

FRaGenLP: генератор случайных задач линейного программирования для кластерных вычислительных систем*

Л.Б. Соколинский, И.М. Соколинская

Южно-Уральский государственный университет

В статье рассматривается масштабируемый алгоритм FRaGenLP для генерации больших совместных случайных задач линейного программирования произвольной размерности n на кластерных вычислительных системах. Для обеспечения совместности и ограниченности допустимой области система ограничений включает в себя $2n+1$ стандартных неравенств, называемых опорными. Случайные неравенства добавляются в систему последовательно так, чтобы сохранялась совместность ограничений. Кроме этого, вводятся две метрики «похожести», которые препятствуют добавлению нового случайного неравенства, «похожего» на какое-либо из уже включенных в систему, включая опорные. Также отклоняются случайные неравенства, которые при фиксированной целевой функции не влияют на решение опорной задачи линейного программирования. Параллельная реализация алгоритма FRaGenLP выполнена на языке C++ с использованием параллельного BSF-каркаса, инкапсулирующего в проблемно-независимой части своего кода все аспекты, связанные с распараллеливанием программы на базе библиотеки MPI. Приводятся результаты масштабных вычислительных экспериментов на кластерной вычислительной системе, подтверждающие эффективность использованного подхода.

Ключевые слова: случайная задача линейного программирования, генератор задач, FRaGenLP, кластерные вычислительные системы, BSF-каркас.

1. Введение

Эпоха больших данных [1,2] породила задачи линейного программирования (ЛП) сверхбольших размерностей [3]. Подобные задачи возникают в экономике, промышленности, логистике, статистике, квантовой физике и других областях. Решение таких сверхбольших задач невозможно без масштабируемых параллельных алгоритмов, ориентированных на кластерные вычислительные системы. В соответствии с этим в последние годы интенсифицировались усилия по разработке новых и модернизации известных параллельных алгоритмов решения задач ЛП. В качестве примеров можно привести работы [4–8]. При разработке новых масштабируемых алгоритмов для решения сверхбольших задач линейного программирования возникает необходимость их тестирования на известных и случайных задачах. Одним из наиболее известных репозиторий больших задач линейного программирования является Netlib-Lp [9]. Данный репозиторий считается эталонным при апробации новых алгоритмов решения задач линейного программирования. Однако при отладке решателей часто возникает необходимость в генерации случайных задач ЛП с определенными характеристиками, из которых основными являются размерность пространства и количество ограничений. В работе [10] был предложен один из первых методов генерации случайных задач ЛП с известными решениями. Метод позволяет генерировать тестовые задачи произвольного размера с широким диапазоном числовых характеристик. Идея метода заключается в том, что за основу берется задача ЛП с известным решением, которая затем модифицируется случайным образом так, чтобы решение не изменилось. Основным недостатком этого метода является то, что предварительная фиксация оптимального решения в значительной мере ограничивает случайный характер итоговой задачи ЛП. В [11] предложен генератор GENGUB, позволяющий строить случайные задачи ЛП с известным решением и заданными характеристиками, та-

* Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 20-07-00092-а.

кими как размер задачи, плотность ненулевых значений матрицы коэффициентов, степень вырожденности, количество избыточных неравенств и др. Отличительной особенностью генератора GENGUB является введение *обобщенных верхних границ*, представляющих собой совокупность ограничений, в которых каждая переменная появляется как минимум один раз (имеет ненулевой коэффициент). Указанный метод имеет тот же недостаток, что и предыдущий: предварительная фиксация оптимального решения в значительной мере ограничивает случайный характер результирующей задачи ЛП. В статье [12] предложен метод генерации случайных задач ЛП с заранее выбранным типом решения: ограниченное или неограниченное, единственное или множественное. В зависимости от выбранного типа решения конструируется допустимая область задачи ЛП как объединение линейного многообразия, конуса и многогранника. Каждая из перечисленных структур генерируется с помощью случайных векторов с целочисленными координатами. Далее генерируется целевая функция, приводящая к решению требуемого типа. Описанный генератор задач ЛП используется в основном для учебных целей и мало подходит для тестирования новых алгоритмов линейного программирования в силу ограниченности многообразия генерируемых задач.

В данной статье предлагается альтернативный способ генерации случайных задач ЛП, особенность которого состоит в том, что генерируются совместные задачи заданной размерности с неизвестным решением. Такая задача подается на вход решателя. Полученное решение проверяется программой-валидатором [13], которая сертифицирует решение. Предложенный метод генерации случайных задач ЛП реализован в виде параллельной программы *FRaGenLP* (*Feasible Random Generator of LP*) для кластерных вычислительных систем. Статья организована следующим образом. В разделе 2 дается формальное описание предлагаемого метода генерации случайных задач ЛП и приводится последовательная версия алгоритма FRaGenLP. В разделе 3 рассматривается параллельная версия алгоритма FRaGenLP. В разделе 4 предлагается ее реализация с использованием параллельного BSF-каркаса и приводятся результаты масштабных вычислительных экспериментов на кластерной вычислительной системе, подтверждающие эффективность предложенного подхода. В разделе 5 суммируются полученные результаты и раскрываются планы по использованию генератора FRaGenLP для разработки искусственной нейронной сети, способной решать задачи ЛП большой размерности.

2. Метод генерации случайных задач ЛП

Метод генерации случайных задач ЛП, используемый в программе FRaGenLP, позволяет генерировать случайные совместные ограниченные задачи ЛП произвольной размерности n , решение которых заранее неизвестно. Для обеспечения корректности задачи ЛП система ограничений включает в себя следующие *опорные неравенства*:

$$\left\{ \begin{array}{llll} x_1 & & & \leq \alpha \\ & x_2 & & \leq \alpha \\ & & \ddots & \dots \dots \\ & & & x_n \leq \alpha \\ -x_1 & & & \leq 0 \\ & -x_2 & & \leq 0 \\ & & \ddots & \dots \dots \\ & & & -x_n \leq 0 \\ x_1 & +x_2 & \dots & +x_n \leq \alpha(n-1)+100 \end{array} \right. \quad (1)$$

Здесь $\alpha \in \mathbb{R}_{>0}$ – положительный масштабирующий коэффициент, являющийся параметром программы FRaGenLP. Количество опорных неравенств равно $2n+1$. Количество случайных неравенств определяется параметром $d \in \mathbb{Z}_{\geq 0}$. Общее количество неравенств m определяется по формуле

$$m = 2n + 1 + d. \quad (2)$$

Коэффициенты целевой функции задаются вектором

$$c = \theta(n, n-1, n-2, \dots, 1), \quad (3)$$

где $\theta \in \mathbb{R}_{>0}$ – положительный постоянный множитель, являющийся параметром программы FRaGenLP и удовлетворяющий условию

$$\theta \leq \frac{\alpha}{2}. \quad (4)$$

Везде далее будем предполагать, что необходимо найти допустимую точку, в которой достигается максимум целевой функции. При количестве случайных неравенств $d = 0$ генерируется задача ЛП, ограничения которой включают только опорные неравенства (1). В этом случае задача ЛП имеет следующее единственное решение:

$$\bar{x} = (\alpha, \dots, \alpha, \alpha/2). \quad (5)$$

Коэффициенты случайного неравенства $a_i = (a_{i1}, \dots, a_{in})$ и правая часть b_i вычисляются с помощью функции $\text{rand}(l, r)$, генерирующей случайное вещественное число из интервала $[l, r]$ ($l, r \in \mathbb{R}; l < r$), и функции $\text{rsgn}()$, случайным образом выбирающей число из множества $\{1, -1\}$:

$$\begin{aligned} a_{ij} &:= \text{rsgn}() \cdot \text{rand}(0, a_{\max}), \\ b_i &:= \text{rsgn}() \cdot \text{rand}(0, b_{\max}). \end{aligned} \quad (6)$$

Здесь $a_{\max}, b_{\max} \in \mathbb{R}_{>0}$ – константы, являющиеся параметрами программы FRaGenLP. В качестве знака неравенства всегда выбирается $\leq$. Введем следующие обозначения:

$$f(x) = \langle c, x \rangle; \quad (7)$$

$$h_{\text{cntr}} = (\alpha/2, \dots, \alpha/2); \quad (8)$$

$$\text{dist}(a_i, b_i, h_{\text{cntr}}) = \frac{|\langle a_i, h_{\text{cntr}} \rangle - b_i|}{\|a_i\|}; \quad (9)$$

$$\pi(x, a_i, b_i) = x - \frac{\langle a_i, x \rangle - b_i}{\|a_i\|^2} a_i. \quad (10)$$

Формула (7) определяет целевую функцию задачи ЛП. Здесь и далее $\langle \cdot, \cdot \rangle$ обозначает скалярное произведение векторов. Формула (8) определяет центральную точку *ограничивающего гиперкуба*, задаваемого первыми $2n$ неравенствами системы (1). Формула (9) определяет функцию $\text{dist}(a_i, b_i, h_{\text{cntr}})$, вычисляющую расстояние от гиперплоскости $\langle a_i, x \rangle = b_i$ до центра ограничивающего гиперкуба. Здесь и далее $\|\cdot\|$ обозначает евклидову норму. Формула (10) определяет вектор-функцию, вычисляющую ортогональную проекцию точки x на гиперплоскость $\langle a_i, x \rangle = b_i$.

Случайное неравенство $\langle a_i, x \rangle \leq b_i$ получается путем генерации координат вектора a_i и свободного члена b_i с помощью датчика псевдослучайных дробно-рациональных чисел. Сгенерированное случайное неравенство добавляется в систему ограничений в том, и только в том случае, когда выполняются следующие условия:

$$\langle a_i, h_{\text{cntr}} \rangle \leq b_i; \quad (11)$$

$$\rho < \text{dist}(a_i, b_i, h_{\text{cntr}}) \leq \theta; \quad (12)$$

$$f(\pi(h_{ctr}, a_i, b_i)) > f(h_{ctr}); \quad (13)$$

$$\forall l \in \{1, \dots, i-1\} : \neg \text{like}(a_i, b_i, a_l, b_l). \quad (14)$$

Условие (11) требует, чтобы центр ограничивающего гиперкуба являлся допустимой точкой для добавляемого неравенства. Если это условие не выполняется, то вместо неравенства $\langle a_i, x \rangle \leq b_i$ можно взять неравенство $-\langle a_i, x \rangle \leq -b_i$. Условие (12) требует, чтобы расстояние от гиперплоскости $\langle a_i, x \rangle = b_i$ до центра h_{ctr} ограничивающего гиперкуба было больше ρ и меньше θ . Константа $\rho \in \mathbb{R}_{>0}$ является параметром программы FRaGenLP и должна удовлетворять условию $\rho < \theta$, где θ , в свою очередь, удовлетворяет условию (4). Условие (13) требует, чтобы значение целевой функции в точке проекции h_{ctr} на гиперплоскость $\langle a_i, x \rangle = b_i$ было больше, чем в самой точке h_{ctr} . Это условие в сочетании с (11) и (12) отсекает ограничения, которые не могут повлиять на решение задачи ЛП. Условие (14) требует, чтобы новое неравенство было *непохоже* на все ранее добавленные, включая опорные. Указанное условие использует булевскую функцию *like*, определяющую *похожесть* (*likeness*) двух неравенств $\langle a_i, x \rangle \leq b_i$ и $\langle a_l, x \rangle \leq b_l$ по следующей формуле:

$$\text{like}(a_i, b_i, a_l, b_l) = \left\| \frac{a_i}{\|a_i\|} - \frac{a_l}{\|a_l\|} \right\| < L_{max} \wedge \left| \frac{b_i}{\|a_i\|} - \frac{b_l}{\|a_l\|} \right| < S_{min}. \quad (15)$$

Константы $L_{max}, S_{min} \in \mathbb{R}_{>0}$ являются параметрами программы FRaGenLP. При этом параметр L_{max} должен удовлетворять условию

$$L_{max} \leq 0.7 \quad (16)$$

(обоснование этому ограничению будет дано ниже). В соответствии с формулой (15) неравенства $\langle a_i, x \rangle \leq b_i$ и $\langle a_l, x \rangle \leq b_l$ считаются похожими, если выполняются следующие два условия:

$$\left\| \frac{a_i}{\|a_i\|} - \frac{a_l}{\|a_l\|} \right\| < L_{max}; \quad (17)$$

$$\left| \frac{b_i}{\|a_i\|} - \frac{b_l}{\|a_l\|} \right| < S_{min}. \quad (18)$$

Условие (17) оценивает параллельность гиперплоскостей $\langle a_i, x \rangle = b_i$ и $\langle a_l, x \rangle = b_l$, ограничивающих допустимые области исходных неравенств. Для этого сначала вычисляются единичные векторы нормалей $e_i = a_i / \|a_i\|$ и $e_l = a_l / \|a_l\|$ к гиперплоскостям $\langle a_i, x \rangle = b_i$ и $\langle a_l, x \rangle = b_l$. Затем вычисляется норма разности единичных векторов нормалей: $\delta = \|e_i - e_l\|$. Если $\delta = 0$, то гиперплоскости параллельны. Если $0 < \delta < L_{max}$, то гиперплоскости считаются *почти параллельными*. Условие (18) оценивает «близость» гиперплоскостей $\langle a_i, x \rangle = b_i$ и $\langle a_l, x \rangle = b_l$ относительно друг друга. Для этого сначала вычисляются нормализованные свободные члены $\beta_i = b_i / \|a_i\|$ и $\beta_l = b_l / \|a_l\|$. Затем вычисляется модуль их разности $\sigma = |\beta_i - \beta_l|$. Если гиперплоскости параллельны и $\sigma = 0$, то эти гиперплоскости совпадают. Если $0 < \sigma < S_{min}$, то гиперплоскости считаются *почти близкими*. Два линейных неравенства в $\mathbb{R}^n$ считаются *похожими*, если соответствующие им гиперплоскости являются почти параллельными и почти близкими.

Ограничение (16) для параметра L_{max} основано на следующем утверждении.

Утверждение 1. Пусть заданы два единичных вектора $e, e' \in \mathbb{R}^n$ и угол $\varphi < \pi$ между ними. Тогда

$$\|e - e'\| = \sqrt{2(1 - \cos \varphi)}. \quad (19)$$

Доказательство. По определению нормы в евклидовом пространстве имеем

$$\|e - e'\| = \sqrt{\sum_j (e_j - e'_j)^2} = \sqrt{\sum_j (e_j^2 - 2e_j e'_j + e_j'^2)} = \sqrt{\sum_j e_j^2 - 2 \sum_j e_j e'_j + \sum_j e_j'^2} = \sqrt{1 - 2\langle e_j, e'_j \rangle + 1}.$$

Таким образом

$$\|e - e'\| = \sqrt{2(1 - \langle e_j, e'_j \rangle)}. \quad (20)$$

По определению угла в евклидовом пространстве для единичных векторов имеем

$$\langle e_j, e'_j \rangle = \cos \varphi.$$

Подставив правую часть в (20) получаем

$$\|e - e'\| = \sqrt{2(1 - \cos \varphi)}.$$

Утверждение доказано.

Разумно считать два единичных вектора e, e' почти параллельными, если угол φ между ними меньше, чем $\pi/4$. В этом случае согласно (19) имеем

$$\|e - e'\| < \sqrt{2\left(1 - \cos \frac{\pi}{4}\right)}.$$

Учитывая, что $\cos(\pi/4) \approx 0.707$, получаем требуемую оценку:

$$\|e - e'\| < 0.7.$$

На рис. 1 представлена случайная задача, сгенерированная программой FRaGenLP в соответствии с выше описанным методом для размерности $n = 2$ с использованием следующих значений параметров: $d = 5$, $\alpha = 200$, $\theta = 100$, $\rho = 50$, $S_{min} = 100$, $L_{max} = 0.35$, $a_{max} = 1000$, $b_{max} = 10000$. Фиолетовым цветом на рисунке обозначена прямая, определяемая коэффициентами целевой функции; черными линиями обозначены прямые, соответствующие опорным неравенствам; красными линиями обозначены прямые, соответствующие случайным неравенствам. Для наглядности на рисунке также обозначены зелеными пунктирными линиями большая и малая окружности, задаваемые уравнениями $(x_1 - 100)^2 + (x_2 - 100)^2 = 100^2$ и $(x_1 - 100)^2 + (x_2 - 100)^2 = 50^2$ соответственно. Любая случайная прямая должна пересекать большую окружность и не пересекать малую окружность в соответствии с условием (12). Полупрозрачным красным цветом выделена область допустимых точек получившейся задачи линейного программирования.

Описанный метод представлен в виде последовательного алгоритма 1. На шаге 1 вводятся параметры алгоритма. На шаге 2 счетчику случайных неравенств присваивается значение 0. На шаге 3 формируется пустой список строк матрицы A . На шаге 4 формируется пустой список строк свободных членов B . На шаге 5 в списки A и B добавляются коэффициенты и свободные члены опорных неравенств (1). На шаге 6 проверяется количество случайных неравенств d :

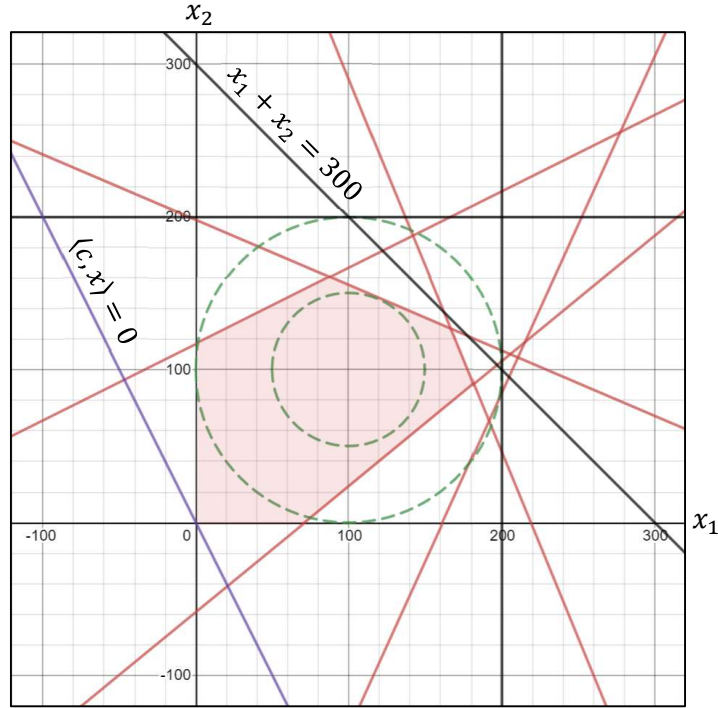


Рис. 1. Случайная задача линейного программирования для $n = 2, d = 5, \alpha = 200, \theta = 100, \rho = 50, S_{\min} = 100, L_{\max} = 0.35, a_{\max} = 1000, b_{\max} = 10000$ (нарисовано с помощью графического калькулятора Desmos [14]).

Алгоритм 1. Последовательный алгоритм генерации случайной задачи ЛП

```

1: input  $n, d, \alpha, \theta, \rho, S_{\min}, L_{\max}, a_{\max}, b_{\max}$ 
2:  $k := 0$ 
3:  $A := []$ 
4:  $B := []$ 
5:  $AddSupport(A, B)$ 
6: for  $j = n \dots 1$  do  $c_j := \theta \cdot j$ 
7: if  $d = 0$  goto 20
8: for  $j = 1 \dots n$  do  $a_j := rsign() \cdot rand(0, a_{\max})$ 
9:  $b := rsign() \cdot rand(0, b_{\max})$ 
10: if  $\langle a, h_{\text{ctr}} \rangle \leq b$  goto 13
11: for  $j = 1 \dots n$  do  $a_j := -a_j$ 
12:  $b := -b$ 
13: if  $dist(a, b, h_{\text{ctr}}) < \rho$  or  $dist(a, b, h_{\text{ctr}}) > \theta$  goto 8
14: if  $f(\pi(h_{\text{ctr}}, a_i, b_i)) > f(h_{\text{ctr}})$  goto 8
15: for all  $(\bar{a}, \bar{b}) \in (A, B)$  do if  $like(a, b, \bar{a}, \bar{b})$  goto 8
16:  $A := A \# a$ 
17:  $B := B \# b$ 
18:  $k := k + 1$ 
19: if  $k < d$  goto 8
20: output  $A, B, c$ 
21: stop

```

если $d = 0$, то происходит вывод только опорных неравенств, после чего алгоритм заканчивает свою работу. На шаге 7 генерируются коэффициенты целевой функции в соответствии с формулой (3). На шагах 8 и 9 генерируются коэффициенты и свободный член нового случайного неравенства. На шаге 10 проверяется условие (11). Если это условие не верно, то знаки коэффициентов и свободного члена меняются на противоположные (шаги 11, 12). На шаге 13 проверяется условие (12). На шаге 14 проверяется условие (13). На шаге 15 проверяется условие (14). На

шаге 16 коэффициенты нового случайного неравенства добавляются в матрицу A . На шаге 17 свободный член нового случайного неравенства добавляется в столбец B . На шаге 18 счетчик неравенств увеличивается на единицу. На шаге 19 проверяется, достигло ли количество неравенств требуемого числа, и, при необходимости, осуществляется переход к следующей итерации. На шаге 20 происходит вывод результатов. Шаг 21 завершает работу алгоритма.

3. Параллельный алгоритм генерации случайных задач ЛП

Неявные циклы с возвратом на метку 8, возникающие на шагах 13–15 алгоритма 1, могут потребовать значительных временных затрат. Так, при генерации задачи ЛП, представленной на рис. 1, возвраты на метку 8 осуществлялись 112581 раз с шага 13, 32771 раз с шага 14, и 726 раз с шага 15. В силу этого генерация большой случайной задачи ЛП на обычном персональном компьютере может длиться несколько суток. Поэтому нами была разработана параллельная версия генератора задач FRaGenLP для кластерных вычислительных систем. Эта версия представлена в виде алгоритма 2. Данный алгоритм разработан в соответствии с моделью параллельных вычислений BSF [15], использующей парадигму «мастер-рабочий» [16]. В соответствии с этой моделью узел-мастер является центром управления и коммуникации. Все узлы-рабочие выполняют один и тот же код, но над разными данными.

Сначала рассмотрим *шаги, выполняемые узлом-мастером*. На шаге M1 мастер считывает параметры задачи. На шаге M2 счетчику случайных неравенств присваивается значение 0. В параллельной версии опорные и случайные неравенства хранятся в разных списках. На шагах M3 и M4 мастер создает пустые списки A_S и B_S для коэффициентов и свободных членов опорных неравенств. На шаге M5 в эти списки добавляются опорные неравенства (1). На шаге M6 генерируются коэффициенты целевой функции в соответствии с формулой (3). На шагах M7 и M8 создаются пустые списки A_R и B_R для коэффициентов и свободных членов случайных неравенств. На шаге M9 проверяется количество случайных неравенств d : если $d \leq 0$, то происходит вывод только опорных неравенств (шаг M10), после чего мастер заканчивает свою работу (переход с шага M11 на шаг M37). Если $d > 0$, то мастер продолжает свою работу. На шаге M19 мастер получает по одному случайному неравенству от всех рабочих. Рабочие обеспечивают для своих неравенств выполнение свойств (11)–(13) и проверяют их похожесть на все опорные неравенства. Мастер в цикле M20–M33 проверяет все поступившие случайные неравенства на похожесть с ранее включенными в списки A_R и B_R случайными неравенствами (вложенный цикл M22–M27). Если обнаруживается похожее случайное неравенство (шаги M23–M26), то новое неравенство игнорируется, и осуществляется переход к проверке следующего поступившего неравенства (шаг M28). Если похожих неравенств не нашлось, новое случайное неравенство добавляется мастером в списки A_R , B_R (шаги M29, M30), и счетчик случайных неравенств увеличивается на единицу (шаг M31). Если количество неравенств достигло заданного значения (шаг M32), то происходит выход из внешнего цикла, в противном случае проверки продолжают. После завершения проверок и добавлений поступивших случайных неравенств мастер рассылает всем рабочим общее количество случайных неравенств, включенных в систему на данный момент (шаг M34). Если это количество не достигло требуемого, то происходит переход к следующей глобальной итерации (шаг M35), в противном случае мастер выводит результаты и завершает свою работу (шаги M36, M37).

Теперь рассмотрим *шаги, выполняемые l -м рабочим*. На шаге W1 рабочий считывает параметры задачи. На шаге W2 проверяется количество случайных неравенств d : если $d \leq 0$, то рабочий немедленно заканчивает свою работу, в противном случае создаются пустые списки A_S и B_S для коэффициентов и свободных членов опорных неравенств (шаги W3, W4). На шаге W5 в эти списки добавляются опорные неравенства (1). На шагах W6–W9 рабочий генерирует новое случайное неравенство. На шаге W10 проверяется условие (11). Если это условие не верно, то знаки коэффициентов и свободного члена сгенерированного неравенства меняются на противоположные (шаги W11, W12). Шаги W13–W15 проверяют условия (12), (13). Шаги W16–W18

Алгоритм 2. Параллельный алгоритм генерации случайной задачи ЛП

Мастер	l -й рабочий ($l=1, \dots, L$)
M1: input $n, d, \alpha, \theta, \rho, S_{min}, L_{max}, a_{max}, b_{max}$	W1: input $n, d, \alpha, \theta, \rho, S_{min}, L_{max}, a_{max}, b_{max}$
M2: $k := 0$	W2: if $d \leq 0$ goto W37
M3: $A_S := []$	W3: $A_S := []$
M4: $B_S := []$	W4: $B_S := []$
M5: $AddSupport(A_S, B_S)$	W5: $AddSupport(A_S, B_S)$
M6: for $j = n \dots 1$ do $c_j := \theta \cdot j$	W6: for $j = 1 \dots n$ do begin
M7: $A_R := []$	W7: $a_j^{(l)} := rsign() \cdot rand(0, a_{max})$
M8: $B_R := []$	W8: end
M9: if $d > 0$ goto M19	W9: $b^{(l)} := rsign() \cdot rand(0, b_{max})$
M10: output A_S, B_S, c	W10: if $\langle a^{(l)}, h_{cntr} \rangle \leq b^{(l)}$ goto W13
M11: goto M37	W11: for $j = 1 \dots n$ do $a_j := -a_j$
M12:	W12: $b^{(l)} := -b^{(l)}$
M13:	W13: if $dist(a^{(l)}, b^{(l)}, h_{cntr}) < \rho$ goto W6
M14:	W14: if $dist(a^{(l)}, b^{(l)}, h_{cntr}) > \theta$ goto W6
M15:	W15: if $f(\pi(h_{cntr}, a^{(l)}, b^{(l)})) > f(h_{cntr})$ goto W6
M16:	W16: for all $(\bar{a}, \bar{b}) \in (A_S, B_S)$ do begin
M17:	W17: if $like(a^{(l)}, b^{(l)}, \bar{a}, \bar{b})$ goto W6
M18:	W18: end
M19: $RecvFromAllWorkers(a^{(1)}, b^{(1)}, \dots, a^{(L)}, b^{(L)})$	W19: $SendToMaster(a^{(l)}, b^{(l)})$
M20: for $l = 1 \dots L$ do begin	W20:
M21: $is_like := false$	W21:
M22: for all $(\bar{a}, \bar{b}) \in (A_R, B_R)$ do begin	W22:
M23: if $like(a^{(l)}, b^{(l)}, \bar{a}, \bar{b})$ then begin	W23:
M24: $is_like := true$	W24:
M25: goto M28	W25:
M26: end	W26:
M27: end	W27:
M28: if is_like continue	W28:
M29: $A_R := A_R \# a^{(l)}$	W29:
M30: $B_R := B_R \# b^{(l)}$	W30:
M31: $k := k + 1$	W31:
M32: if $k = d$ goto M34	W32:
M33: end	W33:
M34: $SendToAllWorkers(k)$	W34: $RecvFromMaster(k)$
M35: if $k < d$ goto M19	W35: if $k < d$ goto W6
M36: output A_S, B_S, A_R, B_R, c	W36:
M37: stop	W37: stop

проверяют похожесть сгенерированного неравенства на опорные неравенства с использованием формулы (14). Если хотя бы одно из этих условий не выполняется, осуществляется переход на шаг W6 для повторной генерации нового случайного неравенства. Если все указанные условия выполнены, рабочий пересылает мастеру построенное случайное неравенство (шаг W19). После этого рабочий ждет, когда мастер пришлет ему новое значение счетчика случайных неравенств k (шаг W34). Если k меньше заданного количества случайных неравенств d , то происходит переход на шаг W6 для генерации следующего случайного неравенства (шаг W35), в противном случае рабочий завершает свою работу (шаг W37).

4. Программная реализация и вычислительные эксперименты

Параллельный алгоритм 2 был нами реализован на языке C++ с использованием параллельного BSF-каркаса [17,18]. BSF-каркас базируется на модели параллельных вычислений BSF [15] и инкапсулирует в проблемно независимой части своего кода все аспекты, связанные с

Табл. 1. Характеристики кластера «Торнадо ЮУрГУ».

Количество процессорных узлов	480
Процессоры	Intel Xeon X5680 (6 cores 3.33 GHz)
Количество процессоров в узле	2
Оперативная память узла	24 GB DDR3
Соединительная сеть	InfniBand QDR (40 Gbit/s)
Операционная система	Linux CentOS

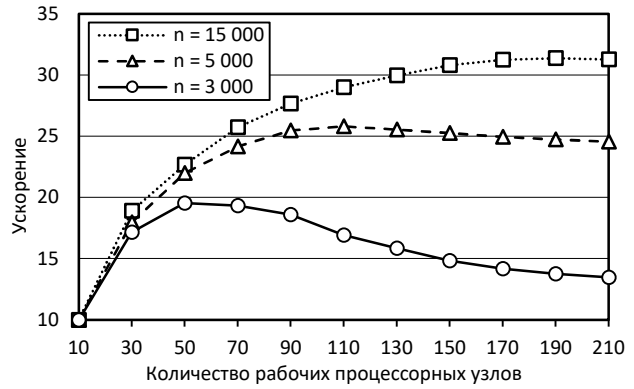


Рис. 2. Графики ускорения алгоритма FRaGenLP для различных размерностей.

распараллеливанием программы с помощью библиотеки MPI [19]. BSF-каркас требует преобразования алгоритма в форму, работающую со списками с использованием функций высшего порядка *Map* и *Reduce*, определяемых формализмом Бирда-Миртенса (Bird–Meertens formalism) [20]. Это преобразование было сделано следующим образом. Длина списков *Map* и *Reduce* устанавливалась равной количеству рабочих MPI-процессов. Элементы списка *Map* определялись как пустые структуры:

struct PT_bsf_mapElem_T {}.

Каждый элемент списка *Reduce* хранил коэффициенты и свободный член одного случайного неравенства $\langle a, x \rangle \leq b$:

struct PT_bsf_reduceElem_T {float a[n]; float b}.

Каждый рабочий MPI-процесс генерировал по одному случайному неравенству с помощью функции *PC_bsf_MapF*, которая включала в себя шаги W6–W18 алгоритма 2. Неравенство, удовлетворяющее всем условиям, записывалось рабочим в локальный список *Reduce*, состоящий из одного элемента. Мастер получал от каждого рабочего сгенерированные элементы и помещал их в свой список *Reduce* (это выполнялось проблемно-независимой частью BSF-каркаса). Полученные элементы вручную проверялись на похожесть с ранее добавленными случайными неравенствами и, при отсутствии конфликтов, добавлялись в списки A_R, B_R . Указанная обработка (шаги M20–M33) была нами реализована в предопределенной функции *PC_bsf_ProcessResults*. Исходные коды параллельной программы FRaGenLP свободно доступны в сети Интернет по адресу <https://github.com/leonid-sokolinsky/BSF-LPP-Generator>.

С использованием указанной программы нами были проведены масштабные вычислительные эксперименты на вычислительном кластере «Торнадо ЮУрГУ» [21], характеристики которого приведены в табл. 1. Вычисления проводились для размерности n , равной 3 000, 5 500 и 15 000. Количество неравенств, соответственно, составляло 6 301, 10 001 и 31 501. Из них случайных: 300, 500, 1500 соответственно. Во всех экспериментах были установлены следующие значения параметров, введенных в разделе 2: длина ребра ограничивающего гиперкуба $\alpha = 200$, радиус большой гипертсферы $\theta = 100$, радиус малой гипертсферы $\rho = 50$, верхняя граница «почти параллельности» для гиперплоскостей $L_{max} = 0.35$, минимальная допустимая близость для гиперплоскостей $S_{min} = 100$, максимальное допустимое абсолютное значение при генерации коэффициентов случайных неравенств $a_{max} = 1000$, максимальное допустимое абсолютное значение при генерации правых частей случайных неравенств $b_{max} = 10000$. Результаты экспериментов представлены на рис. 2. Время генерации случайной задачи ЛП с 31501 ограничениями на конфигурации из одного узла-мастера и одного узла-рабочего составило 12 минут. На конфигурации из одного узла-мастера и 170 узлов-рабочих генерация такой же задачи заняла 22 секунды. Анализ результатов показывает, что граница масштабируемости предложенного алгоритма существенно зависит от размерности задачи (под границей масштабируемости здесь понимается максимум кривой ускорения). При $n = 3000$ граница масштабируемости составила приблизительно 50 процессорных узлов. Для задачи размерности $n = 5000$ эта граница увеличилась до 110 узлов, а на

задаче размерности $n = 15000$ она достигла 200 узлов. Дальнейшее увеличение размерности задачи приводило к нехватке оперативной памяти на процессорных узлах. Следует отметить, что граница масштабируемости предложенного алгоритма также существенно зависит от количества случайных неравенств. Увеличение этого числа в 10 раз приводило к двукратному уменьшению границы масштабируемости. Это объясняется тем, что при увеличении количества узлов-рабочих значительно возрастала доля последовательных вычислений, выполняемых узлом-мастером на шагах M20–M27, в течение которых узлы-рабочие простаивали.

5. Заключение

В данной работе представлен параллельный алгоритм FRaGenLP для генерации случайных совместных ограниченных задач ЛП на кластерных вычислительных системах. Генерируемые системы ограничений наряду со случайными неравенствами включают в себя стандартный набор неравенств, называемых опорными. Опорные неравенства гарантируют ограниченность допустимой области задачи ЛП. С геометрической точки зрения допустимая область опорных неравенств представляет собой гиперкуб, прилежащий к осям координат, у которого срезана дальняя от центра координат вершина. Целевая функция задается определенным образом, так, чтобы ее коэффициенты монотонно убывали. Коэффициенты и свободные члены случайных неравенств генерируются с помощью датчика случайных чисел. Если допустимая область сгенерированного случайного неравенства не включает в себя центр ограничивающего гиперкуба, то знак неравенства меняется на противоположный. Однако не всякое случайное неравенство включается в систему. Отклоняются случайные неравенства, которые не могут повлиять на решение задачи ЛП при заданной целевой функции. Также отклоняются все неравенства, для которых ограничивающая гиперплоскость пересекает малую гиперсферу, расположенную в центре ограничивающего гиперкуба (это гарантирует совместность). Кроме того, отклоняется всякое случайное неравенство, «похожее» на хотя бы одно неравенство, уже добавленное в систему (включая опорные). Для определения «похожести» неравенств введены две формальные метрики: почти параллельности и близости соответствующих им гиперплоскостей. Параллельный алгоритм спроектирован на базе модели параллельных вычислений BSF, использующей парадигму «мастер-рабочий». В соответствии с этой моделью узел-мастер является центром управления и коммуникации. Все узлы-рабочие выполняют один и тот же код, но над разными данными. Параллельная реализация выполнена на языке C++ с использованием параллельного BSF-каркаса, инкапсулирующего в проблемно независимой части своего кода все аспекты, связанные с распараллеливанием программы с помощью библиотеки MPI. Исходные коды разработанной параллельной программы свободно доступны в сети Интернет по адресу <https://github.com/leonid-sokolinsky/BSF-LPP-Generator>. С использованием этой реализации были проведены масштабные вычислительные эксперименты на кластерной вычислительной системе. Проведенные эксперименты показали, что алгоритм FRaGenLP демонстрирует для размерности 15 000 хорошую масштабируемость вплоть до 200 процессорных узлов. Случайная задача ЛП, включающая в себя 31 501 неравенство, была сгенерирована на конфигурации из 171 процессорного узла за 22 секунды. На одном процессорном узле для этого потребовалось 12 минут. Разработанную программу предполагается использовать для генерации 70 000 образцов для обучения искусственной нейронной сети, способной быстро решать большие задачи ЛП.

Литература

1. Jagadish H. V. et al. Big data and its technical challenges // Communications of the ACM. 2014. Vol. 57, no. 7. P. 86–94. DOI:10.1145/2611567.
2. Hartung T. Making Big Sense From Big Data // Frontiers in Big Data. 2018. Vol. 1. P. 5. 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. Короткие статьи и описания плакатов. Челябинск: Издательский центр ЮУрГУ, 2017. С. 471–484.

4. Соколинский Л.Б., Соколинская И.М. Исследование масштабируемости апекс-метода для решения сверхбольших задач линейного программирования на кластерных вычислительных системах // Суперкомпьютерные дни в России: Труды международной конференции. 21-22 сентября 2020 г. Москва: МАКС Пресс, 2020. С. 49–59.
5. 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. Zaroliagis C., Pantziou G., Kontogiannis S. Cham: Springer, 2015. P. 281–307. DOI:10.1007/978-3-319-24024-4_17.
6. Huangfu Q., Hall J.A.J. Parallelizing the dual revised simplex method // Mathematical Programming Computation. Springer Verlag, 2018. Vol. 10, no. 1. P. 119–142. DOI:10.1007/s12532-017-0130-5.
7. Tar P., Stágel B., Maros I. Parallel search paths for the simplex algorithm // Central European Journal of Operations Research. Springer Verlag, 2017. Vol. 25, no. 4. P. 967–984. DOI:10.1007/s10100-016-0452-9.
8. Yang L., Li T., Li J. Parallel predictor-corrector interior-point algorithm of structured optimization problems // 3rd International Conference on Genetic and Evolutionary Computing, WGEC 2009. 2009. P. 256–259. DOI:10.1109/WGEC.2009.68.
9. Gay D.M. Electronic mail distribution of linear programming test problems // Mathematical Programming Society COAL Bulletin. 1985. no. 13. P. 10–12.
10. Charnes A. et al. On Generation of Test Problems for Linear Programming Codes // Communications of the ACM. 1974. Vol. 17, no. 10. P. 583–586. DOI:10.1145/355620.361173.
11. Arthur J.L., Friendewey J.O. GENGUB: A generator for linear programs with generalized upper bound constraints // Computers and Operations Research. Pergamon, 1993. Vol. 20, no. 6. P. 565–573. DOI:10.1016/0305-0548(93)90112-V.
12. Castillo E., Pruneda R.E., Esquivel Mó. Automatic generation of linear programming problems for computer aided instruction // International Journal of Mathematical Education in Science and Technology. Taylor & Francis Group, 2001. Vol. 32, no. 2. P. 209–232. DOI:10.1080/00207390010010845.
13. Dhiiflaoui M. et al. Certifying and Repairing Solutions to Large LPs How Good are LP-solvers? // SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms. USA: Society for Industrial and Applied Mathematics, 2003. P. 255–256.
14. Desmos Graphing Calculator [Electronic resource]. URL: <https://www.desmos.com/calculator> (accessed: 09.03.2021).
15. 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.
16. Sahni S., Vairaktarakis G. The master-slave paradigm in parallel computer and industrial settings // Journal of Global Optimization. 1996. Vol. 9, no. 3–4. P. 357–377. DOI:10.1007/BF00121679.
17. Sokolinsky L.B. BSF-skeleton. User manual: arXiv:2008.12256 [cs.DC]. 2020. 22 p.
18. Sokolinsky L.B. BSF-skeleton [Electronic resource]. 2019. URL: <https://github.com/leonid-sokolinsky/BSF-skeleton> (accessed: 09.03.2021).
19. Gropp W. MPI 3 and Beyond: Why MPI Is Successful and What Challenges It Faces // Recent Advances in the Message Passing Interface. EuroMPI 2012. Lecture Notes in Computer Science, vol. 7490 / ed. Träff J.L., Benkner S., Dongarra J.J. Berlin, Heidelberg: Springer, 2012. P. 1–9. DOI:10.1007/978-3-642-33518-1_1.
20. Bird R.S. Lectures on Constructive Functional Programming // Constructive Methods in Computing Science. NATO ASI Series F: Computer and Systems Sciences, vol. 55 / ed. Broy M. Berlin, Heidelberg: Springer, 1988. P. 151–216.
21. Kostenetskiy P.S., Safonov A.Y. SUSU Supercomputer Resources // Proceedings of the 10th Annual International Scientific Conference on Parallel Computing Technologies (PCT 2016). CEUR Workshop Proceedings. Vol. 1576. 2016. P. 561–573.