными с решателями, подключенными к Everest, можно использовать систему AMPLX [4], использующую один из самых популярных алгебраических языков оптимизационного моделирования AMPL, ampl.com.

Также, вместо лицензированного, коммерческого, транслятора AMPL возможно применять интерпретатор языка Python и свободно доступный в исходных кодах пакет Pyomo (PYthon Optimization Modeling Objects) [6], pyomo.org, обеспечивающий взаимодействие с AMPL-совместимыми решателями. Общая схема применения Pyomo для расчетов в среде сервисов оптимизации Everest приведена в статье [7].

Предлагаемый алгоритм был разработан автором самостоятельно и казался оригинальным. Однако, в книге [8], Гл. 2, в п. 5 «Метод расслоения ограничений» описывается похожая схема для случая строчной декомпозиции на две подматрицы. Там же указаны ссылки на более ранние работы ([9], [10]), где применялась такая декомпозиция.

II. Общая схема декомпозиции

Рассмотрим задачу линейного программирования достаточно общего вида:

$$\begin{gathered} c^{\mathsf{T}}x \to \min\limits_{x} \\ Ax = b,\ b \in \mathbb{R}^m \\ x \geqslant 0,\ x \in \mathbb{R}^n \end{gathered} \tag{1}$$

у которой, либо заведомо нет, либо неизвестно и трудно выявить характерные свойства «блочности»: т. е. наличие выделенных групп переменных и/или ограничений (см. Рис. 1. Схематичная структура блочных матриц.

1

Пусть матрица $\boldsymbol{A}$ ограничений задачи ( ) разрезана на горизонтальные «ленты» (см. Рис. **Ошибка! Неверная ссылка закладки.**). Здесь матрицы, имеют размерность и

Статья получена 20 декабря 2018. Работа выполнена за счет гранта Российского научного фонда (проект №16-11-10352).

В.В. Волошинов, Институт проблем передачи информации им. А.А. Харкевича РАН (e-mail: vv_voloshinov@iitp.ru).

, соответственно, где

.

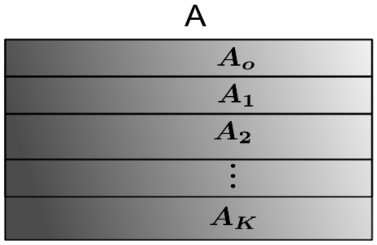

Рис. 2). Также возможно ситуация, когда блочная структура в задаче большой размерности наличествует, но она слишком «крупна», чтобы дать заметный прирост производительности. Во всех указанных случаях непосредственное применение известных декомпозиционных схем невозможно. Будем предполагать, что задача имеет решение, т.е. множество ее допустимых решений не пусто. Дополнительные предположения будут сделаны по ходу изложения.

Предлагается схема декомпозиции произвольной задачи, сводящая ее решение к итеративной процедуре, включающей этап решения набора независимых подзадач, размерность которых меньше, чем у исходной. Это обстоятельство позволяет надеяться на ускорение, если указанные подзадачи решать параллельно.

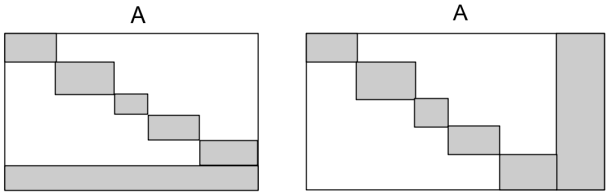

Рис. 1. Схематичная структура блочных матриц.

Пусть $(m\times n)$ матрица $\boldsymbol{A}$ 1

$A_0,$ и A_k $(k{=}1{:}K)$ ограничений задачи ( ) разрезана на горизонтальные «ленты» (см. Рис. **Ошибка! Неверная ссылка закладки.**). Здесь матрицы, имеют размерность $(m_o\times n)$ и $(m_k\times n)$ $(k{=}1{:}K)$, соответственно, где $m_o+\sum_{k=1:K} m_k = m$.

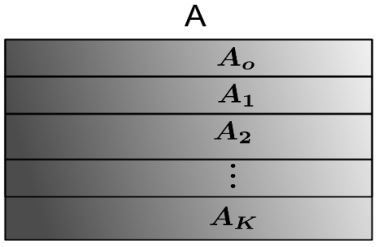

Рис. 2. «Строчная» декомпозиция

1

Тогда задача ( ) может быть записана в следующей эквивалентной форме:

$$
\begin{array}{ll}
 & c^{\mathsf{T}}x \to \min\limits_{x,\xi_k} \\
y_o & A_o x = b_o,\ b_o \in \mathbb{R}^{m_o} \\
y_k & A_k\xi_k = b_k,\ b_k \in \mathbb{R}^{m_k}\ \ (i{=}1{:}K)\,, \\
z_k & x - \xi_k = 0,\ \ (k{=}1{:}K)\,, \\
 & x \geqslant 0,\ \xi_k \geqslant 0,\ x{\in}\mathbb{R}^n,\ \xi_k{\in}\mathbb{R}^n.
\end{array}
\tag{2}
$$

со «вспомогательными» переменными ξ_k и соответствующими ограничениями (вектора b_o и $b_k(k{=}1{:}K)$ - результат разделения компонент вектора правой части ограничений b). Слева от вертикальных 2

черт в ( ) указаны обозначения для двойственных оценок групп равенств. Перед тем, как выписать 2

двойственную задачу для ( ), укажем вид двойственной 1

задачи для ( ):

$$
\begin{array}{l}
y^{\mathsf{T}}b \to \max\limits_{y} \\
y^{\mathsf{T}}A \leqq c^{\mathsf{T}},\ y{\in}\mathbb{R}^m.
\end{array}
\tag{3}
$$

Теперь, «по аналогии», запишем двойственную задачу 2

для ( )

$$
\begin{array}{rcl}
\sum\limits_{k-0:K} y_i^{\mathsf{T}}b_k = y_0^{\mathsf{T}}b_0 + \sum\limits_{k-1:K} y_k^{\mathsf{T}}b_k & \to & \max\limits_{y_k} \\
y_0^{\mathsf{T}}A_0 + \sum\limits_{k-1:K} z_k^{\mathsf{T}} & \leq & c^{\mathsf{T}}, \\
y_k^{\mathsf{T}}A_k - z_k^{\mathsf{T}} & \leq & 0\ (k{=}1{:}K).
\end{array}
\tag{4}
$$

Используя введенные обозначения, определим набор функций, зависящих от вектора двойственных оценок $z \doteq (z_1, z_2, \ldots, z_k, \ldots, z_K) \in \mathbb{R}^{K\cdot n}$:

$$
\begin{array}{l}
\phi_o(z) \doteq \max\limits_{y_0\in\Omega_0(z)} \left\{y_0^{\mathsf{T}}b_0\right\}, \\
\text{где}\ \Omega_0(z) \doteq \left\{y{\in}\mathbb{R}^{m_0},\ y_0^{\mathsf{T}}A_0 + \sum\limits_{k=1}^{K} z_k^{\mathsf{T}} \leqq c^{\mathsf{T}}\right\} \\
\phi_k(z) \doteq \max\limits_{y_k\in\Omega_k(z)} \left\{y_k^{\mathsf{T}}b_k\right\}, \\
\text{где}\ \Omega_k(z) \doteq \left\{y_k{\in}\mathbb{R}^{m_k},\ y_k^{\mathsf{T}}A_k - z_k^{\mathsf{T}} \leqq 0\right\}\ \ (k{=}0{:}K)\,.
\end{array}
\tag{5}
$$

Нетрудно заметить, что оптимальное значение 4

критерия в двойственной задаче ( ) совпадает со значением максимума суммы вышеуказанных функций:

$$
\Phi(z) \doteq \phi_0(z) + \sum_{k=1:K} \phi_k(z) \to \max_{z}
\tag{6}
$$

Прежде, чем перейти к описанию схемы алгоритма

5

заметим, что значения функций могут быть также определены в результате решения задач линейного

5

программирования (по аналогии с определениями ( ), здесь введены обозначения для множеств допустимых решений соответствующих задач):

$$\begin{array}{l}\phi_o(z) \doteq \min\limits_{x\in Q_0}\left\{\left(c-\sum\limits_{k=1:K} z_k\right)^{\mathsf{T}}\cdot x\right\},\\ \text{где } Q_0 \doteq \{x\in\mathsf{R}^n, A_0x=b_0, x\geqq 0\}\\ \phi_k(z) \doteq \min\limits_{x\in Q_k}\left\{z_k^{\mathsf{T}}x\right\},\\ \text{где } Q_k \doteq \{x\in\mathsf{R}^n, A_kx=b_k,\ x\geqq 0\}\ \ (k=1:K).\end{array} \qquad (7)$$

III. Описание алгоритма

Алгоритмически, предлагаемая вычислительная схема соответствует поиску максимума вогнутой функции Ф*(z)*

6

( ) субградиентным методом [2]. Вогнутость этой функции следует из вогнутости ее слагаемых

7

$\phi_0(z),\ \phi_k(z)\ (k=1:K)$, что видно из выражений ( ) (минимум аффинных по z функций по множествам, независящим от z).

Для запуска вычислительной схемы нужен вектор z^0, для которого значение функции $\Phi(z^0)$ конечно. Здесь будет полезно следующее утверждение.

Теорема III.1. Пусть множества допустимых решений $Q \doteq \{x\in\mathbb{R}^n, Ax=b, x\geq 0\}$ и $\Omega \doteq \{y\in\mathbb{R}^m,\ y^{\mathsf{T}}A\leqq c^{\mathsf{T}}\}$, соответственно, в исходной и двойственной к ней задачах не пусты. Тогда:

1) $\forall z \Rightarrow \max\{\phi_0(z), \phi_k(z)\ (k=1:K)\} < +\infty$;

2) $\exists z^o \Rightarrow \min\{\phi_0(z^o), \phi_k(z^o)\ (k=1:K)\} > -\infty$

Доказательство. Если $Q\neq\varnothing$, то и все множества Q_0,

7

$Q_k\ (k=1:K)$, ( ), также не пусты. Это следует из очевидных включений $Q\subset Q_0$ и $Q\subset Q_k\ (k=1:K)$. Таким образом можно считать, что п. 1) доказан. Другими словами, для произвольного вектора z, значения всех функций $\phi_0(z),\ \phi_k(z)\ (k=1:K)$ ограничены сверху.

Для доказательства п. 2) построим такой вектор z^o,

5

чтобы все множества $\Omega_0(z^o), \Omega_k(z^o)\ (k=1:K)$ из ( ) были бы непусты. Рассмотрим произвольный вектор y^0 из непустого множества Ω, т. е. $y^{o\mathsf{T}}A\leqq c^{\mathsf{T}}$.

По правилу строчной декомпозиции этот вектор можно записать в виде $y^o=(y_0^o, y_1^o, \ldots, y_k^o, \ldots, y_K^o)$, где подвектора y_0^o, $y_k^o\ (k=1:K)$ соответствуют подматрицам

2

A_o, A_k (см. ( )). По определению, для этих подвекторов выполнены неравенства:

$$y_0^{o\mathsf{T}}A_0 + \sum_{k=1:K} y_k^{o\mathsf{T}}A_k \leqq c^{\mathsf{T}} \qquad (8)$$

8

Положим $z_k^o \doteq y_k^{o\mathsf{T}}A_k$. Из ( ) и определений получаем:

$$y_0^{o\mathsf{T}}A_0 + \sum_{k=1}^{K} z_k^{o\mathsf{T}} \leqq c^{\mathsf{T}} \Rightarrow y_0^o \in \Omega_0(z^o)$$

$$y_k^{o\mathsf{T}}A_k - z_k^{o\mathsf{T}} = 0 \Rightarrow y_k^o \in \Omega_k(z^o) \text{ для всех } (k=0:K)$$

Теорема доказана.

Заметим, что если все коэффициенты целевой функции исходной задачи неотрицательны, $c\geqq 0$ (что часто имеет место на практике), то можно положить $z^o \equiv 0$.

Таким образом, из совместности прямой и двойственной задач следует, что все функции $\phi_0(z)$, $\phi_k(z)\ (k=1:K)$ являются *собственными* вогнутыми функциями, а их значения определяются в результате

7

решения задач линейного программирования ( ).

Пусть для некоторого z, после решения указанных подзадач найдены вектора $x_0^*[z], x_1^*[z] \ldots x_k^*[z] \ldots x_K^*[z]$:

$$x_0^*[z] = \operatorname*{Argmin}_x\left\{\left(c-\sum_{k=1:K} z_k\right)^{\mathsf{T}}\cdot x : x\in Q_0\right\},$$

$$x_k^*[z] = \operatorname*{Argmin}_x\left\{z_k^{\mathsf{T}}x : x\in Q_k\right\}\ \ (k=1:K).$$

Как известно из выпуклого анализа, эти соотношения позволяют определить вектора из субдифференциалов функций $\phi_0(z),\ \phi_k(z)\ (k=1:K)$ в точке z :

$$\begin{array}{l}(-x_0^*[z], \ldots, -x_0^*[z], \ldots, -x_0^*[z]) \in \partial\phi_0(z) \subset \mathbb{R}^{K\cdot n},\\ (0, \ldots, 0, x_k^*[z], 0, \ldots, 0) \in \partial\phi_k(z) \subset \mathbb{R}^{K\cdot n}\ \ (k=1:K).\end{array} \qquad (9)$$

В первой вектор-строке K раз повторяется n-мерный вектор $x_0^*[z]$, взятый с обратным знаком. Во второй

9

группе соотношений ( ), для k-й функции все компоненты субградиента равны нулю, кроме k-й

6

группы из n компонент вектора $x_k^*[z]$. Из ( ) следует, что все элементы субдифференциала $\partial\Phi(z)$ могут быть представлены в виде сумм указанных векторов:

$$\partial\Phi(z) = \partial\phi_0(z) \oplus \sum_{k-1:K} \partial\phi_k(z) \qquad (10)$$

Проверим, что принадлежность нуля $\partial\Phi(z)$ соответствует решению исходной задачи. Действительно, если для некоторого z^*, $0 \in \partial\Phi(z^*)$, то это означает, что нашелся вектор $x_0^*[z^*]$, для которого

$$x_0^*[z] \in \operatorname*{Argmin}_x \left\{ \left(c - \sum_{k-1:K} z_k^* \right)^\top x : x \in \mathbb{R}^n, A_0 x = b_0, x \geqq 0 \right\},$$

$$x_0^*[z] \in \operatorname*{Argmin}_x \left\{ z_k^{*\top} x : x \in \mathbb{R}^n, A_k x = b_k, x \geqq 0 \right\} \quad (k=1:K).$$

Это можно записать в эквивалентной форме.

$$\forall x \in \mathbb{R}^n, A_0 x = b_0, x \geqq 0 : \left(c - \sum_{k-1:K} z_k^* \right)^\top x_0^*[z] \leqslant \left(c - \sum_{k-1:K} z_k^* \right)^\top x,$$

$$\forall x \in \mathbb{R}^n, A_k x = b_k,\ x \geqq 0 :\ z_k^{*\top} x_0^*[z] \leqslant z_k^{*\top} x \quad (k=1:K).$$

Складывая теперь левые и правые части указанных неравенств для скалярных произведений получим

$$\forall x \in \mathbb{R}^n, A_0 x = b_0, A_k x = b_k\ (k=1:K), x \geqq 0 : c^\top x_0^*[z^*] \leqslant c^\top x.$$

Но последнее и означает, что $x_0^*[z^*]$ - решение исходной задачи. Теперь мы можем описать предлагаемый алгоритм следующим образом.

Шаг 0. *Выбор начального вектора z.*

На этом шаге нужно выбрать некоторый вектор z^o, для которого значение функции $\Phi(z^0)$ конечно. Здесь можно использовать Теорему III.1. Например, если вектор c неотрицателен, то можно положить z^0 равным нулю. Если удалось найти какое-нибудь допустимое решение двойственной задачи, то можно положить $z_k^o \doteq y_k^{o\top} A_k$, где $y^o = (y_0^o, y_1^o, \ldots, y_k^o, \ldots, y_K^o)$.

Шаг 2s+1 (s=0,1,...). *Решение подзадач.*

На этом, «нечетном», шаге решаются независимые подзадачи (всего — (K+1) подзадач)

$$x_0^s = \operatorname*{Argmin}_x \left\{ \left(c - \sum_{k-1:K} z_k^s \right)^\top x : x \in \mathbb{R}^n, A_0 x = b_0, x \geqq 0 \right\},$$
$$x_k^s = \operatorname*{Argmin}_x \left\{ z_k^{s\top} x : x \in \mathbb{R}^n, A_k x = b_k,\ x \geqq 0 \right\} (k=1:K). \qquad (11)$$

Шаг 2(s+1) (s=0,1,...). *Проверка критерия остановки и смена вектора z..*

На этом, «четном», шаге вначале проверяется выполнение с требуемой точностью равенств

$$x_0^s = x_k^s\ (k=1:K) \qquad (12)$$

Если равенства выполнены, то алгоритм останавливается, выдавая решение x_0^s. Если это не так, то формируется вектор

$$\bar{z}^s \doteq (x_1^s - x_0^s, x_2^s - x_0^s, \ldots, x_k^s - x_0^s, \ldots, x_K^s - x_0^s) \qquad (13)$$

9

который, как следует из ( ) принадлежит субдифференциалу $\partial\Phi(z^s)$. Естественно теперь сделать «шаг» вдоль субдифференциала в надежде получить большее значение функции $\Phi(z)$. Однако длину такого шага надо заранее ограничить, чтобы остаться в множестве конечных значений $\Phi(z)$. Используем для этого двойственное определение слагаемых этой

5

функции соотношениями ( ). Заметим, что найдя

6

решения задач ( ) любым AMPL-совместимым пакетом, помимо векторов «прямых» переменных, становятся известны соответствующие вектора двойственных переменных $y_0^s, y_1^s, y_2^s, \ldots, y_k^s, \ldots, y_K^s$, являющиеся

5

решениями двойственных задач ( ). Исходя из этого можно предложить выбирать следующее значение вектора $z^{(s+1)}$ по правилу:

$$z^{(s+1)} \doteq z^s + t^s \bar{z}^s$$

где число t^s определяется в результате решения следующей «координирующей»-задачи (мастер-задача в англоязычной терминологии)

$$\begin{aligned} t &\to \max_{t \geqslant 0} \\ A_0^\top y_0^s + \sum_{k=1:K} (z_k^s + t z_k^s) &\leqq c, \\ A_k^\top y_k^s - (z_k + t \bar{z}_k^s) &\leqq 0\ (k=1:K). \end{aligned} \qquad (14)$$

IV. О возможностях ускорения и применимости предложенного алгоритма

Перспектива получить ускорение решения задачи большой размерности, не имеющей явной блочной структуры, применяя предложенную вычислительную схему, основана на следующих соображениях.

Во-первых, «нечетный» шаг алгоритма ((2s+1)-й) заключается в поиске решения (K+1) задач линейного программирования с матрицами ограничений размера $(m_o \times n)$ и $(m_k \times n)$ $(k=1:K)$. Несмотря на то, что число переменных в этих задачах остается прежним (n), число строк может быть, вообще говоря, выбрано произвольным. Требуется лишь, чтобы выполнялось равенство $m_o + \sum_{k=1:K} m_k = m$.

Учтем теперь давно известное «эмпирическое» правило, согласно которому трудоемкость симплекс-метода пропорциональна $m^3 n$, где m - число

ограничений (без учета условий неотрицательности на переменные), а n - размерность вектора переменных (см. [11]). Это замечание позволяет надеяться, что даже если декомпозиционный алгоритм совершит большее число итераций (чередуя параллельное решение независимых подзадач с решение одной мастер-задачи), чем тот же симплекс, примененный к исходной задаче ( ), то общее время поиска решения сократится. Например, если матрицу исходной задачи «разрезать» на 100 «горизонтальных лент» (K=99), то нечетный шаг алгоритма, при наличии сотни доступных пакетов решения ЛП-задач, подключенных к системе AMPLX, может выполняться, примерно в 10^6 (миллион!) раз быстрее, чем решение исходной, «полноразмерной», задачи. Даже если общее число итераций будет больше, чем потребовалось бы для исходной задачи, эта «фора» параллельной фазы декомпозиционного алгоритма позволяет надеяться на ускорение.

1

Условием практической применимости этой схемы является наличие удобных программных средств: декомпозиции исходной задачи на указанные подзадачи; формирования «промежуточной» задачи четного шага (пересчет вектора z); компоновки исходных данных новых подзадач по новому вектору z. Все эти действия удобно совершать именно в системе AMPLX, поскольку она основана на языке высокоуровневом оптимизационного моделирования AMPL. Кроме возможностей автоматического создания исходных данных всех задач в виде т. н. стаб-файлов (их часто еще называют NL-файлами), которые можно непосредственно отправлять AMPL-совместимым пакетам, результаты решения, SOL-файлы импортируются в AMPL-программу с решениями, полученными от пакетов (вместе с двойственными переменными).

Распараллеливание процессов решения независимых подзадач будет обеспечиваться диспетчером сервера Everest, к которому подключена система AMPLX.

Возможности AMPLX по реализации алгоритмов Данцига-Вулфа и Бендерса были успешно проверены на демонстрационных примерах их реализации на языке AMPL. Читатель может сравнить примеры по именам *multi.mod (*.dat)* и *benders.mod (*.dat)* на сайте ampl.com, страница https://ampl.com/resources/the-ampl-book/example-files, и соответствующую модификацию для системы AMPLX по ссылкам http://distcomp.ru/~vladimirv/restopt/amplx/dw/ и http://distcomp.ru/~vladimirv/restopt/amplx/benders/.

На тех же принципах возможна реализация вычислений на языке Python средствами пакета Pyomo (см. [7]).

V. Заключение

Разработан декомпозиционный алгоритм решения задачи линейного программирования общего вида, у которой, либо заведомо нет, либо трудно выделить «блочные» группы переменных и/или ограничений, позволяющих применить известные алгоритмы типа Данцига-Вульфа. Также возможно ситуация, когда блочная структура в задаче большой размерности наличествует, но она слишком «крупна», чтобы дать заметный прирост производительности. Предложенный алгоритм основан на произвольной «ленточной» декомпозиции ограничений исходной задачи и сводит решение к итеративной схеме, включающей этап решения набора независимых подзадач, размерность которых заметно меньше, чем у исходной. Это может ускорить расчет, если указанные подзадачи решать параллельно в распределенной среде сервисов оптимизации на платформе Everest.

Благодарности

Исследование выполнено за счет гранта Российского научного фонда (проект №16-11-10352).

Библиография

[1] Лэсдон Л.С, *Оптимизация больших систем*. М.: Наука, 1975.

[2] Мину М. *Математическое программирование. Теория и алгоритмы*: Пер. с фр. А. И. Штерна. М.: Наука, 1990.

[3] Sukhoroslov O., Volkov S., Afanasiev A. “A Web-Based Platform for Publication and Distributed Execution of Computing Applications”, in *Parallel and Distributed Computing (ISPDC), 14th International Symposium*, IEEE, 2015, pp. 175-184.

[4] Smirnov S., Voloshinov V., Sukhosroslov O. “Distributed Optimization on the Base of AMPL Modeling Language and Everest Platform” in *Procedia Computer Science*, vol. 101, 2016, pp. 313–322.

[5] Robert Fourer, David M. Gay, Brian W. Kernighan. *AMPL: A mathematical programming language*. Thomson/Brooks/Cole, 2003

[6] Hart, William E., Carl D. Laird, Jean-Paul Watson, David L. Woodruff, Gabriel A. Hackebeil, Bethany L. Nicholson, John D. Siirola. *Pyomo – Optimization Modeling in Python. 2nd Edition.* Vol. 67. Springer, 2017

[7] Афанасьев А.П., Волошинов В.В., Соколов А.В. «Обратные задачи моделирования на основе регуляризации и распределенных вычислений в среде Everest». В сборнике: *Аналитика и управление данными в областях с интенсивным использованием данных Сборник научных трудов XIX Международной конференции DAMDID / RCDL'2017.* Под ред. Л.А. Калиниченко, Я. Манолопулос, Н.А. Скворцова, В.А. Сухомлина. 2017. С. 132-140.

[8] Авербах И.Л., Цурков В.И. *Оптимизация в блочных задачах с целочисленными ограничениями.* М.: Наука, 1995

[9] Guignard M., Kim S. Lagrangean decomposition: A model yielding stronger lagrangean bounds, in *Mathematical Programming,* vol. 39, iss. 2, 1987, pp. 215-228

[10] Glover F., Klingman D. Layering strategies for creating exploitable structure in linear and integer programs, in *Mathematical Programming,* vol. 40, iss. 1-3, 1988, pp. 165-181

[11] Муртаф Б. *Современное линейное программирование*. М.: Мир, 1984.

Decomposition algorithm for linear programming problems without a block structure in the Everest computing environment

V.V. Voloshinov

Abstract — **The article describes decomposition method for solving linear programming problems in a general case when it is impossible or difficult to reveal the block structure (of the constraint matrix) used by the classic Danzig-Wulf or Benders decomposition algorithms. Instead, an arbitrary partition of the constraint matrix into a number of submatrices corresponding to groups of rows is used. As in the Danzig-Wolfe algorithm, one searches a maximum of a concave function (on dual variables) by a subgradient method. Iterative routine of solving the original problem includes the phases of solving sets of independent subproblems of a smaller dimension than the original one. Number of submatrices (and subproblems) depends on the dimension of the original problem and the computational power of the distributed environment, where parallel solving of above independent subproblems is done. In concrete term, it is proposed to use the optimization services deployed on the Everest platform, everest.distcomp.org. Programming of the algorithm and data exchange with solvers connected to Everest, may be done by AMPLX system based on algebraic optimization modeling language AMPL, ampl.com. Also, instead of the commercial translator AMPL, it is possible to use the Python interpreter and the freely available Pyomo package, pyomo.org, which provides interaction with AMPL-compatible solvers. The software implementation of the algorithm by Everest platform, allows to hope to get a noticeable acceleration when solving large-scale problems.**

Keywords — **linear programming, Lagrangian relaxation, tape decomposition, distributed computing, Everest platform.**

REFERENCES

[1] Ljesdon L.S, Optimizacija bol'shih sistem. M.: Nauka, 1975.

[2] Minu M. Matematicheskoe programmirovanie. Teorija i algoritmy: Per. s fr. A. I. Shterna. M.: Nauka, 1990.

[3] Sukhoroslov O., Volkov S., Afanasiev A. “A Web-Based Platform for Publication and Distributed Execution of Computing Applications”, in Parallel and Distributed Computing (ISPDC), 14th International Symposium, IEEE, 2015, pp. 175-184.

[4] Smirnov S., Voloshinov V., Sukhosroslov O. “Distributed Optimization on the Base of AMPL Modeling Language and Everest Platform” in Procedia Computer Science, vol. 101, 2016, pp. 313–322.

[5] Robert Fourer, David M. Gay, Brian W. Kernighan. AMPL: A mathematical programming language. Thomson/Brooks/Cole, 2003

[6] Hart, William E., Carl D. Laird, Jean-Paul Watson, David L. Woodruff, Gabriel A. Hackebeil, Bethany L. Nicholson, John D. Siirola. Pyomo – Optimization Modeling in Python. 2nd Edition. Vol. 67. Springer, 2017

[7] Afanas'ev A.P., Voloshinov V.V., Sokolov A.V. «Obratnye zadachi modelirovanija na osnove reguljarizacii i raspredelennyh vychislenij v srede Everest». V sbornike: Analitika i upravlenie dannymi v oblastjah s intensivnym ispol'zovaniem dannyh Sbornik nauchnyh trudov XIX Mezhdunarodnoj konferencii DAMDID / RCDL'2017. Pod red. L.A. Kalinichenko, Ja. Manolopulos, N.A. Skvorcova, V.A. Suhomlina. 2017. S. 132-140.

[8] Averbah I.L., Curkov V.I. Optimizacija v blochnyh zadachah s celochislennymi ogranichenijami. M.: Nauka, 1995

[9] Guignard M., Kim S. Lagrangean decomposition: A model yielding stronger lagrangean bounds, in Mathematical Programming, vol. 39, iss. 2, 1987, pp. 215-228

[10] Glover F., Klingman D. Layering strategies for creating exploitable structure in linear and integer programs, in Mathematical Programming, vol. 40, iss. 1-3, 1988, pp. 165-181

[11] Murtaf B. Sovremennoe linejnoe programmirovanie. M.: Mir, 1984.