УДК 519.688, 004.272.2 doi 10.26089/NumMet.v21r328

ОБ ОДНОМ ИТЕРАЦИОННОМ МЕТОДЕ РЕШЕНИЯ ЗАДАЧ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ НА КЛАСТЕРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

И. М. Соколинская[1], **Л. Б. Соколинский**[2]

Статья посвящена исследованию нового метода решения сверхбольших задач линейного программирования. Указанный метод получил название “апекс-метод”. Апекс-метод работает по схеме предиктор-корректор. На фазе предиктор находится точка, лежащая на границе n-мерного многогранника, задающего допустимую область задачи линейного программирования. На фазе корректор организуется итерационный процесс, в результате которого строится последовательность точек, сходящаяся к точному решению задачи линейного программирования. В статье дается формальное описание апекс-метода и приводятся сведения о его параллельной реализации на языке C++ с использованием библиотеки MPI. Приводятся результаты масштабных вычислительных экспериментов на кластерной вычислительной системе по исследованию масштабируемости апекс-метода.

Ключевые слова: линейное программирование, задача линейного программирования большой размерности, апекс-метод, схема предиктор-корректор, итерационный метод, параллельный алгоритм, кластерная вычислительная система.

1. Введение. Быстрое развитие технологий накопления и обработки больших данных [1, 2] привело к появлению оптимизационных математических моделей в виде сверхбольших задач линейного программирования (ЛП) [3]. Такие задачи возникают в индустрии, экономике, логистике, статистике, квантовой физике и других областях [4–7]. Классическое программное обеспечение во многих случаях не позволяет решить подобные масштабные задачи линейного программирования за приемлемое время [8]. Вместе с тем в ближайшие 2–3 года появятся вычислительные системы экзафлопсного уровня производительности [9], потенциально способные решать подобные задачи. В соответствии с этим актуальной является задача разработки новых эффективных методов для решения сверхбольших задач линейного программирования с помощью экзамасштабных вычислительных систем.

До настоящего времени одним из самых распространенных способов решения задачи ЛП являлся класс алгоритмов, предложенных и разработанных Данцигом на основе симплекс-метода [10]. Симплекс-метод оказался эффективным для решения большого класса задач ЛП. Однако симплекс-метод имеет некоторые фундаментальные особенности, ограничивающие его применение для больших задач линейного программирования. Во-первых, в определенных случаях симплекс-методу приходится перебирать все вершины симплекса, что соответствует экспоненциальной временной сложности [11–13]. Во-вторых, симплекс-метод в большинстве случаев удовлетворительно решает задачи ЛП, содержащие до 50000 переменных, однако на больших задачах часто наблюдается потеря точности [14], которая не может быть компенсирована даже путем применения таких ресурсоемких процедур, как “аффинное масштабирование” или “итерационное уточнение” [15]. В-третьих, в общем случае симплекс-метод плохо масштабируется на многопроцессорных системах с распределенной памятью. Были предприняты многочисленные попытки построить масштабируемую реализацию симплекс-метода, однако они не увенчались успехом [16]. Во всех случаях граница масштабируемости составляла от 16 до 32 процессорных узлов (например [17]). Хачиян в [18] предложил вариацию метода эллипсоидов Шора–Юдина–Немировского [19, 20], решающую любую

[1] Южно-Уральский государственный университет (национальный исследовательский университет), просп. Ленина, 76, 454080, Челябинск; к.ф.-м.н., доцент, e-mail: irina.sokolinskaya@susu.ru

[2] Южно-Уральский государственный университет (национальный исследовательский университет), просп. Ленина, 76, 454080, Челябинск; д.ф.-м.н., профессор, проректор по информатизации, e-mail: leonid.sokolinsky@susu.ru, ORCID 0000–0001–9997–3918

задачу ЛП за полиномиальное время, однако попытки применить этот подход на практике оказались безуспешными, так как в подавляющем большинстве случаев алгоритм Хачияна демонстрировал намного худшую эффективность по сравнению с симплекс-методом. Основываясь на работе Хачияна, Кармаркар в работе [21] предложил метод внутренних точек, который демонстрирует полиномиальное время решения задачи ЛП и применим на практике. Итерационные алгоритмы, основанные на методе Кармаркара, способны решать сверхбольшие задачи ЛП с миллионами переменных и миллионами уравнений [22–26]. Более того, эти алгоритмы являются самокорректирующимися, и поэтому обеспечивают высокую точность вычислений. В качестве недостатка метода внутренних точек следует отметить тот факт, что для начала работы алгоритма необходимо иметь точку, удовлетворяющую всем ограничениям задачи линейного программирования. Нахождение такой внутренней точки может сводиться к решению дополнительной задачи линейного программирования [27]. В качестве альтернативы можно указать метод псевдопроекций, основанный на использовании фейеровских отображений [28]. Еще одним существенным недостатком метода внутренних точек является его плохая масштабируемость применительно к многопроцессорным системам с распределенной памятью. Было сделано несколько попыток построить параллельную реализацию для частных случаев (например [29, 30]), но эффективную параллельную реализацию для экзамасштабных многопроцессорных систем в общем случае построить не удалось. В соответствии с этим является актуальным направление исследований, связанное с поиском новых масштабируемых методов решения задач линейного программирования.

Авторами в работе [3] был предложен масштабируемый итерационный метод по схеме предиктор-корректор для решения больших задач линейного программирования, ориентированный на кластерные вычислительные системы. Метод состоит из двух фаз: *Quest* (поиск) и *Target* (позиционирование). На фазе *Quest* происходит поиск допустимой точки, удовлетворяющей системе неравенств, задающих ограничения задачи линейного программирования. На фазе *Target* строится последовательность допустимых точек, сходящаяся к точному решению задачи линейного программирования. Исследованию фазы *Quest* были посвящены работы [3, 31–33]. В работе [31] было введено понятие псевдопроекции на выпуклое замкнутое множество, обобщающее понятие проекции. Метод псевдопроекции применяется на фазе *Quest* для нахождения начальной допустимой точки. Для вычисления псевдопроекции используются фейеровские приближения [34], способные "самоисправляться" при накоплении погрешности вычислений. В статье [32] было показано, что при вычислении псевдопроекции на многогранники большой размерности могут эффективно использоваться многоядерные ускорители. В [3] была доказана теорема о необходимом условии сходимости итерационного процесса вычисления псевдопроекции в случае, когда исходные данные изменяются в результате параллельного переноса многогранника. В работе [35] был предложен алгоритм для фазы *Target*, в соответствии с которым формируется специальная система точек, имеющая форму n-мерного осесимметричного креста, которая передвигается в n-мерном пространстве таким образом, чтобы решение задачи линейного программирования постоянно находилось в ε-окрестности центральной точки креста. Однако, вычислительная сложность этого алгоритма характеризуется экспоненциальной зависимостью от размерности задачи.

В данной статье предлагается и исследуется новый масштабируемый итерационный метод решения больших задач линейного программирования, ориентированный на кластерные вычислительные системы. Указанный метод получил название "апекс-метод". Апекс-метод также состоит из двух фаз — *Quest* и *Target*, однако, в отличие от предыдущего подхода, имеет полиномиальную сложность. Статья организована следующим образом. В разделе 2 дается математическая постановка задачи и формальное описание апекс-метода. В разделе 3 приводятся сведения о параллельной программной реализации фазы *Target* и описываются результаты масштабных вычислительных экспериментов по исследованию масштабируемости апекс-метода на большой кластерной вычислительной системе. В разделе 4 суммируются полученные результаты и намечаются направления дальнейших исследований.

2. Апекс-метод. Пусть в пространстве $\mathbb{R}^n$ задана задача ЛП

$$\bar{x} = \arg\max \{\langle c, x\rangle \,|Ax \leqslant b\}\,^1, \tag{1}$$

[1] $\langle c, x\rangle$ обозначает скалярное произведение двух векторов.

где матрица A имеет m строк. Мы здесь предполагаем, что ограничение $x \geqslant 0$ также включено в систему $Ax \leqslant b$ в виде неравенств

$$\begin{array}{ccccccccccc} -x_1 & + & 0 & + & \dots & \dots & \dots & + & 0 & \leqslant & 0; \\ 0 & - & x_2 & + & 0 & + & \dots & + & 0 & \leqslant & 0; \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & + & \dots & \dots & \dots & + & 0 & - & x_n & \leqslant & 0. \end{array}$$

Обозначим через a_i i-тую строку матрицы A. Везде далее мы предполагаем, что a_i не равна нулевому вектору для всех $i = 1, \dots, m$. Обозначим через M n-мерный многогранник, ограничивающий множество допустимых точек задачи (1). Такой многогранник всегда является выпуклым замкнутым множеством. Мы также будем предполагать, что многогранник M является ограниченным. По определению, точка x является *граничной* по отношению к многограннику M, если любая ее окрестность имеет непустое пересечение как с M, так и с его дополнением. Определим *границу* Γ_M многогранника M как множество всех его граничных точек.

Апекс-метод строится по схеме предиктор-корректор [36] и состоит из двух фаз: *Quest* (предиктор) и *Target* (корректор). Фаза *Quest* находит некоторую точку $\tilde{x} \in M$. Фаза *Target*, используя $\tilde{x}$, вычисляет последовательность точек $\{u_0, u_1, \dots, u_k, \dots\}$, обладающую следующими свойствами:

$$u_k \in \Gamma_M; \tag{2}$$

$$\langle c, u_k \rangle < \langle c, u_{k+1} \rangle; \tag{3}$$

$$\lim_{k \to \infty} \langle c, u_k \rangle = \bar{x}. \tag{4}$$

Условие (2) означает, что все точки последовательности "лежат" на границе многогранника. Условие (3) означает, что значение целевой функции в каждой следующей точке должно быть больше, чем в предыдущей. Условие (4) означает, что последовательность сходится к точному решению задачи (1).

Фаза *Quest* находит точку $\tilde{x} \in M$, используя операцию *псевдопроектирования* [28], являющуюся обобщением операции ортогонального проектирования. Дадим ее формальное определение. Пусть задано отображение $\varphi_M : \mathbb{R}^n \to \mathbb{R}^n$:

$$\varphi_M(x) = \frac{1}{h} \sum_{i=1}^{m} \rho_i^+(x), \tag{5}$$

где

$$\rho_i^+(x) = \frac{\max\{\langle a_i, x \rangle - b_i, 0\}}{\|a_i\|^2} a_i, \tag{6}$$

h — количество ненулевых слагаемых в сумме $\sum_{i=1}^{m} \rho_i^+(x)$. Тогда *псевдопроекция* $\pi_M(x)$ точки x на многогранник M определяется следующей формулой:

$$\pi_M(x) = \lim_{k \to \infty} \varphi_M^{(k)}(x), \tag{7}$$

где

$$\varphi_M^{(k)}(x) = \underbrace{\varphi_M(\varphi_M(\dots \varphi_M(x) \dots))}_{k}.$$

Алгоритм 1. Вычисление псевдопроекции

1: **input** x_0
2: $k := 0$
3: $x_{k+1} := x_{k+1} - \varphi_M(x_k)$
4: **if** $\|x_{k+1} - x_k\| < \varepsilon$ **goto** 7
5: $k := k + 1$
6: **goto** 3
7: **output** $\tilde{x} = x_{k+1}$
8: **stop**

Рассмотрим итерационный алгоритм 1, реализующий операцию псевдопроектирования. Алгоритм завершает свою работу, когда расстояние между соседними приближениями становится меньше малой величины $\varepsilon > 0$, являющейся параметром алгоритма. Доказательство сходимости алгоритма 1 можно найти в статье [37]. В работе [38] авторами была предложена и исследована масштабируемая параллельная реализация алгоритма 1 в виде операций над списками. С использованием стоимостной метрики модели параллельных вычислений BSF [39] было показано, что граница масштабируемости[2] указанного параллельного алгоритма может быть оценена, как $O(\sqrt{n})$.

[2]Под границей масштабируемости параллельного алгоритма для кластерной вычислительной системы понимается максимальное количество вычислительных узлов, вплоть до которого наблюдается рост ускорения, то есть распараллеливание является эффективным.

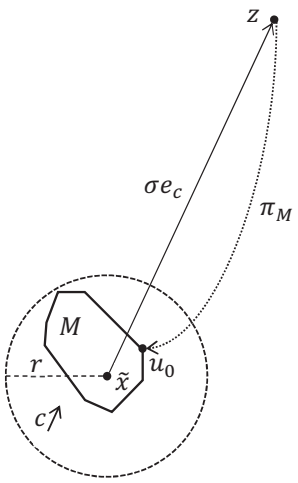

Рис. 1. Построение начального приближения u_0

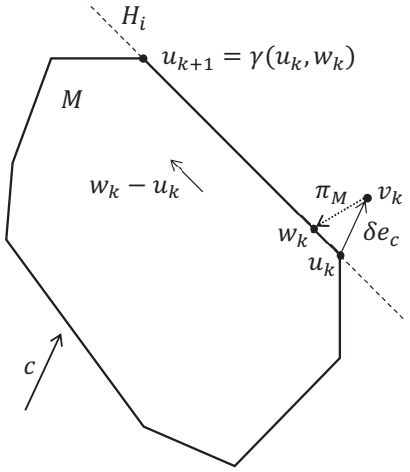

Рис. 2. Построение следующего приближения $u_{k+1} = \gamma(u_k, w_k)$

Фаза *Target* вычисляет последовательность точек $\{u_0, u_1, \ldots, u_k, \ldots\}$, удовлетворяющую свойствам (2)–(4), сходящуюся к точному решению задачи (1). Для вычисления начального приближения u_0 используется *точка апекса* z, которая определяется следующим образом. Пусть $\tilde{x} \in M$ — точка, полученная на фазе Quest. Обозначим через S n-мерный шар с радиусом r и центром в точке $\tilde{x}$, обладающий свойством: $M \subset S$ (рис. 1). Зафиксируем некоторое положительное число $\sigma \in \mathbb{R}_{>0}$, такое, что $\sigma \gg r$. Определим единичный вектор

$$e_c = \frac{c}{\|c\|},$$

где c — вектор коэффициентов целевой функции задачи (1). Тогда *точка апекса* вычисляется по формуле

$$z = \tilde{x} + \sigma e_c. \tag{8}$$

Зададим начальное приближение u_0 следующим образом:

$$u_0 = \pi_M(z),$$

то есть u_0 получается в результате псевдопроектирования точки апекса z на многогранник M. Используя схему доказательства сходимости алгоритма из [37], несложно показать, что в этом случае точка u_0 будет лежать на границе Γ_M многогранника M. Это означает, что найдется гиперплоскость H_i ($i \in \{1, \ldots, m\}$), определяемая уравнением $< a_i, x >= b_i$, такая, что

$$u_0 \in H_i \cap \Gamma_M. \tag{9}$$

Таким образом для точки u_0 выполняется условие (2).

Теперь предположим, что уже найдено приближение u_k ($k \in \mathbb{Z}_{\geqslant 0}$), удовлетворяющее условиям (2) и (3). Для построения следующего приближения u_{k+1} вычислим промежуточную точку

$$v_k = u_k + \delta e_c, \tag{10}$$

которая получается путем прибавления к точке u_k единичного вектора e_c, сонаправленного с вектором коэффициентов целевой функции c, умноженным на малую величину $\delta \in \mathbb{R}_{>0}$ (рис. 2). Применив операцию псевдопроектирования π_M к точке v_k, получаем следующую промежуточную точку

$$w_k = \pi_M(v_k). \tag{11}$$

Поскольку отображение φ_M, используемое для вычисления псевдопроекции в формуле (7), является однозначным M-фейеровским[3] отображением [34], то при достаточно малом δ из (9) и (10) следует, что

[3]Однозначное отображение $\varphi_M : \mathbb{R}^n \to \mathbb{R}^n$ называется фейеровским относительно множества M или кратко M-фейеровским, если $\forall y \in M\,(\varphi(y) = y) \vee \forall x \notin M\,(\forall y \in M(\|\varphi(x) - y\| < \|x - y\|))$.

$w_k \in H_i \cap \Gamma_M$, то есть w_k будет лежать на той же грани, что и u_k. Если значение целевой функции в точке u_k будет больше либо равно значения целевой функции в точке w_k, то точка u_k принимается за приближенное решение задачи (1). Предположим, что справедливо обратное, то есть $\langle c, w_k\rangle > \langle c, u_k\rangle$. Определим луч $L_{u_k w_k}$, исходящий из точки u_k в направлении точки w_k:

$$L_{u_k w_k} = \{x \in \mathbb{R}^n \,|x = u_k + \eta(w_k - u_k), \eta \in \mathbb{R}_{\geqslant 0}\}.$$

Определим отображение $\gamma : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ следующим образом:

$$\gamma(u_k, w_k) = \arg\max\{\|x - u_k\| \mid x \in L_{u_k w_k} \cap M\}. \tag{12}$$

Другими словами, отображение γ вычисляет точку, лежащую на луче $L_{u_k w_k}$, принадлежащую многограннику M, и максимально удаленную от точки u_k. Указанную точку возьмем в качестве следующего приближения $u_{k+1} = \gamma(u_k, w_k)$. Очевидно, что для точки u_{k+1} также будут выполняться условия (2) и (3).

Фаза *Target* состоит из головной процедуры (алгоритм 2) и процедуры вычисления отображения γ (алгоритм 3). Сначала рассмотрим алгоритм 2. На шаге 1 вводится точка $\tilde{x} \in M$, полученная на фазе *Quest* с помощью алгоритма 1. На шаге 2 вычисляется единичный вектор e_c, сонаправленный с вектором c, координаты которого совпадают с коэффициентами целевой функции задачи (1). На шаге 3 вычисляется точка апекса z в соответствии с формулой (8) (рис. 1). На шаге 4 с помощью алгоритма 1 вычисляется начальное приближение u_0, представляющее собой псевдопроекцию точки апекса z на многогранник M (формула (7) и рис. 1). На шаге 5 счетчик итераций k устанавливается в начальное значение 0. Итерационный цикл начинается на шаге 6, который вычисляет промежуточную точку v_k по формуле (10). На шаге 7 по формуле (11) вычисляется промежуточная точка w_k, лежащая на той же грани многогранника M, что и текущее приближение u_k (рис. 2). На шаге 8 проверяется условие завершения: если значение целевой функции в точке w_k не превышает значения целевой функции в точке u_k, то итерационный процесс заканчивается, и u_k выводится в качестве приближенного решения задачи (1). В противном случае точка w_k задает направление $(w_k - u_k)$, в котором значение целевой функции будет увеличиваться. На шаге 9 вычисляется допустимая точка u_{k+1}, в которой достигается максимум целевой функции по направлению $(w_k - u_k)$. Шаг 10 увеличивает на единицу счетчик итераций k. На шаге 11 осуществляется переход на шаг 6, начинающий следующую итерацию. Сходимость алгоритма 2 непосредственно следует из замкнутости и ограниченности многогранника M. Таким образом, условие (4) выполняется.

Алгоритм 2. Фаза *Target*: головная процедура

1: **input** $\tilde{x}$
2: $e_c := c/\|c\|$
3: $z := \tilde{x} + \sigma e_c$
4: $u_0 := \pi_M(z)$
5: $k := 0$
6: $v_k := u_k + \delta e_c$
7: $w_k := \pi_M(v_k)$
8: **if** $\langle c, w_k\rangle \leqslant \langle c, u_k\rangle$ **goto** 12
9: $u_{k+1} := \gamma(u_k, w_k)$
10: $k := k + 1$
11: **goto** 6
12: **output** $\bar{x} = u_k$
13: **stop**

Алгоритм 3. Фаза *Target*: процедура вычисления $\gamma(u_k, w_k)$

1: **input** u_k, w_k
2: $e_{u_k w_k} := (w_k - u_k)/\|w_k - u_k\|$
3: $\tau_0 := \mu; \quad t_0 := u_k$
4: $j := 0; \quad i := 0$
5: **if** $\tau_i < \varepsilon$ **goto** 13
6: $t_{j+1} := t_j + \tau_i e_{u_k w_k}$
7: **if** $t_{j+1} \notin M$ **goto** 10
8: $j := j + 1$
9: **goto** 6
10: $\tau_{i+1} := \tau_i/2$
11: $i := i + 1$
12: **goto** 5
13: **output** $u_{k+1} = t_j$; **stop**

Вычисления приближенного значения отображения γ, определяемого формулой (12) и используемого на шаге 9 алгоритма 2, выполняются по алгоритму 3. Указанный алгоритм позволяет вычислить последовательность точек $\{t_0, t_1, \ldots, t_j, \ldots\}$, такую, что

$$t_0 = u_k, \tag{13}$$

$$\lim_{j\to\infty} t_j = \gamma(u_k, w_k). \tag{14}$$

На шаге 1 вводятся исходные значения u_k и w_k. На шаге 2 вычисляется единичный вектор $e_{u_k w_k}$, сонаправленный с вектором $(w_k - u_k)$. На шаге 3 в качестве начального значения приращения τ_0 устанавливается константа $\mu \in \mathbb{R}_{>0}$, являющаяся параметром алгоритма, а в качестве начального приближения t_0 устанавливается точка u_k. На шаге 4 счетчики итераций j, i устанавливаются в значение 0. На шаге 5 проверяется условие завершения: если приращение τ_i меньше малой величины $\varepsilon \in \mathbb{R}_{>0}$, являющейся параметром алгоритма, то происходит переход на шаг 13, где в качестве результата выводится $u_{k+1} = t_j$, после чего алгоритм завершает свою работу. В противном случае выполняется шаг 6, вычисляющий следующее приближение t_{j+1} путем смещения точки t_j в направлении $(w_k - u_k)$ на расстояние τ_i. На шаге 7 проверяется, не вышла ли точка t_{j+1} за пределы M. Если это имеет место, то происходит переход на шаг 10, где в качестве новой величины смещения τ_{i+1} устанавливается $\frac{1}{2}\tau_i$. На шаге 11 счетчик i увеличивается на единицу и происходит переход на шаг 5 для повторного вычисления точки t_{j+1} с использованием меньшего смещения τ_i. Если при проверке условия на шаге 7 оказывается, что новое приближение t_{j+1} принадлежит многограннику M, то выполняется шаг 8, увеличивающий на единицу счетчик j, и происходит переход на шаг 6 (рис. 3). Очевидно, что последовательность точек $\{t_0, t_1, \ldots, t_j, \ldots\}$, генерируемая алгоритмом 3, удовлетворяет условиям (13), (14).

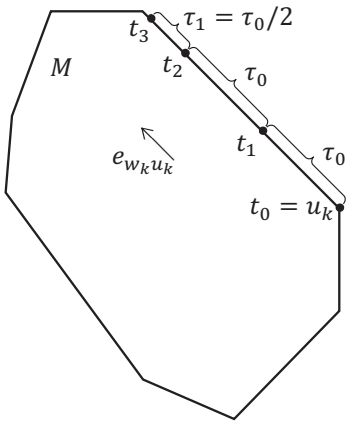

Рис. 3. Иллюстрация работы алгоритма 3

3. Программная реализация и вычислительные эксперименты. Нами была выполнена параллельная реализация алгоритма 2 на языке C++ с использованием программного каркаса BSF [40, 41] и библиотеки параллельного программирования MPI. Схема параллельной реализации идентична схеме, описанной в статье [42]. Исходные коды свободно доступны в сети Интернет по адресу https://github.com/leonid-sokolinsky/Apex-method. В качестве задачи была использована масштабируемая система неравенств размерности n из статьи [32]

$$\begin{cases} x_0 & & & \leqslant & 200 \\ & x_1 & & \leqslant & 200 \\ & \ddots & & \ldots & \ldots \\ & & x_{n-1} & \leqslant & 200 \\ x_0 + x_1 & \ldots + & x_{n-1} & \leqslant & 200(n-1)+100 \\ x_0 + x_1 & \ldots + & x_{n-1} & \geqslant & 100 \\ x_0 & & & \geqslant & 100 \\ x_0 & & & \geqslant & 0 \\ & x_1 & & \geqslant & 0 \\ & \ddots & & \ldots & \ldots \\ & & x_{n-1} & \geqslant & 0 \end{cases}. \tag{15}$$

Количество неравенств m в этой системе вычисляется по формуле $m = 2n + 2$. В качестве вектора коэффициентов целевой функции был взят вектор

$$c = (10n, 10(n-1), 10(n-2), \ldots, 10). \tag{16}$$

В этом случае точным решением задачи (1) для любой размерности $n \geqslant 2$ будет являться точка

$$\bar{x} = (200, 200, \ldots, 200, 100).$$

Таблица 1

Характеристики вычислительного кластера “Торнадо ЮУрГУ”

Количество процессорных узлов	480
Процессоры	Intel Xeon X5680 (6 cores 3.3 GHz)
Количество процессоров в узле	2
Оперативная память узла	24 GB DDR3
Соединительная сеть	InfiniBand QDR (40 Gbit/s)
Операционная система	Linux CentOS

Вычислительные эксперименты проводились на вычислительном кластере “Торнадо ЮУрГУ” [43], характеристики которого приведены в табл. 1. Результаты экспериментов представлены на рис. 4. Вычисления проводились для размерностей 5 000, 7 500 и 10 000. Количество неравенств, соответственно, составляло 10 002, 15 002 и 20 002. Эксперименты показали, что граница масштабируемости апекс-метода существенным образом зависит от размера задачи. При $n = 5000$ граница масштабируемости составила приблизительно 55 процессорных узлов. Для задачи размерности $n = 7500$ эта граница увеличилась до 80 узлов, а на задаче размерности $n = 10000$ она достигла 100 узлов. Дальнейшее увеличение размерности задачи приводило к нехватке оперативной памяти на процессорных узлах. Следует отметить, что вычисления производились с двойной точностью, при которой вещественное число представляется в формате с плавающей точкой и занимает 64 разряда. Попытка перейти на одинарную точность, требующую только 32 разряда для представления вещественного числа, оказалась неудачной из-за операции вычисления псевдопроекции, используемой на шаге 4 алгоритма 2. В этом случае алгоритм 1, вычисляющий псевдопроекцию, переставал сходиться. Эксперименты также показали, что параметр σ, определяющий в соответствии с формулой (8) удаленность точки апекса z от многогранника M (рис. 1), при больших значениях мало влияет на общее время решения задачи. Для модельного примера (15) и целевой функции с коэффициентами из формулы (16) параметр σ не может быть меньше $200n$, так как в этом случае точка апекса z окажется внутри многогранника M. Если точка апекса находится на недостаточно большом расстоянии от многогранника, то ее псевдопроекция оказывается внутренней точкой некоторой грани. Если же точку апекса взять достаточно далеко (в экспериментах использовалось значение $\sigma = 20000n$), то псевдопроекция в исследуемом примере всегда попадала в одну из вершин многогранника. Также отметим, что во всех случаях приближения $u_0, u_1, u_2, \ldots$ оказывались вершинами многогранника. Вычислительный эксперимент показал, что более 99% времени решения задачи ЛП апекс-методом приходится на вычисление псевдопроекций (шаг 4 алгоритма 2). При этом вычисление одного приближения u_k для задачи размерности $n = 10000$ на 100 процессорных узлах составило 44 минуты.

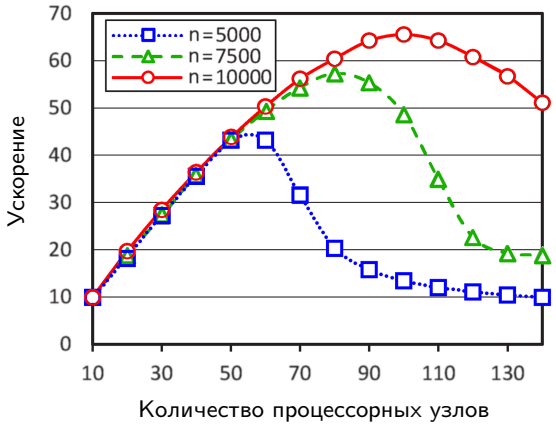

Рис. 4. Эксперименты по исследованию масштабируемости алгоритма 2

4. Заключение. В статье предложен новый масштабируемый итерационный метод решения задачи линейного программирования (ЛП), получивший название “апекс-метод”. Апекс-метод строится по схеме предиктор-корректор и состоит из двух фаз. На первой фазе, называемой *Quest*, происходит поиск допустимой точки задачи ЛП с использованием метода псевдопроекций. На второй фазе, называемой *Target*, вычисляется последовательность точек на поверхности многогранника, ограничивающего допустимую область, сходящаяся к точному решению задачи ЛП. Указанный метод реализован в виде программы на языке C++ с использованием библиотеки параллельного программирования MPI. Описаны вычислительные эксперименты по решению больших задач ЛП на кластерной вычислительной системе. Проведенные эксперименты показали, что апекс-метод масштабируется с ростом размерности задачи. Сильной стороной

апекс-метода является его “самокорректируемость” по отношению к возникающим погрешностям вычислений. Апекс-метод также потенциально может использоваться для решения нестационарных задач ЛП. Недостатком метода является высокая вычислительная сложность построения псевдопроекций.

В рамках дальнейших исследований мы планируем решить следующие задачи.

1. Выполнить математическое доказательство сходимости апекс-метода.
2. Выполнить тестирование апекс-метода на случайных задачах ЛП, генерируемых специальным алгоритмом.
3. Решить с использованием апекс-метода ряд эталонных задач из репозитория Netlib-Lp [44, 45].
4. Использовать апекс-метод для генерации тестового набора данных для разработки и обучения сверточных нейронных сетей, способных в кооперации с суперкомпьютером быстро решать сверхбольшие задачи ЛП.

Исследование выполнено при финансовой поддержке РФФИ (№ 20-07-00092 а), Правительства РФ в соответствии с Постановлением № 211 от 16.03.2013 г. (соглашение № 02.A03.21.0011) и Министерства науки и высшего образования РФ (государственное задание FENU–2020–0022).

Работа рекомендована Программным комитетом международной конференции “Суперкомпьютерные дни в России” (https://russianscdays.org/agenda/accepted/vak).

СПИСОК ЛИТЕРАТУРЫ

1. *Jagadish H.V., Gehrke J., Labrinidis A., et al.* Big data and its technical challenges // Communications of the ACM. 2014. **57**, N 7. 86–94.
2. *Hartung T.* Making big sense from big data // Frontiers in Big Data. 2018. **1**. doi 10.3389/fdata.2018.00005.
3. *Соколинская И.М., Соколинский Л.Б.* О решении задачи линейного программирования в эпоху больших данных // Параллельные вычислительные технологии — XI международная конференция, ПаВТ’2017, г. Казань, 3–7 апреля 2017 г. Челябинск: Издательский центр ЮУрГУ, 2017. 471–484.
4. *Chung W.* Applying large-scale linear programming in business analytics // Proc. 2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM). New York: IEEE Press, 2016. 1860–1864.
5. *Gondzio J., Gruca J.A., Hall J.A.J., et al.* Solving large-scale optimization problems related to Bell’s Theorem // Journal of Computational and Applied Mathematics. 2014. **263**. 392–404.
6. *Sodhi M.S.* LP modeling for asset-liability management: a survey of choices and simplifications // Operations Research. 2005. **53**, N 2. 181–196.
7. *Brogaard J., Hendershott T., Riordan R.* High-frequency trading and price discovery // Review of Financial Studies. 2014. **27**, N 8. 2267–2306.
8. *Bixby R.E.* Solving real-world linear programs: a decade and more of progress // Operations research. 2002. **50**, N 1. 3–15.
9. *Dongarra J., Gottlieb S., Kramer W.T.C.* Race to exascale // Computing in Science & Engineering. 2019. **21**, N 1. 4–5.
10. *Dantzig G.B.* Linear programming and extensions. Princeton: Princeton Univ. Press, 1998.
11. *Klee V., Minty G.J.* How good is the simplex algorithm? // Inequalities–III. Vol. 3. New York: Academic Press, 1972. 159–175.
12. *Jeroslow R.G.* The simplex algorithm with the pivot rule of maximizing criterion improvement // Discrete Mathematics. 1973. **4**, N 4. 367–377.
13. *Zadeh N.* A bad network problem for the simplex method and other minimum cost flow algorithms // Mathematical Programming. 1973. **5**, N 1. 255–266.
14. *Bartels R.H., Stoer J., Zenger Ch.* A realization of the simplex method based on triangular decompositions // Handbook for Automatic Computation. Vol. II: Linear Algebra. Berlin: Springer, 1971. 152–190.
15. *Tolla P.* A survey of some linear programming methods // Concepts of Combinatorial Optimization. Hoboken: Wiley, 2014. 157–188.
16. *Mamalis B., Pantziou G.* Advances in the parallelization of the simplex method // Lecture Notes in Computer Science. Vol. 9295. Cham: Springer, 2015. 281–307.
17. *Lubin M., Hall J.A.J., Petra C.G., Anitescu M.* Parallel distributed-memory simplex for large-scale stochastic LP problems // Computational Optimization and Applications. 2013. **55**, N 3. 571–596.
18. *Хачиян Л.Г.* Полиномиальный алгоритм в линейном программировании // Доклады АН СССР. 1979. **244**, № 5. 1093–1096.
19. *Шор Н.З.* Метод отсечения с растяжением пространства для решения задач выпуклого программирования // Кибернетика. 1977. **13**, № 1. 94–95.

20. *Юдин Д.Б., Немировский А.С.* Информационная сложность и эффективные методы решения выпуклых экстремальных задач // Экономика и математические методы. 1976. **12**, № 2. 357–369.
21. *Karmarkar N.* A new polynomial-time algorithm for linear programming // Combinatorica. 1984. **4**, N 4. 373–395.
22. *Fathi-Hafshejani S., Mansouri H., Peyghami M.R., Chen S.* Primal-dual interior-point method for linear optimization based on a kernel function with trigonometric growth term // Optimization. 2018. **67**, N 10. 1605–1630.
23. *Asadi S., Mansouri H.* A Mehrotra type predictor–corrector interior-point algorithm for linear programming // Numerical Algebra, Control and Optimization. 2019. **9**, N 2. 147–156.
24. *Yuan Y.* Implementation tricks of interior-point methods for large-scale linear programs // Proc. SPIE **8285**. Graphic and Image Processing. 2011. doi: 10.1117/12.913019.
25. *Kheirfam B., Haghighi M.* A full-Newton step infeasible interior-point method for linear optimization based on a trigonometric kernel function // Optimization. 2016. **65**, N 4. 841–857.
26. *Xu Y., Zhang L., Zhang J.* A full-modified-Newton step infeasible interior-point algorithm for linear optimization // Journal of Industrial and Management Optimization. 2016. **12**, N 1. 103–116.
27. *Roos C., Terlaky T., Vial J.-P.* Interior point methods for linear optimization. New York: Springer, 2005.
28. *Sokolinskaya I.* Parallel method of pseudoprojection for linear inequalities // Parallel Computational Technologies. Vol. 910. Cham: Springer, 2018. 216–231.
29. *Hafsteinsson H., Levkovitz R., Mitra G.* Solving large scale linear programming problems using an interior point method on a massively parallel SIMD computer // Parallel Algorithms and Applications. 1994. **4**, N 3–4. 301–316.
30. *Karypis G., Gupta A., Kumar V.* A parallel formulation of interior point algorithms // Proc. 1994 ACM/IEEE Conference on High Performance Computing, Networking, Storage, and Analysis. Los Alamitos: IEEE Press, 1994. 204–213.
31. *Ершова А.В., Соколинская И.М.* О сходимости масштабируемого алгоритма построения псевдопроекции на выпуклое замкнутое множество // Вестник ЮУрГУ. Серия: Математическое моделирование и программирование. 2011. № 37. 12–21.
32. *Соколинская И.М., Соколинский Л.Б.* Модифицированный следящий алгоритм для решения нестационарных задач линейного программирования на кластерных вычислительных системах с многоядерными ускорителями // Суперкомпьютерные дни в России: Труды международной конференции (26–27 сентября 2016 г., г. Москва). Москва: Изд-во МГУ, 2016. 294–306.
33. *Соколинская И.М., Соколинский Л.Б.* Исследование масштабируемости модифицированного алгоритма Чиммино для линейных неравенств // Суперкомпьютерные дни в России: Труды международной конференции (24–25 сентября 2018 г., г. Москва). Москва: Изд-во МГУ, 2018. 673–683.
34. *Ерёмин И.И.* Фейеровские методы для задач выпуклой и линейной оптимизации. Челябинск: Издательский центр ЮУрГУ, 2009.
35. *Соколинская И.М., Соколинский Л.Б.* Масштабируемый алгоритм для решения нестационарных задач линейного программирования // Вычислительные методы и программирование: новые вычислительные технологии. 2018. **19**. 540–550.
36. *Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P.* Numerical recipes: the art of scientific computing. New York: Cambridge Univ. Press, 2007.
37. *Censor Y., Elfving T., Herman G.T., Nikazad T.* On diagonally relaxed orthogonal projection methods // SIAM Journal on Scientific Computing. 2008. **30**, N 1. 473–504.
38. *Соколинский Л.Б., Соколинская И.М.* Параллельный алгоритм решения нестационарных систем линейных неравенств // Параллельные вычислительные технологии — XIV международная конференция (ПаВТ'2020), г. Пермь, 31 марта–2 апреля 2020 г. Челябинск: Издательский центр ЮУрГУ, 2020. 275–286.
39. *Ежова Н.А., Соколинский Л.Б.* BSF: модель параллельных вычислений для многопроцессорных систем с распределенной памятью // Параллельные вычислительные технологии — XII международная конференция, ПаВТ'2018, г. Ростов-на-Дону, 2–6 апреля 2018 г. Челябинск: Издательский центр ЮУрГУ, 2018. 253–265.
40. *Ежова Н.А., Соколинский Л.Б.* Модель параллельных вычислений для многопроцессорных систем с распределенной памятью // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2018. **7**, № 2. 32–49.
41. *Sokolinsky L.B.* BSF-skeleton [Electronic resource]. 2019. https://github.com/leonid-sokolinsky/BSF-skeleton (accessed: 27.08.2020).
42. *Ежова Н.А., Соколинский Л.Б.* Модель параллельных вычислений BSF-MR // Системы управления и информационные технологии. 2019. № 3. 15–21.
43. *Kostenetskiy P.S., Safonov A.Y.* SUSU supercomputer resources // Proc. 10th Annual International Scientific Conference on Parallel Computing Technologies (PCT 2016). CEUR Workshop Proceedings. Vol. 1576. 2016. 561–573.
44. *Gay D.M.* Netlib-Lp [Electronic resource]. URL: http://www.netlib.org/lp/ (accessed: 27.08.2020).
45. *Koch T.* The final NETLIB-LP results // Operations Research Letters. 2004. **32**, N 2. 138–142.

On an Iterative Method for Solving Linear Programming Problems on Cluster Computing Systems

I. M. Sokolinskaya[1], **L. B. Sokolinsky**[2]

[1] *South Ural State University, Faculty of Computational Mathematics and Informatics; prospekt Lenina 76, Chelyabinsk, 454080, Russia; Ph.D., Associate Professor, e-mail: irina.sokolinskaya@susu.ru*

[2] *South Ural State University; prospekt Lenina 76, Chelyabinsk, 454080, Russia; Dr. Sci., Professor, Vice-Rector for Informatization, e-mail: Leonid.Sokolinsky@susu.ru; ORCID 0000–0001–9997–3918*

Received June 28, 2020

Abstract: The paper is devoted to a new method for solving large-scale linear programming (LP) problems. This method is called the apex-method. The apex-method uses the predictor–corrector framework. The predictor step calculates a point belonging to the feasible region of the LP problem. The corrector step calculates a sequence of points converging to the exact solution of the LP problem. The paper gives a formal description of the apex-method and provides information about its parallel implementation in C++ language using the MPI library. The results of large-scale computational experiments on a cluster computing system to study the scalability of the apex method are discussed.

Keywords: linear programming, large-scale problems, apex-method, predictor–corrector framework, iterative method, parallel algorithm, cluster computing system

References

1. H. V. Jagadish, J. Gehrke, A. Labrinidis, et al., "Big Data and Its Technical Challenges," Commun. ACM **57** (7), 86–94 (2014).

2. T. Hartung, "Making Big Sense from Big Data," Frontiers in Big Data **1** (2018). doi 10.3389/fdata.2018.00005

3. I. M. Sokolinskaya and L. B. Sokolinsky, "On the Solution of Linear Programming Problem in the Era of Big Data," in *Proc. Int. Conf. on Parallel Computational Technologies, Kazan, Russia, April 3–7, 2017* (South Ural State Univ., Chelyabinsk, 2017), pp. 471–484.

4. W. Chung, "Applying Large-Scale Linear Programming in Business Analytics," in *Proc. IEEE Int. Conf. on Industrial Engineering and Engineering Management (IEEM), Singapore, December 6–9, 2015* (IEEE Press, New York, 2016), pp. 1860–1864.

5. J. Gondzio, J. A. Gruca, J. A. J. Hall, et al., "Solving Large-Scale Optimization Problems Related to Bell's Theorem," J. Comput. Appl. Math. **263**, 392–404 (2014).

6. M. S. Sodhi, "LP Modeling for Asset-Liability Management: A Survey of Choices and Simplifications," Oper. Res. **53** (2), 181–196 (2005).

7. J. Brogaard, T. Hendershott, and R. Riordan, "High-Frequency Trading and Price Discovery," Rev. Financ. Stud. **27** (8), 2267–2306 (2014).

8. R. E. Bixby, "Solving Real-World Linear Programs: A Decade and More of Progress," Oper. Res. **50** (1), 3–15 (2002).

9. J. Dongarra, S. Gottlieb, and W. T. C. Kramer, "Race to Exascale," Comput. Sci. Eng. **21** (1), 4–5 (2019).

10. G. B. Dantzig, *Linear Programming and Extensions* (Princeton Univ. Press, Princeton, 1998).

11. V. Klee and G. J. Minty, "How Good is the Simplex Algorithm?," in *Inequalities III* (Academic, New York, 1972), Vol. 3, pp. 159–175.

12. R. G. Jeroslow, "The Simplex Algorithm with the Pivot Rule of Maximizing Criterion Improvement," Discrete Math. **4** (4), pp. 367–377 (1973).

13. N. Zadeh, "A Bad Network Problem for the Simplex Method and Other Minimum Cost Flow Algorithms," in *Mathematical Programming* (Springer, Berlin, 1973), Vol. 5, pp. 255–266.

14. R. H. Bartels, J. Stoer, and Ch. Zenger, "A Realization of the Simplex Method Based on Triangular Decompositions," in *Handbook for Automatic Computation. Vol. II: Linear Algebra* (Springer, Berlin, 1971), pp. 152–190.

15. P. Tolla, "A Survey of Some Linear Programming Methods," in *Concepts of Combinatorial Optimization* (Wiley, Hoboken, 2014), pp. 157–188.

16. B. Mamalis and G. Pantziou, "Advances in the Parallelization of the Simplex Method," in *Lecture Notes in Computer Science* (Springer, Cham, 2015), Vol. 9295, pp. 281–307.

17. M. Lubin, J. A. J. Hall, C. G. Petra, and M. Anitescu, "Parallel Distributed-Memory Simplex for Large-Scale Stochastic LP Problems," Comput. Optim. Appl. **55** (3), 571–596 (2013).

18. L. G. Khachiyan, "A Polynomial Algorithm in Linear Programming," Dokl. Akad. Nauk. SSSR **244** (5), 1093–1096 (1979) [Sov. Math. Dokl. **20**, 191–194 (1979)].

19. N. Z. Shor, "Cut-off Method with Space Extension in Convex Programming Problems," Kibernetika, No. 1, 94–95 (1977) [Cybernetics **13** (1), 94–95 (1977)].

20. D. B. Yudin and A. S. Nemirovskii, "Informational Complexity and Effective Methods for the Solution of Convex Extremal Problems," Ekonomika Mat. Metody **12** (2), 357–369 (1976).

21. N. Karmarkar, "A New Polynomial-Time Algorithm for Linear Programming," Combinatorica **4** (4), 373–395 (1984).

22. S. Fathi-Hafshejani, H. Mansouri, M. R. Peyghami, and S. Chen, "Primal-Dual Interior-Point Method for Linear Optimization Based on a Kernel Function with Trigonometric Growth Term," Optimization **67** (10), 1605–1630 (2018).

23. S. Asadi and H. Mansouri, "A Mehrotra Type Predictor–Corrector Interior-Point Algorithm for Linear Programming," Numer. Algebra Control Optim. **9** (2), 147–156 (2019).

24. Y. Yuan, "Implementation Tricks of Interior-Point Methods for Large-Scale Linear Programs," Proc. SPIE **8285**, Graphic and Image Processing (2011). doi 10.1117/12.913019

25. B. Kheirfam and M. Haghighi, "A Full-Newton Step Infeasible Interior-Point Method for Linear Optimization Based on a Trigonometric Kernel Function," Optimization **65** (4), 841–857 (2016).

26. Y. Xu, L. Zhang, and J. Zhang, "A Full-Modified-Newton Step Infeasible Interior-Point Algorithm for Linear Optimization," J. Ind. Manag. Optim. **12** (1), 103–116 (2016).

27. C. Roos, T. Terlaky, and J.-P. Vial, *Interior Point Methods for Linear Optimization* (Springer, New York, 2005).

28. I. Sokolinskaya, "Parallel Method of Pseudoprojection for Linear Inequalities," in *Parallel Computational Technologies* (Springer, Cham, 2018), Vol. 910, pp. 216–231.

29. H. Hafsteinsson, R. Levkovitz, and G. Mitra, "Solving Large Scale Linear Programming Problems Using an Interior Point Method on a Massively Parallel SIMD Computer," Parallel Algorithms Appl. **4** (3–4), 301–316 (1994).

30. G. Karypis, A. Gupta, V. Kumar, "A Parallel Formulation of Interior Point Algorithms," in *Proc. 1994 ACM/IEEE Conf. on High Performance Computing, Networking, Storage, and Analysis, Washington, DC, USA, November 14–18, 1994* (IEEE Press, Los Alamitos, 1994), pp. 204–213.

31. A. V. Ershova, I. M. Sokolinskaya, "About Convergence of Scalable Algorithm of Construction Pseudoprojection on Convex Closed Set," Vestn. South Ural State Univ. Ser. Mat. Model. Program., No. 37, 12–21 (2011).

32. I. M. Sokolinskaya and L. B. Sokolinsky, "Revised Pursuit Algorithm for Solving Unstable Linear Programming Problems on Modern Computing Clusters with Manycore Accelerators," in *Proc. Int. Conf. on Russian Supercomputing Days, Moscow, Russia, September 26–27, 2016* (Mosk. Gos. Univ., Moscow, 2016), pp. 294–306.

33. I. M. Sokolinskaya and L. B. Sokolinsky, "Scalability Evaluation of a Modified Cimmino Algorithm for Linear Inequalities," in *Proc. Int. Conf. on Russian Supercomputing Days, Moscow, Russia, September 24–25, 2018* (Mosk. Gos. Univ., Moscow, 2018), pp. 673–683.

34. I. I. Eremin, *Fejer Methods for Problems of Convex and Linear Optimization* (South Ural State Univ., Chelyabinsk, 2009) [in Russian].

35. I. M. Sokolinskaya and L. B. Sokolinsky, "A Scalable Algorithm for Solving Non-Stationary Linear Programming Problems," Vychisl. Metody Programm. **19**, 540–550 (2018).

36. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing* (Cambridge Univ. Press, New York, 2007).

37. Y. Censor, T. Elfving, G. T. Herman, and T. Nikazad, "On Diagonally Relaxed Orthogonal Projection Methods," SIAM J. Sci. Comput. **30** (1), 473–504 (2008).

38. L. B. Sokolinsky and I. M. Sokolinskaya, "Parallel Algorithm for Solving Non-Stationary Systems of Linear Inequalities," in *Proc. 14th Int. Conf. on Parallel Computational Technologies, Perm, Russia, March 31–April 2, 2020* (South Ural State Univ., Chelyabinsk, 2020), pp. 275–286.

39. N. A. Ezhova and L. B. Sokolinsky, "BSF: A model of Parallel Computation for Multiprocessor Systems with Distributed Memory," in *Proc. Int. Conf. on Parallel Computational Technologies, Rostov-on-Don, Russia, April 2–6, 2018* (South Ural State Univ., Chelyabinsk, 2018), pp. 253–265.

40. N. A. Ezhova and L. B. Sokolinsky, "Parallel Computational Model for Multiprocessor Systems with Distributed Memory," Vestn. South Ural State Univ. Ser. Vychisl. Mat. Inf. **7** (2), 32–49 (2018).

41. L. B. Sokolinsky, "BSF-skeleton," (2019) https://github.com/leonid-sokolinsky/BSF-skeleton. Cited August 27, 2020.

42. N. A. Ezhova and L. B. Sokolinsky, "BSF-MR Parallel Computation Model," Sistemy Upravl. Inf. Tekhnol., No. 3, 15–21 (2019).

43. P. S. Kostenetskiy and A. Y. Safonov, "SUSU Supercomputer Resources," in *Proc. 10th Annual Int. Scientific Conf. on Parallel Computing Technologies (PCT 2016), Arkhangelsk, Russia, March 29–31, 2016* CEUR Workshop Proc. **1576**, 561–573 (2016).

44. D. M. Gay, "Netlib-Lp," http://www.netlib.org/lp/. Cited August 27, 2020.

45. T. Koch, "The Final NETLIB-LP Results," Oper. Res. Lett. **32** (2), 138–142 (2004).