ISSN 2305-9052 (Print)
ISSN 2410-7034 (Online)

ВЕСТНИК ЮЖНО-УРАЛЬСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

BULLETIN OF THE SOUTH URAL STATE UNIVERSITY

СЕРИЯ

ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА И ИНФОРМАТИКА

2024, том 13, № 3

SERIES

COMPUTATIONAL MATHEMATICS AND SOFTWARE ENGINEERING

2024, volume 13, no. 3

1943

ВЕСТНИК

ЮЖНО-УРАЛЬСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

2024
Т. 13, № 3

ISSN 2305-9052 (Print)
ISSN 2410-7034 (Online)

СЕРИЯ

«ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА И ИНФОРМАТИКА»

Решением ВАК включен в Перечень научных изданий, в которых должны быть опубликованы результаты диссертаций на соискание ученых степеней кандидата и доктора наук

Учредитель — Федеральное государственное автономное образовательное учреждение высшего образования «Южно-Уральский государственный университет (национальный исследовательский университет)»

Тематика журнала:

- Вычислительная математика и численные методы
- Математическое программирование
- Распознавание образов
- Вычислительные методы линейной алгебры
- Решение обратных и некорректно поставленных задач
- Доказательные вычисления
- Численное решение дифференциальных и интегральных уравнений
- Исследование операций
- Теория игр
- Теория аппроксимации
- Информатика
- Искусственный интеллект и машинное обучение
- Системное программирование
- Перспективные многопроцессорные архитектуры
- Облачные вычисления
- Технология программирования
- Машинная графика
- Интернет-технологии
- Системы электронного обучения
- Технологии обработки баз данных и знаний
- Интеллектуальный анализ данных

BULLETIN
OF THE SOUTH URAL STATE UNIVERSITY
SERIES
"COMPUTATIONAL MATHEMATICS AND SOFTWARE ENGINEERING"

2024
Vol. 13, no. 3

ISSN 2305-9052 (Print)
ISSN 2410-7034 (Online)

Vestnik Yuzhno-Ural'skogo Gosudarstvennogo Universiteta.
Seriya "Vychislitel'naya Matematika i Informatika"

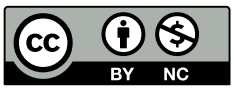
South Ural State University

The scope of the journal:

- Numerical analysis and methods
- Mathematical optimization
- Pattern recognition
- Numerical methods of linear algebra
- Reverse and ill-posed problems solution
- Computer-assisted proofs
- Numerical solutions of differential and integral equations
- Operations research
- Game theory
- Approximation theory
- Computer science
- Artificial intelligence and machine learning
- System software
- Advanced multiprocessor architectures
- Cloud computing
- Software engineering
- Computer graphics
- Internet technologies
- E-learning
- Database processing
- Data mining

Содержание

Contents



УДК 519.852, 004.021, 004.032.24 **DOI: 10.14529/cmse240301**

ЧИСЛЕННАЯ РЕАЛИЗАЦИЯ МЕТОДА ПОВЕРХНОСТНОГО ДВИЖЕНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ

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

Южно-Уральский государственный университет

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

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

Поступила в редакцию: 10.06.2024

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

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

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Соколинский Л.Б., Ольховский Н.А., Соколинская И.М. Численная реализация метода поверхностного движения для решения задач линейного программирования // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2024. Т. 13, № 3. С. 5–31. DOI: 10.14529/cmse240301.

Введение

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

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

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

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

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

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

1. Теоретический базис

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

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

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

[1]Допустимый многогранник — область допустимых решений задачи линейного программирования.

неравенство $A\boldsymbol{x} \leqslant \boldsymbol{b}$ в форме

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

Линейная целевая функция задачи (1) имеет вид

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

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

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

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

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

Определим допустимый многогранник

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

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

Дадим определение рецессивного полупространства [10].

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

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

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

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

Определим

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

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

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

Положим

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

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

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

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

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

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

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

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

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

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

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

Следующее утверждение дает способ вычисления оптимального пути на гиперплоскости.

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

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

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

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

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

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

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

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

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

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

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

$\|\tilde{\boldsymbol{w}} - \boldsymbol{u}\| = \|\boldsymbol{w} - \boldsymbol{u}\|$ и $\tilde{\boldsymbol{w}} \neq \boldsymbol{w}$ (см. рис. 1). Здесь и далее $\|\cdot\|$ обозначает евклидову норму. Вычислим $\langle \boldsymbol{c}, \boldsymbol{w} \rangle$. В соответствии с определением 2 ортогональная проекция $\boldsymbol{\pi}_H(\boldsymbol{v})$ точки

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

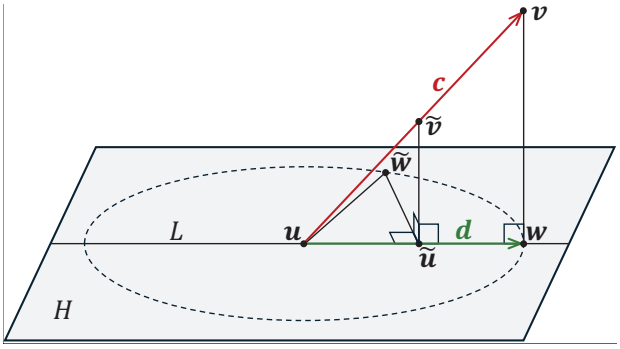


Рис. 1. Иллюстрация к доказательству предложения 1.
Пунктиром обозначена гиперокружность с радиусом $\|\boldsymbol{w}-\boldsymbol{u}\|$

$\boldsymbol{v}$ на гиперплоскость H, задаваемую формулой (9), вычисляется следующим образом:

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

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

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

Используя (14), находим

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

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

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

Отсюда следует

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

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

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

такое, что

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

Положим

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

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

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

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

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

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

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

Очевидно,

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

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

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

Отсюда получаем

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

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

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

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

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

что противоречит (13). □

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

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

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

2. Операция псевдопроектирования на линейное многообразие

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

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

Обозначим k_L — размерность линейного многообразия L. При $k_L < n-1$ многообразие L не является гиперплоскостью, и для определения вектора движения $\boldsymbol{d}$ по этому многообразию в направлении максимального увеличения целевой функции нельзя применить формулу (22), так как такое линейное многообразие невозможно задать одним линейным уравнением в пространстве $\mathbb{R}^n$. Однако мы можем найти указанный вектор $\boldsymbol{d}$ с помощью операции псевдопроектирования [10]. Определим проекционное отображение $\boldsymbol{\varphi}(\cdot)$:

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

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

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

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

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

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

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

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

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

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

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

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

$$P = L^{\perp}. \tag{26}$$

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

$$\forall \boldsymbol{v} \in P : \boldsymbol{\pi}_{H_{i'}}(\boldsymbol{v}) \in P.$$

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

$$\boldsymbol{p} = \boldsymbol{\pi}_{H_{i'}}(\boldsymbol{v}). \tag{27}$$

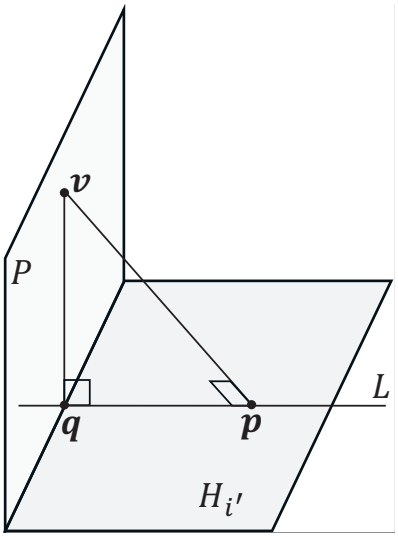


Рис. 2. Иллюстрация к лемме 1

Без ограничения общности мы можем считать, что $\boldsymbol{p} \in L$ (см. рис 2). Предположим, что точка $\boldsymbol{p}$ не принадлежит линейному многообразию P. Обозначим $\boldsymbol{q}$ — точка пересечения линейного многообразия L с его ортогональным дополнением P:

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

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

$$\boldsymbol{q} = \boldsymbol{\pi}_P(\boldsymbol{p}). \tag{28}$$

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

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

Лемма доказана. □

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

Утверждение 2. Пусть выполняются следующие условия:

$$\mathcal{J} \subseteq \{1, \ldots, m\}, \tag{29}$$

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

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

$$\boldsymbol{\rho}_L(\boldsymbol{x}) = \boldsymbol{\pi}_L(\boldsymbol{x}),$$

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

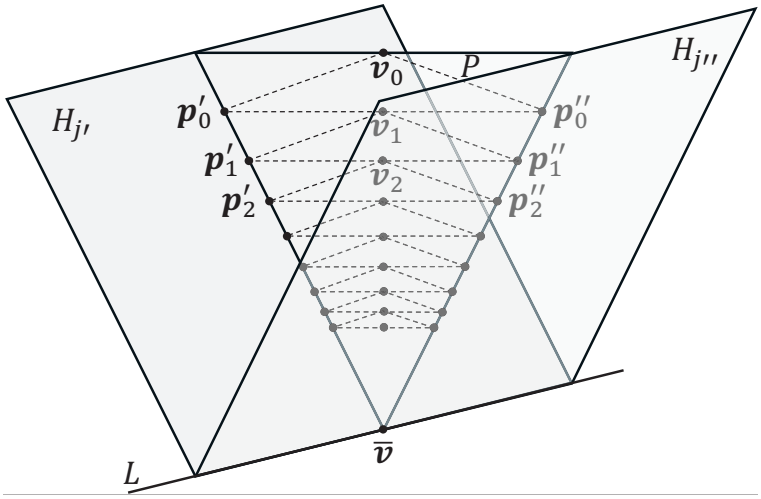


Рис. 3. Иллюстрация к доказательству утверждения 2 при $n = 3$

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

$$\boldsymbol{v}_0 \in P = L^{\perp}. \tag{31}$$

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

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

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

$$\boldsymbol{p}_0 = \boldsymbol{\pi}_{H_j}(\boldsymbol{v}_0).$$

В соответствии с леммой 1 имеем

$$\boldsymbol{\pi}_{H_j}(\boldsymbol{v}_0) \in P.$$

Отсюда и из (24) следует, что

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

Это означает, что последовательность

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

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

$$\forall \boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{\rho}_L(\boldsymbol{x}) = \boldsymbol{\pi}_L(\boldsymbol{x}).$$

Утверждение доказано. □

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

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

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

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

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

3. Алгоритм движения по граням AlFaMove

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

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

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

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

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

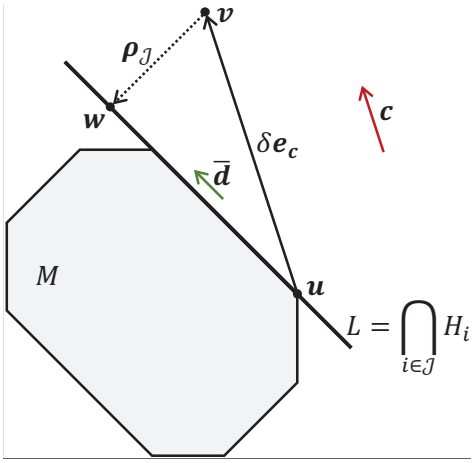


Рис. 4. Вычисление вектора направления движения $\bar{\boldsymbol{d}}$ по грани линейного многообразия L

левой функции f присваивается бесконечно малое число[3]. На шаге 10 строится единичный вектор $\boldsymbol{e_c}$, сонаправленный с вектором $\boldsymbol{c}$. На шаге 11 строится точка $\boldsymbol{v}$ путем прибавления вектора $\delta \boldsymbol{e_c}$ к вектору $\boldsymbol{u}$ (см. рис. 4). Здесь δ — «большой» положительный параметр: чем

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

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

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

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

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

Алгоритм 3 Алгоритм движения по граням AlFaMove

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

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

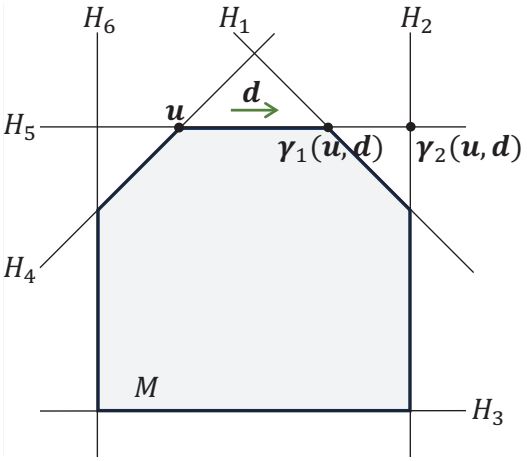


Рис. 5. Действие функции $\boldsymbol{\mu}$:
$\boldsymbol{\mu}(\boldsymbol{u},\boldsymbol{d}) = \boldsymbol{\gamma}_1(\boldsymbol{u},\boldsymbol{d})$

лизованная в алгоритме 2. Предполагается, что $\boldsymbol{d}_0 \neq \boldsymbol{0}$[4]. Это условие контролируется на шаге 4. На шаге 5 счетчик итераций k устанавливается в значение ноль. Шаг 6 открывает цикл **repeat**, выполняющий движение по граням допустимого многогранника до тех пор, пока очередной вектор движения $\boldsymbol{d}_k$ не окажется нулевым вектором. Это означает, что последнее найденное приближение $\boldsymbol{u}_k$ является решением задачи ЛП (1). Шаг 7 в теле цикла вычисляет следующее приближение $\boldsymbol{u}_{k+1}$ с помощью вектор-функции $\boldsymbol{\mu}$ (определение см. ниже). Шаг 8 вычисляет вектор движения $\boldsymbol{d}_{k+1}$ для вновь найденного приближения $\boldsymbol{u}_{k+1}$. Шаг 9 увеличивает счетчик итераций k на единицу. Если последний вектор движения равен нулевому вектору, то на шаге 10 происходит выход из цикла **repeat**/**until**, после чего на шаге 11 выводятся координаты точки $\boldsymbol{u}_k$ в качестве решения задачи ЛП (1). Шаг 12 завершает работу алгоритма AlFaMove.

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

$$\mathcal{Q} = \{i \in \{1,\ldots,m\} \mid \langle \boldsymbol{a}_i, \boldsymbol{u}\rangle < b_i \wedge \langle \boldsymbol{a}_i, \boldsymbol{d}\rangle > 0\}. \tag{32}$$

Тогда

$$\boldsymbol{\mu}(\boldsymbol{u},\boldsymbol{d}) = \arg\min_{i\in\mathcal{Q}} \{\|\boldsymbol{u}-\boldsymbol{x}\| \mid \boldsymbol{x} = \boldsymbol{\gamma}_i(\boldsymbol{u},\boldsymbol{d})\}. \tag{33}$$

Здесь $\boldsymbol{\gamma}_i(\boldsymbol{u},\boldsymbol{d})$ обозначает вектор-функцию, вычисляющую косоугольную проекцию точки $\boldsymbol{u}$ на гиперплоскость H_i по направлению $\boldsymbol{d}$:

$$\boldsymbol{\gamma}_i(\boldsymbol{u},\boldsymbol{d}) = \boldsymbol{u} - \frac{\langle \boldsymbol{a}_i, \boldsymbol{u}\rangle - b_i}{\langle \boldsymbol{a}_i, \boldsymbol{d}\rangle}\boldsymbol{d}.$$

Действие функции $\boldsymbol{\mu}$ проиллюстрировано на рис. 5. В соответствии с рисунком гиперплоскости H_4 и H_5 не удовлетворяют неравенству $\langle \boldsymbol{a}_i, \boldsymbol{u}\rangle < b_i$ в (32). Гиперплоскости H_3 и H_6 не удовлетворяют неравенству $\langle \boldsymbol{a}_i, \boldsymbol{d}\rangle > 0$ в (32). Таким образом, $\mathcal{Q} = \{1,2\}$. Так как $\|\boldsymbol{u} - \boldsymbol{\gamma}_1(\boldsymbol{u},\boldsymbol{d})\| < \|\boldsymbol{u} - \boldsymbol{\gamma}_2(\boldsymbol{u},\boldsymbol{d})\|$, то $\boldsymbol{\mu}(\boldsymbol{u},\boldsymbol{d}) = \boldsymbol{\gamma}_1(\boldsymbol{u},\boldsymbol{d})$.

[4]Равенство вектора $\boldsymbol{d}_0$ нулевому вектору означает, что точка $\boldsymbol{u}_0$ является решением задачи ЛП (1).

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

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

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

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

Согласно утверждению 2 отсюда следует

$$\boldsymbol{d}_{AlFaMove} = \boldsymbol{\pi}_{\mathcal{J}}(\boldsymbol{v}) - \boldsymbol{u}, \tag{34}$$

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

4. Параллельная версия алгоритма AlFaMove

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

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

Алгоритм 4 Параллельная версия алгоритма AlFaMove

мастер	l**-тый рабочий** $(l = 0, \ldots, L-1)$
input $n, m, A, \boldsymbol{b}, \boldsymbol{u}_0$	**input** $n, m, A, \boldsymbol{b}, \boldsymbol{c}$
$k := 0$	
repeat	**repeat**
Broadcast $\boldsymbol{u}_k$	**RecvFromMaster** $\boldsymbol{u}_k$
	$\mathcal{U} := [\,]$
	for $i = 1 \ldots m$ **do**
	if $\langle \boldsymbol{a}_i, \boldsymbol{u}_k \rangle = b_i$ **then**
	$\mathcal{U} := \mathcal{U} +\!\!+ [i]$
	end if
	end for
	$K := 2^{\lvert\mathcal{U}\rvert} - 1$
	$L :=$ NumberOfWorkers
	$\mathcal{L}_{map(l)} := [lK/L, \ldots, (l+1)K/L - 1]$
	$\mathcal{L}_{reduce(l)} := Map(\mathrm{F}_{\boldsymbol{u}_k}, \mathcal{L}_{map(l)})$
	$(\boldsymbol{d}_l, f_l) := Reduce(\oplus, \mathcal{L}_{reduce(l)})$
Gather $\mathcal{L}_{reduce}$	**SendToMaster** $(\boldsymbol{d}_l, f_l)$
$(\boldsymbol{d}_k, f_k) := Reduce(\oplus, \mathcal{L}_{reduce})$	
if $\boldsymbol{d}_k = \boldsymbol{0}$ **then**	
$exit :=$ **true**	
else	
$\boldsymbol{u}_{k+1} := \boldsymbol{\mu}(\boldsymbol{u}_k, \boldsymbol{d}_k)$	
$k := k + 1$	
$exit :=$ **false**	
end if	
Broadcast $exit$	**RecvFromMaster** $exit$
until $exit$	**until** $exit$
output $\boldsymbol{u}_k, f_k$	
stop	**stop**

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

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

определенная следующим образом:

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

где

$$\boldsymbol{w} = \boldsymbol{\rho}_{\sigma(j)}(\boldsymbol{u} + \delta \boldsymbol{c}/\|\boldsymbol{c}\|) \tag{36}$$

и

$$\boldsymbol{e_d} = \frac{\boldsymbol{w} - \boldsymbol{u}}{\|\boldsymbol{w} - \boldsymbol{u}\|}.$$

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

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

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

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

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

Определим бинарную ассоциативную операцию

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

являющуюся первым параметром функции высшего порядка *Reduce*:

$$\left(\boldsymbol{d}', f'\right) \oplus \left(\boldsymbol{d}'', f''\right) = \begin{cases} (\boldsymbol{d}', f')\,, & \text{если} f' \geqslant f''; \\ (\boldsymbol{d}'', f'')\,, & \text{если} f' < f''. \end{cases} \tag{37}$$

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

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

где согласно (37)

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

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

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

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

а их конкатенация дает полный список:

$$\mathcal{L}_{map} = \mathcal{L}_{map(0)} +\!\!+ \ldots +\!\!+ \mathcal{L}_{map(L-1)}. \tag{39}$$

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

5. Вычислительные эксперименты

Мы реализовали параллельную версию алгоритма AlFaMove на языке C++ с использованием программного BSF-каркаса [15], базирующегося на модели параллельных вычислений BSF [14]. BSF-каркас инкапсулирует все аспекты, связанные с распараллеливанием программы на основе библиотеки MPI. Исходные коды параллельной реализации алгоритма AlFaMove решения задач ЛП свободно доступны в репозитории GitHub по адресу `https://github.com/leonid-sokolinsky/AlFaMove`. Разработанная программа была протестирована на большом количестве задач ЛП, взятых из различных источников. Все эти задачи в формате MTX [16] доступны по адресу `https://github.com/leonid-sokolinsky/Set-of-LP-Problems`. В качестве тестов мы также использовали искусственные задачи, полученные с помощью генератора случайных задач ЛП FRaGenLP [17]. Эти задачи доступны по адресу `https://github.com/leonid-sokolinsky/Set-of-LP-Problems/tree/main/Rnd-LP`. Мы не смогли протестировать реализацию AlFaMove на задачах из репозитория Netlib-LP [18], так как во всех этих задачах количество гиперплоскостей, проходящих через начальную точку $\boldsymbol{u}_0$, превышало число 30, что соответствует количеству возможных комбинаций, равному 1 073 741 824. Массивы таких размеров не допускаются доступными нам компиляторами C++.

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

$$\begin{cases} x_1 & & & & & \leqslant & 200 \\ & x_2 & & & & \leqslant & 200 \\ & & & & & \vdots & \\ & & & x_n & & \leqslant & 200 \\ x_1 & + & x_2 & \cdots + & x_n & \leqslant & 200(n-1)+100 \\ x_1 \geqslant 0, & & x_2 \geqslant 0, & \cdots & , x_n \geqslant 0. & & \end{cases} \tag{40}$$

Градиент целевой функции задается вектором

$$\boldsymbol{c} = (1, 2, \ldots, n)\,. \tag{41}$$

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

В первой серии экспериментов исследовалась зависимость ускорения и эффективности распараллеливания алгоритма AlFaMove от количества используемых процессорных узлов кластера при решении задач «гиперкуб с отсеченной вершиной». Результаты этих экспериментов представлены на рис. 6. В данном контексте ускорение $\alpha(L)$ определялось как отно-

Таблица 1. Характеристики кластера «Торнадо ЮУрГУ»

Параметр	Значение
Количество процессорных узлов	480
Процессоры	Intel Xeon X5680 (6 cores, 3.33 GHz)
Число процессоров на узел	2
Память на узел	24 GB DDR3
Соединительная сеть	InfiniBand QDR (40 Gbit/s)
Операционная система	Linux CentOS

шение времени $T(1)$ решения задачи на конфигурации с узлом-мастером и единственным узлом-рабочим ко времени $T(L)$ решения той же задачи на конфигурации с узлом-мастером и L узлами-рабочими:

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

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

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

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

$$x_1 = 0, \quad \ldots \quad x_{n/2} = 0, \quad x_{n/2+1} = 200, \quad \ldots \quad x_n = 200. \tag{42}$$

Эксперименты продемонстрировали хорошую масштабируемость алгоритма AlFaMove на задачах «гиперкуб с отсеченной вершиной», начиная с размерности $n = 18$. В этом случае алгоритм демонстрировал ускорение, близкое к линейному. На меньших размерностях затраты на обмены и латентность начинают превалировать над вычислительными затратами, что приводит к резкому уменьшению границы масштабируемости алгоритма[5]. Для

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

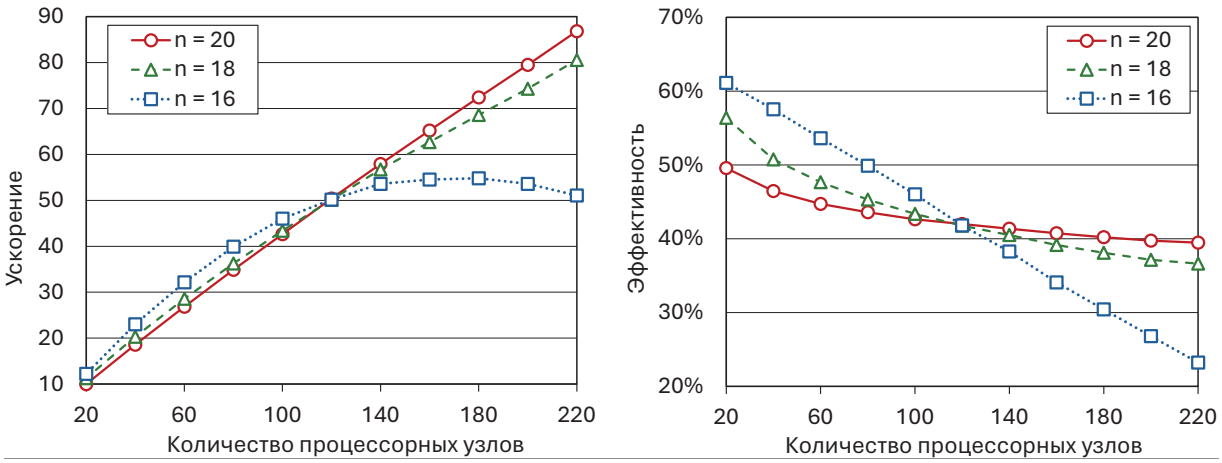


Рис. 6. Ускорение и эффективность распараллеливания алгоритма AlFaMove

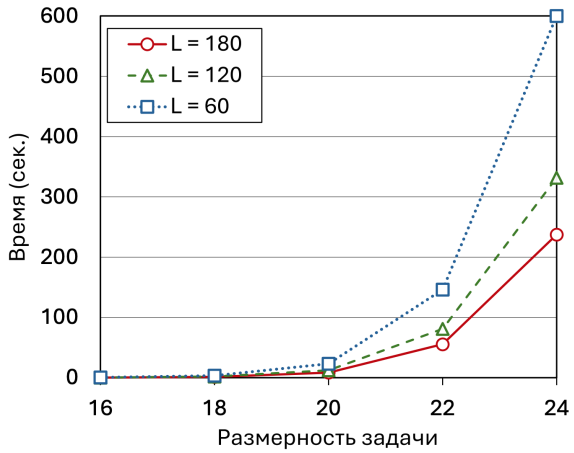


Рис. 7. Зависимость времени работы AlFaMove от размерности задачи на различных многопроцессорных конфигурациях (L — количество рабочих узлов)

$n = 16$ эта граница оказалась равной 180 узлам. Эксперименты также показали, что с увеличением размерности задачи эффективность распараллеливания на малом количестве процессорных узлов (меньше 120) падает. Однако, при большем количестве процессорных узлов наблюдается обратная тенденция. Так, для размерности $n = 16$ эффективность распараллеливания на 20 процессорных узлах составила 61%, после чего снизилась до 23% на 220 узлах. При этом, для $n = 20$ эффективность распараллеливания оказалась равной 50% и 40% соответственно.

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

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

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

Градиент целевой функции задается вектором

$$\boldsymbol{c} = \left(2^{n-1},\, 2^{n-2},\, \ldots\,, 2,\, 1\right).$$

Задача предполагает нахождение максимума целевой функции и имеет единственное решение в точке $(0, \ldots, 0, 5^n)$ со значением целевой функции, равным 5^n. Кле и Минти в статье [20] показали, что в случае, когда начальной вершиной является центр координат, классический симплекс-метод при решении этой задачи посещает все 2^n вершин, делая $2^n - 1$ итераций. Известно, что большинство оптимизационных алгоритмов демонстрируют плохую производительность применительно к кубу Кле—Минти. Мы применили алгоритм AlFaMove для решения кубов Кле—Минти размерности от 5 до 9. Результаты экспериментов, приведенные в табл. 2, показывают, что алгоритм AlFaMove во всех случаях находил решение за $2n - 1$ итераций, в то время, как симплекс-метод совершал $2^n - 1$ итераций.

Таблица 2. Эксперименты с кубами Кле—Минти

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

Относительная погрешность вычислялась по формуле

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

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

Заключение

В статье представлен алгоритм AlFaMove, являющийся численной реализацией метода поверхностного движения для решения задач линейного программирования. Ключевой особенностью этого метода является построение оптимального пути на поверхности допустимого многогранника от начальной точки к решению задачи линейного программирования. Под оптимальным путем понимается путь по поверхности допустимой области в

направлении максимального увеличения значений целевой функции. Практическая значимость предложенного метода состоит в том, что он открывает возможность применения искусственных нейронных сетей прямого распространения для решения нестационарных многомерных задач линейного программирования в режиме реального времени.

Теоретической основой алгоритма AlFaMove является операция построения псевдопроекции на линейные многообразия, которые формируют грани допустимого многогранника. Псевдопроекция реализуется на основе фейеровского процесса и является обобщением понятий ортогональной проекции на линейное многообразие и метрической проекции на выпуклое множество. В случае грани, образуемой гиперплоскостью, псевдопроекция сводится к ортогональной проекции. Доказано, что путь по гиперплоскости, построенный с помощью градиента целевой функции и ортогональной проекции, является оптимальным. Приведен алгоритм проекционного типа для построения псевдопроекции на линейные многообразия меньших размерностей, образуемые пересечением гиперплоскостей. Доказано, что в этом случае псевдопроекция совпадает с ортогональной проекцией. Представлено формализованное описание алгоритма AlFaMove, строящего оптимальный путь на поверхности допустимого многогранника. В основе алгоритма AlFaMove лежит процедура вычисления вектора движения по грани допустимого многогранника из точки текущего приближения в направлении максимального увеличения значения целевой функции. Дано формализованное описание этой процедуры.

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

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

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

Литература

1. Optimization in Large Scale Problems: Industry 4.0 and Society 5.0 Applications / ed. by M. Fathi, M. Khakifirooz, P.M. Pardalos. Cham, Switzerland: Springer, 2019. XI, 340 p. DOI: 10.1007/978-3-030-28565-4.
2. Kopanos G.M., Puigjaner L. Solving Large-Scale Production Scheduling and Planning in the Process Industries. Cham, Switzerland: Springer, 2019. 291 p. DOI: 10.1007/978-3-030-01183-3.
3. Schlenkrich M., Parragh S.N. Solving large scale industrial production scheduling problems with complex constraints: an overview of the state-of-the-art // 4th International Conference on Industry 4.0 and Smart Manufacturing. Procedia Computer Science. Vol. 217 / ed. by F. Longo, M. Affenzeller, A. Padovano, W. Shen. Elsevier, 2023. P. 1028–1037. DOI: 10.1016/J.PROCS.2022.12.301.
4. Соколинская И.М., Соколинский Л.Б. О решении задачи линейного программирования в эпоху больших данных // Параллельные вычислительные технологии (ПаВТ'2017). Короткие статьи и описания плакатов. Челябинск: Издательский центр ЮУрГУ, 2017. С. 471–484. URL: http://omega.sp.susu.ru/pavt2017/short/014.pdf.
5. Branke J. Optimization in Dynamic Environments // Evolutionary Optimization in Dynamic Environments. Genetic Algorithms and Evolutionary Computation, vol. 3. Boston, MA: Springer, 2002. P. 13–29. DOI: 10.1007/978-1-4615-0911-0_2.
6. Dantzig G.B. Linear programming and extensions. Princeton, N.J.: Princeton university press, 1998. 656 p.
7. Зоркальцев В.И., Мокрый И.В. Алгоритмы внутренних точек в линейной оптимизации // Сибирский журнал индустриальной математики. 2018. Т. 21, 1 (73). С. 11–20. DOI: 10.17377/sibjim.2018.21.102.
8. Mamalis B., Pantziou G. Advances in the Parallelization of the Simplex Method // Algorithms, Probability, Networks, and Games. Lecture Notes in Computer Science, vol. 9295 / ed. by C. Zaroliagis, G. Pantziou, S. Kontogiannis. Cham: Springer, 2015. P. 281–307. DOI: 10.1007/978-3-319-24024-4_17.
9. Ольховский Н.А., Соколинский Л.Б. О новом методе линейного программирования // Вычислительные методы и программирование. 2023. Т. 24, № 4. С. 408–429. DOI: 10.26089/NumMet.v24r428.
10. Соколинский Л.Б., Соколинская И.М. О новой версии апекс-метода для решения задач линейного программирования // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2023. Т. 12, № 2. С. 5–46. DOI: 10.14529/cmse230201.
11. Мальцев А.И. Основы линейной алгебры. Москва: Наука. Главная редакция физико-математической литературы, 1970. 402 с.
12. Васин В.В., Ерёмин И.И. Операторы и итерационные процессы фейеровского типа. Теория и приложения. Екатеринбург: УрО РАН, 2005. 211 с.
13. Gould N.I. How good are projection methods for convex feasibility problems? // Computational Optimization and Applications. 2008. Vol. 40, no. 1. P. 1–12. DOI: 10.1007/S10589-007-9073-5.

14. Sokolinsky L.B. BSF: A parallel computation model for scalability estimation of iterative numerical algorithms on cluster computing systems // Journal of Parallel and Distributed Computing. 2021. Vol. 149. P. 193–206. DOI: 10.1016/j.jpdc.2020.12.009.

15. Sokolinsky L.B. BSF-skeleton: A Template for Parallelization of Iterative Numerical Algorithms on Cluster Computing Systems // MethodsX. 2021. Vol. 8. Article number 101437. DOI: 10.1016/j.mex.2021.101437.

16. Boisvert R.F., Pozo R., Remington K.A. The Matrix Market Exchange Formats: Initial Design: tech. rep. / NISTIR 5935. National Institute of Standards; Technology. Gaithersburg, MD, 1996. P. 14. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir5935.pdf.

17. Соколинский Л.Б., Соколинская И.М. О генерации случайных задач линейного программирования на кластерных вычислительных системах // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2021. Т. 10, № 2. С. 38–52. DOI: 10.14529/cmse210103.

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

19. Dolganina N., Ivanova E., Bilenko R., Rekachinsky A. HPC Resources of South Ural State University // Parallel Computational Technologies. PCT 2022. Communications in Computer and Information Science, vol. 1618 / ed. by L. Sokolinsky, M. Zymbler. Cham: Springer, 2022. P. 43–55. DOI: 10.1007/978-3-031-11623-0_4.

20. Klee V., Minty G.J. How good is the simplex algorithm? // Inequalities - III. Proceedings of the Third Symposium on Inequalities Held at the University of California, Los Angeles, Sept. 1-9, 1969 / ed. by O. Shisha. New York-London: Academic Press, 1972. P. 159–175.

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

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

Соколинская Ирина Михайловна, к.ф.-м.н., доцент, кафедра вычислительной математики и высокопроизводительных вычислений, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

DOI: 10.14529/cmse240301

NUMERICAL IMPLEMENTATION OF SURFACE MOVEMENT METHOD IN LINEAR PROGRAMMING

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

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

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

Received: 10.06.2024

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

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

FOR CITATION

Sokolinsky L.B., Olkhovsky N.A., Sokolinskaya I.M. Numerical Implementation of Surface Movement Method in Linear Programming. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2024. Vol. 13, no. 3. P. 5–31. (in Russian) DOI: 10.14529/cmse240301.

References

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

4. Sokolinskaya I.M., Sokolinsky L.B. On the Solution of Linear Programming Problems in the Age of Big Data. Parallel Computational Technologies. PCT 2017. Communications in Computer and Information Science, vol. 753. / ed. by L. Sokolinsky, M. Zymbler. Cham, Switzerland: Springer, 2017. P. 86–100. DOI: 10.1007/978-3-319-67035-5_7.

5. Branke J. Optimization in Dynamic Environments. Evolutionary Optimization in Dynamic Environments. Genetic Algorithms and Evolutionary Computation, vol. 3. Boston, MA: Springer, 2002. P. 13–29. DOI: 10.1007/978-1-4615-0911-0_2.

6. Dantzig G.B. Linear programming and extensions. Princeton, N.J.: Princeton university press, 1998. 656 p.

7. Zorkaltsev V., Mokryi I. Interior point algorithms in linear optimization. Journal of applied and industrial mathematics. 2018. Vol. 12, no. 1. P. 191–199. DOI: 10.1134/S1990478918010179.

8. Mamalis B., Pantziou G. Advances in the Parallelization of the Simplex Method. Algorithms, Probability, Networks, and Games. Lecture Notes in Computer Science, vol. 9295 / ed. by C. Zaroliagis, G. Pantziou, S. Kontogiannis. Cham: Springer, 2015. P. 281–307. DOI: 10.1007/978-3-319-24024-4_17.

9. Olkhovsky N.A., Sokolinsky L.B. Surface Movement Method for Linear Programming. 2024. DOI: 10.48550/arXiv.2404.12640. arXiv: 2404.12640.

10. Sokolinsky L.B., Sokolinskaya I.M. Apex Method: A New Scalable Iterative Method for Linear Programming. Mathematics. 2023. Vol. 11, no. 7. P. 1654. DOI: 10.3390/MATH11071654.

11. Maltsev A. The basics of linear algebra. Moskow: Science. The main editorial office of the phys-math literature, 1970. 402 p. (in Russian).

12. Vasin V.V., Eremin I.I. Operators and Iterative Processes of Fejér Type. Theory and Applications. Berlin, New York: Walter de Gruyter, 2009. 155 p. Inverse and III-Posed Problems Series. DOI: 10.1515/9783110218190.

13. Gould N.I. How good are projection methods for convex feasibility problems?. Computational Optimization and Applications. 2008. Vol. 40, no. 1. P. 1–12. DOI: 10.1007/S10589-007-9073-5.

14. Sokolinsky L.B. BSF: A parallel computation model for scalability estimation of iterative numerical algorithms on cluster computing systems. Journal of Parallel and Distributed Computing. 2021. Vol. 149. P. 193–206. DOI: 10.1016/j.jpdc.2020.12.009.

15. Sokolinsky L.B. BSF-skeleton: A Template for Parallelization of Iterative Numerical Algorithms on Cluster Computing Systems. MethodsX. 2021. Vol. 8. Article number 101437. DOI: 10.1016/j.mex.2021.101437.

16. Boisvert R.F., Pozo R., Remington K.A. The Matrix Market Exchange Formats: Initial Design: tech. rep. / NISTIR 5935. National Institute of Standards; Technology. Gaithersburg, MD, 1996. P. 14. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir5935.pdf.

17. Sokolinsky L.B., Sokolinskaya I.M. FRaGenLP: A Generator of Random Linear Programming Problems for Cluster Computing Systems. Parallel Computational Technologies. PCT 2021. Communications in Computer and Information Science, vol. 1437 / ed. by L. Sokolinsky, M. Zymbler. Cham: Springer, 2021. P. 164–177. DOI: 10.1007/978-3-030-81691-9_12.

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

19. Dolganina N., Ivanova E., Bilenko R., Rekachinsky A. HPC Resources of South Ural State University. Parallel Computational Technologies. PCT 2022. Communications in Computer and Information Science, vol. 1618 / ed. by L. Sokolinsky, M. Zymbler. Cham: Springer, 2022. P. 43–55. DOI: 10.1007/978-3-031-11623-0_4.

20. Klee V., Minty G.J. How good is the simplex algorithm?. Inequalities - III. Proceedings of the Third Symposium on Inequalities Held at the University of California, Los Angeles, Sept. 1-9, 1969 / ed. by O. Shisha. New York-London: Academic Press, 1972. P. 159–175.

УДК 51.77 **DOI: 10.14529/cmse240302**

ПРОЕКТИРОВАНИЕ ОПТИМАЛЬНОЙ ТРАЕКТОРИИ ПРОВЕДЕНИЯ МОНИТОРИНГА ОБЪЕКТОВ

© 2024 М. Саллам, Т.А. Макаровских

Южно-Уральский государственный университет
(454080 Челябинск, пр. Ленина, д. 76)
E-mail: mohamedslam2000@yahoo.com, Makarovskikh.T.A@susu.ru

Поступила в редакцию: 25.04.2024

В статье рассматривается задача разработки системы для моделирования траекторий движения робота, проводящего мониторинг в замкнутой области с земли: осмотр помещений, съемка промышленных объектов, оценка состояния фруктовых деревьев и т.д. Во время мониторинга на пути робота могут возникнуть некоторые препятствия: временно возникшие и устранимые (люди, небольшая мебель, бытовая техника, зона чрезвычайной ситуации) или постоянные (стены, постоянно установленное оборудование, зафиксированная мебель). Управление роботом осуществляется с помощью разработанной авторами программы, в которой хранится база данных маршрутов, успешно преодоленных роботом ранее, для наиболее точного определения траектории движения робота с учетом препятствий, встречающихся вдоль построенного пути. В статье рассматривается алгоритм построения графовой модели области проведения мониторинга для последующего поиска кратчайшего маршрута робота, заключающийся в дискретизации исследуемой области, выявлении возможных путей перемещения робота, анализе имеющихся препятствий и задании расстояний между объектами исследования. Показано, что после построения такого графа возможно применить один из алгоритмов поиска гамильтонова пути. Он соединяет вершины графа, соответствующие точкам проведения мониторинга. Результатом применения алгоритма является кратчайший маршрут робота, либо сообщение о невозможности проведения мониторинга (частичной или полной неразрешимости задачи), если ряд возникших препятствий не позволяет проложить маршрут, проходящий через некоторые (либо через все) вершины.

Ключевые слова: маршрутизация, плоский граф, алгоритм, оптимизация, ограничения, задача о дроне и грузовике, точное земледелие, мониторинг.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Саллам М., Макаровских Т.А. Проектирование оптимальной траектории проведения мониторинга объектов // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2024. Т. 13, № 3. С. 32–46. DOI: 10.14529/cmse240302.

Введение

Мониторинг и анализ состояния различных наземных объектов является актуальной задачей во многих сферах деятельности — от промышленности до сельского хозяйства. Использование человеческих ресурсов в этом случае часто затруднено или невозможно, например, при обследовании нескольких гектаров посадок плодовых деревьев или сельскохозяйственных культур, при мониторинге большой территории в помещении или с вредными для человека условиями окружающей среды, при охране жилых помещений, обследовании инфекционных отделений больниц и т.д.

Во всех рассмотренных выше случаях в обследуемом пространстве имеется ряд препятствий, которые можно разделить на постоянные (деревья, сооружения, неподвижная мебель) и временные (люди, животные, грузы). Устройство (робот), осуществляющее мониторинг, должно учитывать эти препятствия при расчете оптимального способа проведения мониторинга и планировать маршрут своего передвижения с учетом этих препятствий. Использование беспилотных летательных аппаратов (БПЛА) могло бы позволить не учиты-

вать препятствия, однако при обследовании закрытых помещений невозможно проводить мониторинг с воздуха, в ряде случаев и в открытых пространствах (например, при обследовании плодовых деревьев) целесообразно проводить съемку с земли, а не с воздуха.

Известные подходы в целом не учитывают каких-либо препятствий. Классическая постановка задачи учитывает количество объектов, которые необходимо посетить. В [1] используется подход к решению проблемы покрытия как к известной $\mathcal{NP}$-трудной задаче коммивояжера (Travelling Salesman Problem, TSP) путем включения алгоритмов выбора дерева для повышения эффективности и точности. Этот подход представляет исследуемую область в виде графа, где каждая вершина представляет точку покрытия, а ребра — возможные переходы между ними. В этом случае граф является взвешенным, а веса ребер содержат расстояния между посещаемыми вершинами. С целью снижения сложности вычислений, авторы используют различные алгоритмы выбора дерева, чтобы найти наилучшую возможную отправную точку для алгоритма решения задачи коммивояжера. Результаты моделирования показывают, что по сравнению с классическими алгоритмами решения TSP предлагаемый метод значительно повышает качество покрытия при одновременном снижении вычислительных затрат. Этот подход довольно хорош, но он используется для решения задачи для беспилотника, поэтому наличие препятствий там не учитывается.

Некоторые идеи о том, как найти путь, проходящий через все объекты интереса, подробно изучены в [2] и [3], где авторы рассматривают так называемый мегаполис (непустое конечное множество) для выполнения некоторой работы. Эта работа, с точки зрения данного исследования, может и быть мониторингом. Выбор порядка выполнения задач зависит от условий предшествования, которые локализованы для некоторого набора задач. Функции затрат, используемые при формировании аддитивного критерия, могут включать в себя зависимость от перечня задач. Для построения решения предлагается двухэтапная процедура, основанная на методах динамического программирования. Тем не менее, эти подходы также не учитывают ни постоянные, ни движущиеся препятствия. Поэтому целесообразно разработать подход, учитывающий возможные препятствия.

Еще один подход для решения той же задачи рассмотрен в [4], но в этой статье также не учитываются препятствия. Рассматриваемая в этой статье задача имеет общие черты с так называемой задачей о дроне и грузовике [5], в которой также рассматриваются устройство для проведения мониторинга (беспилотник) и устройство для обслуживания беспилотника (грузовик). Отличия в рассматриваемой в нашей статье задаче заключаются в том, что робот должен уметь преодолевать препятствие, а в некоторых случаях грузовик не требуется.

В данной статье рассматривается алгоритм формализации задачи для построения кратчайшего маршрута для робота, осуществляющего мониторинг с земли. Управление роботом осуществляется с помощью разработанной авторами программы, в которой хранится база данных маршрутов, успешно преодоленных роботом, для наиболее точного определения траектории движения робота с учетом препятствий, встречающихся на его пути. Статья состоит из введения, трех разделов и заключения.

В первом разделе приводится постановка задачи, описание ограничений на построение маршрута. Во втором разделе обсуждается общая схема алгоритма проведения мониторинга, графовая постановка задачи, описание структуры базы данных препятствий, определяются условия разрешимости задачи для имеющихся условий. Третий раздел посвящен описанию разработанного авторами программного обеспечения для проведения мониторинга. В заключении приводятся выводы и определяются направления дальнейших исследований.

1. Постановка задачи и описание ограничений

Допустим, необходимо осмотреть группу объектов M_i, $i = 1, \ldots, K$, расположенных в некоторой ограниченной области F. Область F представляет собой многоугольник, как правило, невыпуклый. Координаты N углов многоугольника (x_j, y_j), $j = 1, \ldots, N$ заданы в виде вектора, координаты указаны по часовой стрелке, начиная с наименьшего значения первой координаты. Исследуемые объекты M_i полностью расположены в области F. Расстояния $L(M_i, M_j)$, $i, j = 1, \ldots, K$, $i \neq j$ между всеми парами объектов известны. Задача состоит в том, чтобы проводить мониторинг объектов M_i, используя минимум ресурсов устройства робота или беспилотника. В данном случае под ресурсами понимается: расход заряда батареи (как правило, во время мониторинга требуется подзарядка), размер карты памяти (как правило, объема памяти для мониторинга всех объектов недостаточно и требуется замена карты). Подзарядку аккумулятора или замену карты памяти можно выполнить как на месте, так и после возврата устройства мониторинга на стоянку. В первом случае потребуется так называемый «грузовик», для которого строится маршрут доставки аккумулятора или карты памяти. Во втором случае необходимо учитывать время возвращения на стоянку до того, как будет израсходован заряд аккумулятора, и выстраивать кратчайший маршрут возвращения. Что касается карты памяти, можно осуществлять ее замену в процессе проведения мониторинга либо осуществить алгоритм передачи полученной информации и автоматического удаления переданных файлов.

Таким образом, устройство мониторинга, в дальнейшем будем называть его «робот», находится на парковке с координатами (P_x, P_y). Задача состоит в том, чтобы проложить оптимальный маршрут робота от точки (P_x, P_y), называемой парковкой, до выполнения работ (мониторинга) вблизи объектов M_i, $i = 1, \ldots, K$. В общем случае объекты M_i определяются векторами координат ограничивающих их полигонов. Целью мониторинга является получение изображений с помощью робота для изучения состояния объектов M_i и проведения дальнейшего анализа.

В идеале мониторинг осуществляется без ограничений, но, очевидно, такой случай является исключительным. Чтобы решить поставленную задачу, можно рассмотреть один из частных случаев решения задачи коммивояжера (TSP) [6] и использовать один из известных алгоритмов (после проведения вычислительных эксериментов, можно определить наиболее подходящий алгоритм для рассматриваемой задачи). Однако при перемещении между объектами могут возникать различного рода препятствия, которые робот по определенным причинам не может преодолеть. Наличие этих препятствий может повлечь пересчет и пересмотр построенных ранее маршрутов и привести к частичной или полной неразрешимости задачи.

Препятствия S_i, $i = 1, \ldots, s$ подразделим на временные и постоянные. К временным препятствиям относятся люди и животные (в т.ч. мертвые животные), а также техника (автомобили, тракторы, погрузчики и т.д.), оставленная на пути движения робота, временные хранилища (например, стог сена, контейнер с посылками или товарами). Это временные препятствия, информация о размерах которых хранится и используется для анализа при корректировке траектории в будущем, местоположение таких препятствий в базе не сохраняется. Остальные препятствия являются постоянными (стационарными или малоподвижными). К ним относятся новые стены и перегородки, конструкции, закрепленная мебель. Наличие и размеры этих препятствий также заносится в базу данных для анализа маршрута, поскольку их местоположение является постоянным или долгосрочным.

Чтобы определить траекторию движения робота, необходимо решить следующие задачи.

1. Создать базу данных маршрутов: идеальных (без каких-либо препятствий), учитывающих влияние внешних факторов (температура, сила ветра и т.д.), маршрутов с непреодолимыми препятствиями (доступны только маршруты со стационарными и подвижными препятствиями, добавленные ранее в базу данных). Под препятствием мы будем понимать любой объект, который стоит на пути движения робота и препятствует дальнейшему следованию по заданному ранее маршруту.
2. Определить разрешимость задачи маршрутизации (возможно ли построить маршрут для наблюдения за интересующими объектами при наличии существующих ограничений и препятствий). Разрешимость может быть полной, когда возможно провести мониторинг во всех указанных точках M_i, $i = 1, \ldots, K$; частичной, когда существуют точки M_s, не достижимые при имеющемся наборе препятствий. Если все M_i, $i = 1, \ldots, K$ не достижимы, такая задача называется неразрешимой.

В общем случае, существует K объектов для мониторинга, координаты каждого из них известны. Делается фото или видеозапись этого объекта, которая затем отправляется на сервер для дальнейшего анализа и принятия решения. Робот делает снимки по указанным координатам и переходит к следующему объекту. Когда съемка проведена по всем указанным координатам или была выявлена невозможность дальнейшего наблюдения, робот возвращается на парковку.

Если рабочий ресурс робота исчерпан, он либо возвращается на парковку (необходимо, чтобы было достаточно ресурса для возврата), либо отправляет оператору сообщение о необходимости устранения неполадки. В последнем случае необходимо решить проблему построения маршрута для служебного «грузовика», доставляющего пополненные ресурсы.

2. Алгоритм маршрутизации робота

2.1. Схема алгоритма

Схема алгоритма движения робота во время мониторинга приведена на рис. 1.

На рисунке предполагается, что маршрут P_i уже учитывает все внешние воздействия, и в нем учитываются все возможные подзарядки робота и замены карты памяти, если ресурса недостаточно для завершения полного цикла мониторинга. Перемещение вдоль маршрута P_i означает перемещение робота на одну позицию по маршруту. Считается, что все пространство, исследуемое роботом, делится на элементарные сегменты (плитки) одинаковой длины и ширины.

Возможностью продолжить движение считается возможность добраться до центральной точки следующего элементарного отрезка. Если на пути робота к этой точке обнаружено препятствие, дальнейшее продолжение движения по ранее выбранному пути считается невозможным и выполняется поиск обходного маршрута. Если на одном из этапов мониторинга не удается найти обходной маршрут, робот отправляет оператору сообщение о невозможности дальнейшего проведения мониторинга и отключается (возвращается на стоянку для подзарядки). Маршрут по известной ранее области уже определен и может не рассчитываться заново, тем не менее, после обнаружения препятствия и размещения его на карте проведения мониторинга необходимо переопределить все длины путей, проходящих через область, перекрытую препятствием, а также обновить информацию о достижимости еще не обследованных объектов мониторинга. В большинстве случаев появление каждого

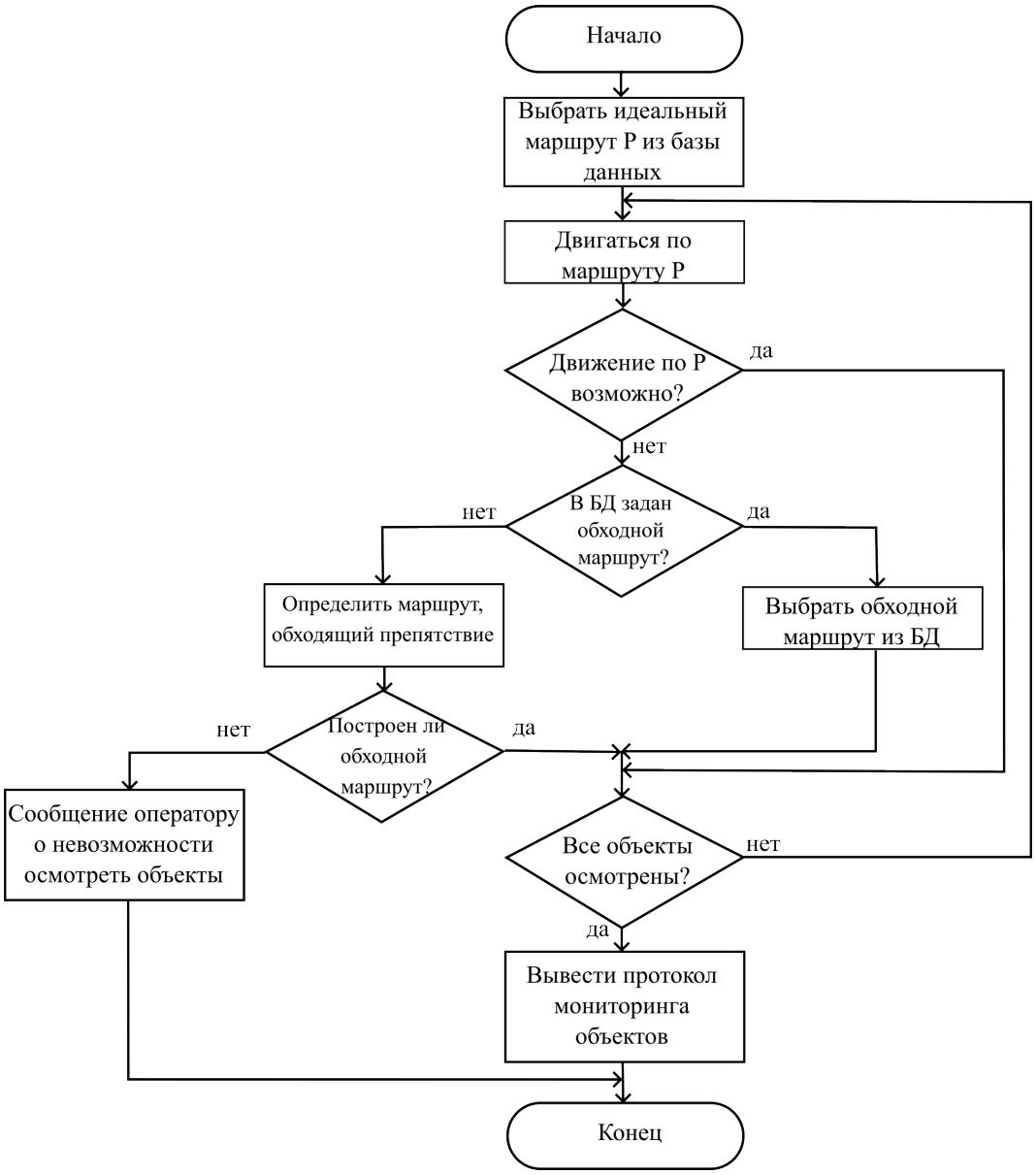


Рис. 1. Схема алгоритма перемещения робота в зоне мониторинга

препятствия требует заново решить задачу TSP для обновленной карты. Хранение ранее построенных маршрутов в базе данных позволит в ряде случаем избежать повторного решения TSP и выбрать уже имеющееся решение задачи. Например, если на пути робота закрыта дверь либо шлагбаум, это постоянное препятствие, решение для которого хранится в базе данных. По известным габаритам временных препятствий (например, припаркованного автомобиля или погрузчика) можно определить время для объезда и принять решение о целесообразности повторного решения задачи маршрутизации.

Препятствия на пути робота могут отслеживаться несколькими способами:

- ультразвук, который считается эффективным и быстрым, он позволяет рассчитать расстояния с точностью до миллиметра, вплоть до расстояния в три метра (которое может быть увеличено в зависимости от типа установленного ультразвукового датчика), чтобы определить расстояние между препятствием и роботом, а также габариты препятствия;

- программируемые интеллектуальные камеры для отслеживания новых препятствий на пути робота.

Таким образом, на основе построенной алгоритмической схемы могут быть поставлены следующие задачи.

1. Построение маршрута P_i с учетом внешних воздействий.
2. Определение возможности продолжения движения с помощью алгоритмов компьютерного зрения.
3. Поиск обходного маршрута с учетом внешних воздействий и обнаруженных препятствий.
4. Обновление информации в базе данных о препятствиях и обходных путях.

2.2. Графовая модель исследуемой области и алгоритм маршрутизации

Всю область, используемую для перемещения робота, разделим на небольшие квадратные плитки R_j, $j = 1, \ldots, M$ с помощью алгоритма дискретизации [7]. Длина стороны каждой плитки равна расстоянию, проходимому роботом, между моментами передачи информации оператору. Обычно эта величина равна 30–50 см. Обозначим эту величину через l. Робот может перемещаться между углами плитки по горизонтали либо вертикали (тогда длина пути между парой моментов обмена информацией равна l), либо по диагонали (в этом случае робот преодолеет расстояние $\sqrt{2}l$ до следующего момента передачи информации).

Представим исследуемую область как граф $G = (V, E)$, множество вершин V которого являются точками, в которых робот обменивается информацией с прикладной программой, а ребра E — связями между ближайшими вершинами (как отмечалось выше, переходы осуществляются по горизонтали, вертикали либо диагонали). Поскольку при решении практической задачи точки дискретизации располагаются на расстоянии порядка 50 см, их количество становится довольно большим, поэтому $|V|$ может содержать тысячи вершин. Следовательно, неразумно использовать все эти вершины для определения оптимального маршрута. Кроме того, граф является разреженным — степень каждой его вершины равна 2–8, в то время как количество вершин значительно выше. Веса ребер (длины переходов) будут равны либо l, либо $\sqrt{2}l$. Для хранения информации о графе будем использовать список смежных вершин.

Подразделим все вершины графа $V = V_p \cup V_m \cup V_d \cup V_t$ на следующие подмножества:

- вершины «парковки» ($p_i \in V_p$) — вершины, с которых робот может подзарядить аккумулятор либо получить сервисное обслуживание, стартовые вершины;
- вершины мониторинга ($m_i \in V_m$) — вершины, в которых проводится мониторинг;
- вершины маршрута объезда ($d_i \in V_d$) вершины, принадлежащие маршруту объезда препятствия;
- транзинтые вершины $v \in V_t$ — все остальные вершины графа, которые могут быть включены в маршрут.

Рассмотрим пример, приведенный на рис. 2. На рисунке приведена невыпуклая область, ограниченная жирной линией. Вершины, принадлежащие границе, в маршруте не используются; P — вершины для парковки V_p; $M_1 \ldots, M_6$ — объекты мониторинга; S_1, S_2 — некоторые препятствия. Вершины, отмеченные красным цветом, являются точками (вершинами) мониторинга V_m, вершины, обозначенные черным цветом, являются вершинами объезда V_d. Транзитные вершины V_t на этом рисунке не указаны, если их количество велико

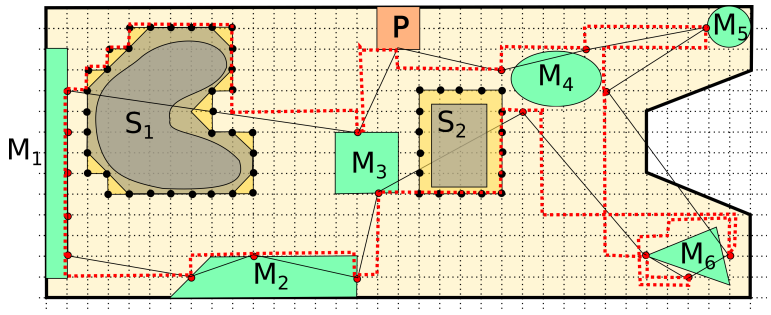


Рис. 2. Пример карты для проведения мониторинга

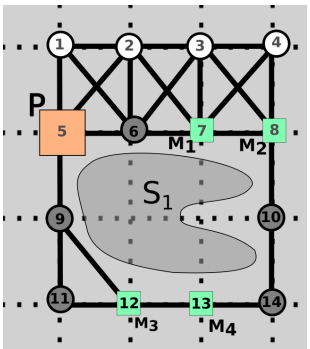
(это места пересечения прямых линий, используемые для дискретизации). Те же обозначения используются для всех последующих примеров.

Информация о графе хранится в текстовом файле, имеющем следующий формат:

- Строка 1: количество вершин и ребер графа через пробел;
- Строка i ($i = 2, \ldots, |V|$): тип текущей вершины (P — парковка, M — мониторинг, D — объезд, T — транзитная), и далее список вершин, смежных v_i, каждое значение через пробел. Сначала указываются вершины, инцидентные ребрам с весом l, далее располагается служебный символ «|», после него следуют вершины, инцидентные ребрам с весом $\sqrt{2}l$. Если диагональных ребер нет, после символа «|» ничего не записывается. Порядок следования вершин, инцидентных ребрам одного веса, не принципиален.

Если вершина одновременно является и вершиной объезда, и вершиной мониторинга (либо парковки), то при задании информации в файле она маркируется как вершина для мониторинга (либо парковки).

Например, для графа, приведенного на рис. 3а, файл, его задающий, имеет следующий вид (рис. 3б).




```
14 25
T 2 5 | 6
T 1 3 6 | 5 7
T 2 7 4 | 6 8
T 3 8 | 7
P 1 6 9 | 2
D 5 2 7 | 1 3
M 6 3 8 | 4 2
M 7 4 10 | 3
D 5 11 | 12
D 8 13 |
T 9 12 |
M 11 4 | 9
M 3 14 |
D 4 10 |
```

(а) Графическое представление (б) Представление в памяти компьютера

Рис. 3. Пример задания графовой модели исследуемой области

Пользуясь информацией для этого графа можно построить производный взвешенный граф $G^* = (V_P \cup V_M, E^*)$, вес каждого ребра которого равен длине кратчайшего пути между соответствующими парами вершин в графе G. Рассмотрим вершины $v, w \in V_P \cup V_M$ в графе G. Если путь из вершины v в вершину w проходит через некоторую вершину $v_k \in V_P \cup V_M$, то вершины v и w не будут смежны в графе G^*. Очевидно, если для всех пар вершин $v, w \in V_P \cup V_M$ пути, их соединяющие, не проходят ни через одну из вершин $v_k \in V_P \cup V_M$, то граф G^* является полным.

Так, для графа, представленного на рис. 3, граф G^* имеет вид, представленный на рис. 4.

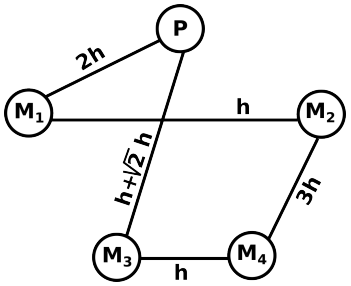


Рис. 4. Граф G^* для рассмотренного примера

Заметим, что в графе отсутствуют ребра $\{P, M_2\}$ (поскольку путь $P - 6 - M_1 - M_2$ содержит вершину M_1), $\{P, M_4\}$ (поскольку путь $P - 9 - M_3 - M_3$ содержит вершину M_3), $\{M_1, M_4\}$ (поскольку путь $M_1 - M_3 - 10 - M_4$ содержит вершину M_3), $\{M_1, M_3\}$ (поскольку путь $M_1 - 6 - P - 9 - M_3$ содержит вершину P), $\{M_2, M_3\}$ (поскольку путь $M_2 - 10 - M_4 - M_3$ содержит вершину M_4). Таким образом, имеем граф G^*, соответствующий метрической постановке задачи коммивояжера.

Приведем алгоритм поиска гамильтонова цикла T минимальной длины в графе G^*. Полученный в случае разрешимой задачи цикл T будет являться решением поставленной задачи о маршрутизации робота, проводящего мониторинг при условии отсутствия ограничений на время перемещения.

Задачу получения оптимального маршрута для проведения мониторинга можно разделить на следующие этапы.

2.2.1. Определение идеального маршрута

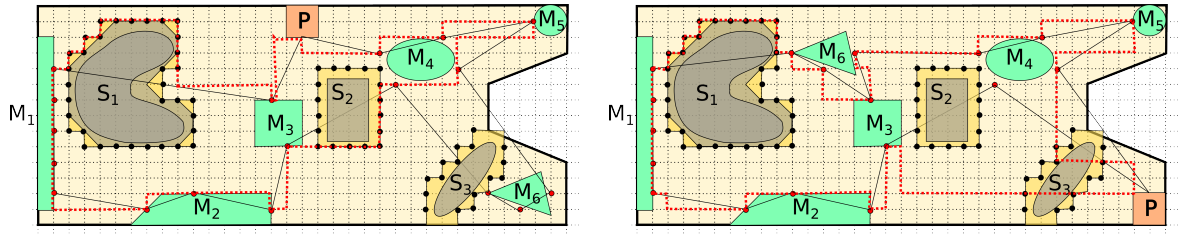
Введем определения разрешимости задачи, которыми будем пользоваться в дальнейших рассуждениях.

Определение 1. Задача маршрутизации является *разрешимой*, если граф G, задающий исследуемую область, является связным, либо все вершины из V_m принадлежат компонентам связности содержащим хотя бы одну вершину из V_p.

В примере, показанном на рис. 2, эта задача разрешима, и линией из красных точек показан маршрут мониторинга.

Определение 2. Задача является *частично разрешимой*, если существует b вершин из V_m, $b \neq |V_m|$, принадлежащих компонентам связности графа G, не содержащим вершину из V_p.

На рис. 5а приведена частично разрешимая задача.



(а) Частично разрешимый маршрут (б) Полностью неразрешимый маршрут

Рис. 5. Пример карты для проведения мониторинга

Объект M_6 не может быть включен в маршрут проведения мониторинга, поскольку все пути к нему перекрыты препятствием S_3. Если S_3 является стационарным препятствием, то M_6 никогда не будет проверен. Если это временное препятствие, то необходимо принять решения для его устранения. Здесь маршрут, отмеченный линией из красных точек, проходит через все контролируемые объекты, кроме M_6.

Определение 3. Задача является *не разрешимой*, если ни одна вершина из V_m не принадлежит компоненте связности графа G, содержащей V_p.

На рис. 5б приведена неразрешимая задача.

В данном случае верно то же рассуждение о препятствии S_3, что и для предыдущего случая.

Для построения маршрута P_{ideal} наименьшей длины между вершинами мониторинга V_m необходимо выполнить следующие действия.

Алгоритм МАРШРУТИЗАЦИЯ (G)

Исходные данные: граф G, l – размер плитки.

Выходные данные: – замкнутая гамильтонова цепь кратчайшей длины с началом в вершине v_0.

Шаг 1. Определить разрешимость задачи маршрутизации. Для этого, используя алгоритм просмотра вершин (поиск в ширину или глубину) определить количество компонент связности заданного графа $G = G_1 \cup G_2 \cup \ldots, \cup G_k$. Если задача не разрешима, завершить работу алгоритма.

Шаг 2. Построение графа для определения маршрута. Для каждой компоненты связности G_i построить взвешенный полный граф $G_{T_i} = G^*(|V_P| \cup |V_M|, E^*)$. Каждое ребро $e \in E^*$ этого графа будет иметь вес, равный $\phi(e) = \min d(v_1(e), v_2(e))$, определяемый как кратчайший путь между соответствующими вершинами в графе G_i, длина каждого ребра которого равна l либо $\sqrt{2}l$. Для решения этой задачи используется алгоритм Дейкстры. Если найденный путь $P(v_1(e), v_2(e))$ содержит в качестве транзитной вершину $v \in V_P \cup V_M$, то вершины $v_1(e)$ и $v_2(e)$ в графе G^* не будут связаны отношением непосредственной достижимости.

Шаг 3. Поиск гамильтонова цикла минимальной длины. В каждом графе $G_{T_i} = K_{|V_P|\cup|V_M|}$ определить гамильтонов цикл минимальной длины. Если длина цикла больше критической $L_{critical}$ (например, для прохождения пути такой длины требуется подзарядка), необходимо учесть посещение произвольной вершины $v \in V_p$. Такая задача известна как метрическая задача VRP (Vehicle Routing Problem) [8]. Он и будет соответствовать маршруту робота для известной карты препятствий. Поскольку данная задача является $\mathcal{NP}$-трудной в сильном смысле, для ее решения можно использовать некоторую

эвристику. Наиболее очевидной эвристикой является «ближайший непройденный город», позволяющей за время $O(|V|^2)$ получить допустимое решение задачи, здесь V — число вершин графа, для которого ищется решение. Возможными альтернативами являются методы муравьиной колонии и алгоритм A* (метод поиска по первому наилучшему совпадению в графе).

Для графа, представленного на рис. 3, может быть построено два кратчайших маршрута:

$$T_1 = P, M_1, M_2, M_4, M_3, P \text{ и } T_2 = P, M_3, M_4, M_2, M_2, P.$$

Длина маршрута в данном случае равна $8l + \sqrt{2}l$. Этот пример является тривиальным случаем, в котором построенный производный граф G^* является простым циклом. В общем случае полученный граф не является простым циклом и для поиска гамильтонового цикла кратчайшей длины потребуется использование эвристических алгоритмов.

Полученный в результате работы алгоритма МАРШРУТИЗАЦИЯ(G) идеальный маршрут изображен тонкой черной линией на рис. 2. В дальнейших рассуждениях будем полагать, что идеальный маршрут уже найден.

2.2.2. Определение маршрутов обхода для каждого препятствия

На данном этапе предпожим, что все препятствия размещены на карте и их местоположение известно. Каждое препятствие S_i покрыто плитками размером, равным шагу дискретизации. Итак, каждая плитка прикрепляется к транзитным вершинам своими углами. Плитка включается в перекрытие препятствия, если хотя бы одно ее внутренняя точка находится внутри контура препятствия. После того, как все плитки определены, вершины, принадлежащие внешнему контуру полученного прямоугольника, помещаются в набор V_d. Пример создания такого покрытия показан на рис. 6. Такой же подход используется для определения маршрутов объезда проверяемой территории. Для каждого обнаруженного и обследованного неподвижного препятствия маршрут его объезда сохраняется в базе данных. Если оно подвижное, то маршрут объезда может быть определен в режиме реального времени с помощью оборудования компьютерного зрения устройства мониторинга. В этом случае алгоритм получения вершин V_d будет другим.

2.2.3. Определение реального маршрута обхода для робота с учетом препятствий

Для этого нужно проанализировать идеальный маршрут P_{ideal} и получить реальный маршрут P_{real}, учитывающий все препятствия и проходящий через $v \in V_t$. Если найдется участок маршрута, относящийся к препятствию (как на рис. 6), то заменить этот отрезок маршрута кратчайшим путем, состоящим только из вершин $v \in V_d$. На этом шаге также требуется проверить разрешимость задачи мониторинга.

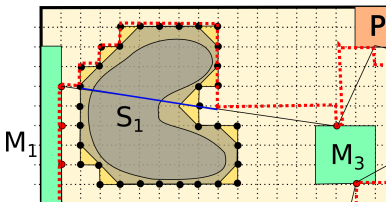


Рис. 6. Путь, часть которого обходит препятствие

2.2.4. Вычислительная сложность определения маршрута мониторинга

Говоря о вычислительной сложности представленного алгоритма, мы видим, что она определяется вычислительной сложностью алгоритма, используемого для решения задачи VRP на шаге 1 алгоритма. Шаг 2 может быть выполнен за полиномиальное время $O(|V|^2)$, потому что в худшем случае требуется просмотреть все вершины графа G. Даже в этом случае время, необходимое для решения этой задачи, может быть достаточно большим, поскольку $|V|$ может содержать более 10^3 элементов. Для ускорения этого процесса могут быть использованы технологии параллельных вычислений.

Если препятствие представляет собой выпуклую область, то оно делится на две части идеальной траекторией, поэтому определение кратчайшего обходного пути заключается в проверке только двух путей. Если препятствие не является выпуклой областью, то количество путей, подлежащих проверке, является конечным множеством, поскольку $|V_{d_i}|$ для S_i является конечным множеством.

3. Программное обеспечение для проведения мониторинга

Основная идея работы системы мониторинга по отслеживанию и досмотру объектов с помощью роботов и беспилотных летательных аппаратов показана на рис. 7.

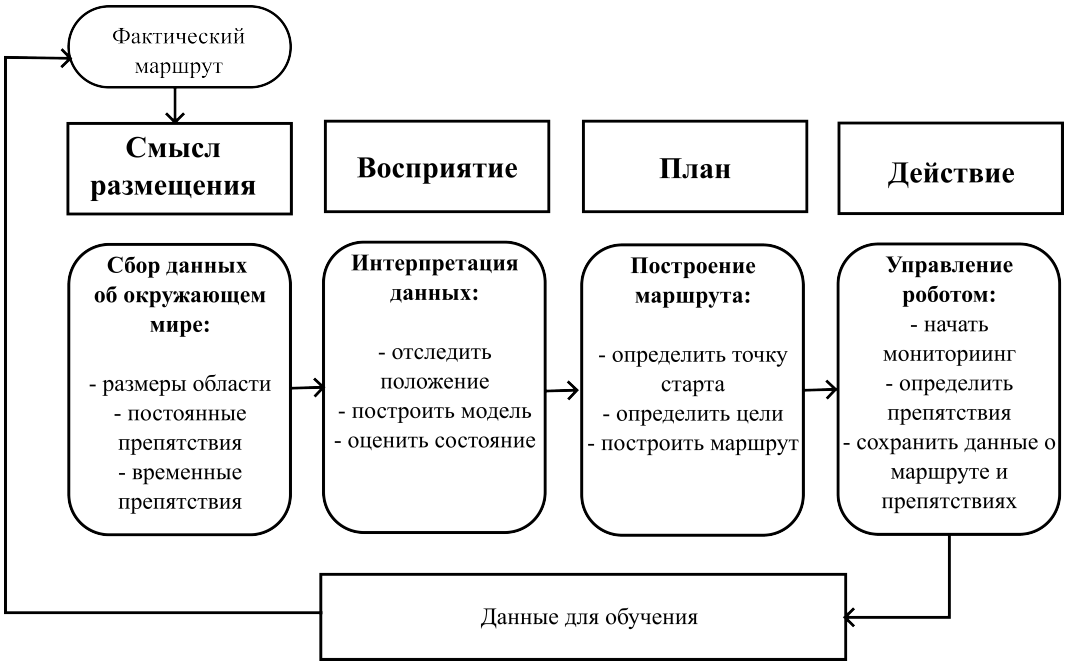


Рис. 7. Технологический цикл работы системы мониторинга

Система функционирует циклически. Здесь на этапе «Смысл размещения» определяется пространство для проведения мониторинга, получается информация об известных системе с предыдущих этапов препятствиях: временных и постоянных. На этапе «Восприятие» робот изучает текущее состояние окружающей среды. Возможно, здесь вносятся изменения в информацию о препятствиях, если они были перемещены либо удалены из обследуемой области. На этапе «План» выполняется построение либо корректировка маршрута от текущей точки до оставшихся непройденными точек проведения мониторинга. Этап «Действие» посвящен переходу устройства мониторинга в новую точку исследуемой области, сохране-

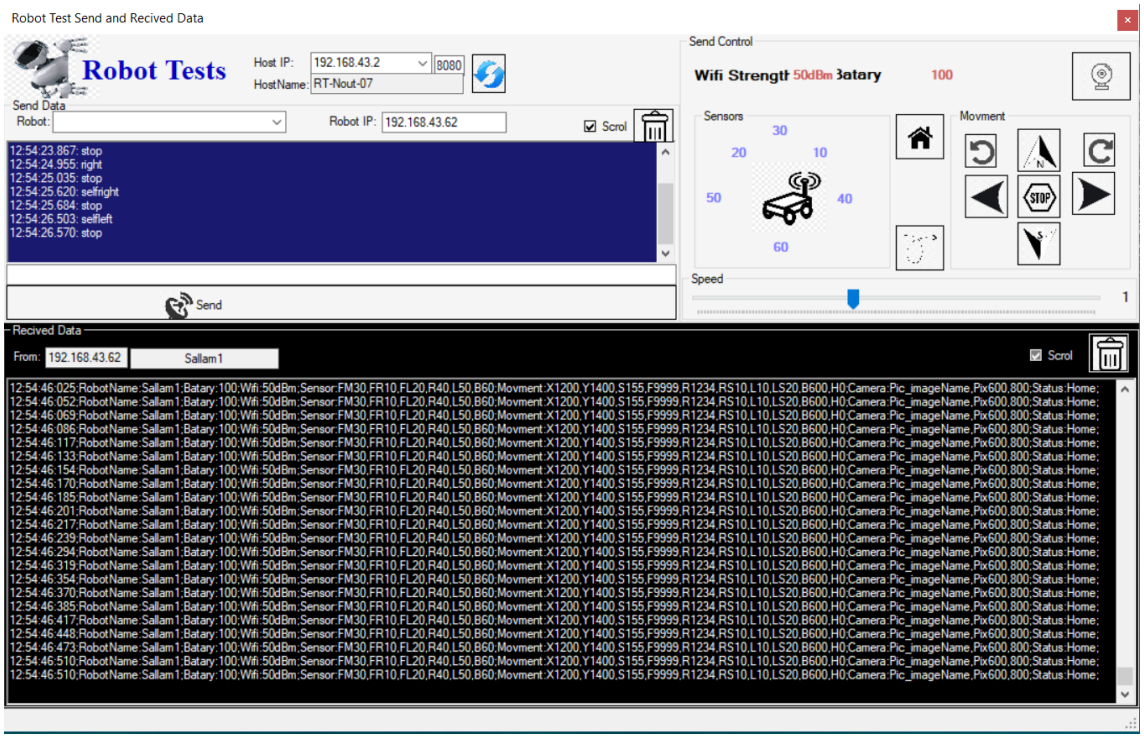


Рис. 8. Окно интерфейса программы для управления роботом при мониторинге объектов

нию полученной текущей информации о препятствиях. Эта информация становится новыми «данными для обучения» при построении дальнейших маршрутов и учитывается при формировании фактического маршрута, по которому движется устройство мониторинга.

Поэтому для управления устройством мониторинга было разработано настольное приложение для тестирования робота и получения данных с него (рис. 8), которое предназначено для дистанционного управления роботом (посредством панели управления) и передачи сообщений в консольном режиме, связи с роботом по Wi-Fi через IP-адрес устройства. Когда робот включен, данные передаются через ESP-карту (плата разработчика, на которой установлен микроконтроллер ESP32 с беспроводными интерфейсами WiFi/Bluetooth для передачи данных). Среди передаваемой информации:

- данные о расстоянии между роботом и ближайшим препятствием;
- данные о количестве шагов, необходимых для достижения этого препятствия по прямой;
- название файла с фотографией препятствия;
- координаты препятствия на карте исследуемой области;
- разрешение изображения;
- информация о состоянии робота (движется, стоит, припаркован).

Полученные данные записываются в CSV-файл. Управление роботом осуществляется на панели управления Send по аналогии с использованием пульта управления. При нажатии кнопки мыши на одной из кнопок управления робот перемещается по траектории, заданной оператором. Информация о командах, передаваемых роботу, передается на консольную панель. Роботом также можно управлять, отправляя команды на консольную панель.

Заключение

В статье рассмотрена задача планирования траектории мониторинга с препятствиями на исследуемой территории. Приведен алгоритм, позволяющий улучшить заданную идеальную траекторию (то есть решение задачи TSP без ограничений) в соответствии с существующими стационарными и подвижными препятствиями на пути робота. Информация о стационарных препятствиях сохраняется в базе данных и может быть получена по запросу. Предполагается, что подвижные препятствия могут иметь не известное расположение на карте исследуемой области и идентифицированы только после старта проведения мониторинга. Этот подход можно использовать для наблюдения за различными объектами с земли, в частности, в промещении.

К задачам будущих исследований можно отнести следующие. Прежде всего, необходимо определить наиболее подходящий для данной задачи алгоритм определения идеального пути и пересчета оптимальной траектории при возникновении подвижных препятствий. Они могут возникнуть после определения начального маршрута, и в этом случае предлагается использовать модуль искусственного интеллекта для распознавания препятствия, определения его контуров, занесения информации о препятствии в базу данных и нанесении его на карту проведения мониторинга. Информация о наличии временных препятствий должна обновляться в соответствии с заданным расписанием. Если препятствие устранено, то оно помещается в архивную таблицу базы данных и извлекается из нее, как только оно будет распознано на карте при очередном мониторинге.

Литература

1. Sundarrajan M., Jothi A., Prabakar D., Kadry S. The Smart Coverage Path Planner for Autonomous Drones Using TSP and Tree Selection // Mining Intelligence and Knowledge Exploration. MIKE 2023. Vol. 13924 / ed. by S. Kadry, R. Prasath. Springer, 2023. P. 161–172. Lecture Notes in Computer Science. DOI: 10.1007/978-3-031-44084-7_16.

2. Ченцов А.Г. Задача маршрутизации на узкие места с системой первоочередных заданий // Известия Института математики и информатики Удмуртского государственного университета. 2023. Т. 61. С. 156–186. DOI: 10.35634/2226-3594-2023-61-09.

3. Ченцов А.Г., Ченцов П.А. Экстремальная двухэтапная задача маршрутизации и процедуры на основе динамического программирования // Труды Института математики и механики УрО РАН. 2022. Т. 28, № 2. С. 215–248. DOI: 10.21538/0134-4889-2022-28-2-215-248.

4. Wu J., Li M., Gao C., *et al.* Research on Deployment Scheme and Routing Optimization Algorithm of Distribution Cable Condition Monitoring Devices // Energies. 2023. Vol. 16. P. 852–879. DOI: 10.3390/en16196930.

5. Chung S.H., Sah B., Lee J. Optimization for drone and drone-truck combined operations: A review of the state of the art and future directions // Computers & Operations Research. 2020. Vol. 123. DOI: 10.1016/j.cor.2020.105004.

6. Петунин А.А., Ченцов А.Г., Ченцов П.А. Оптимальная маршрутизация в задачах последовательного обхода мегаполисов при наличии ограничений // Челябинский физико-математический журнал. 2022. Т. 7, № 2. С. 209–233. DOI: 10.47475/2500-0101-2022-17205.

7. Makarovskikh T., Panyukov A., Abotaleb M., *et al.* Optimal Route for Drone for Monitoring of Crop Yields // Advances in Optimization and Applications. OPTIMA 2023. Vol. 1913 / ed. by N. Olenev, Y. Evtushenko, M. Jaćimović, *et al.* Springer International Publishing, 2024. P. 228–240. Communications in Computer and Information Science. DOI: 10.1007/978-3-031-48751-4_17.

8. Гимади Э.Х., Хачай М.Ю. Экстремальные задачи на множествах перестановок. Екатеринбург: Издательство УМЦ УПИ, 2016. 220 с.

Саллам Мохамед, аспирант, кафедра системного программирования, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

Макаровских Татьяна Анатольевна, д.ф.-м.н., доцент, кафедра системного программирования, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

DOI: 10.14529/cmse240302

DESIGNING THE OPTIMAL TRAJECTORY FOR MONITORING OBJECTS

© 2024 M. Sallam, T.A. Makarovskikh

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

E-mail: mohamedslam2000@yahoo.com, Makarovskikh.T.A@susu.ru

Received: 25.04.2024

The article considers the task of developing a system for modeling the trajectories of a robot monitoring in a closed area from the ground: inspection of premises, survey of industrial facilities, assessment of the condition of fruit trees, etc. During monitoring, some obstacles may arise in the robot's path: temporary and removable (people, small furniture, household appliances, emergency zone) or permanent (walls, permanently installed equipment, fixed furniture). The robot is controlled using a program developed by the authors, which stores a database of routes successfully overcome by the robot earlier, for the most accurate determination of the trajectory of the robot, taking into account obstacles encountered along the constructed path. The article considers an algorithm for constructing a graph model of the monitoring area for the subsequent search for the shortest route of the robot, which consists in discretization the area, identifying possible ways to move the robot, analyzing existing obstacles and setting distances between the objects of study. It is shown that after obtaining such a graph, it is possible to apply one of the algorithms for finding the Hamiltonian path in a graph. It connects the vertices of the graph corresponding to the monitoring points. The result of applying the algorithm is the shortest route of the robot, or a message about the impossibility of monitoring (partial or complete insolubility of the problem) if a number of obstacles do not allow you to plot a route passing through some (or all) vertices.

Keywords: routing, plane graph, algorithm, optimization, constraints, drone and truck problem, precision farming, monitoring.

FOR CITATION

Sallam M., Makarovskikh T.A. Designing the Optimal Trajectory for Monitoring Objects. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2024. Vol. 13, no. 3. P. 32–46. (in Russian) DOI: 10.14529/cmse240302.

References

1. Sundarrajan M., Jothi A., Prabakar D., Kadry S. The Smart Coverage Path Planner for Autonomous Drones Using TSP and Tree Selection. Mining Intelligence and Knowledge Exploration. MIKE 2023. Vol. 13924 / ed. by K. S., P. R. Springer, 2023. P. 161–172. Lecture Notes in Computer Science. DOI: 10.1007/978-3-031-44084-7_16.
2. Chentsov A.G. A bottleneck routing problem with a system of priority tasks. Izvestiya IMI UdGU. 2023. Vol. 61. P. 156–186. (in Russian) DOI: 10.35634/2226-3594-2023-61-09.
3. Chentsov A.G., Chentsov P.A. An extremal two-stage routing problem and procedures based on dynamic programming. Trudy Instituta Matematiki i Mekhaniki UrO RAN. 2022. Vol. 28, no. 2. P. 215–248. (in Russian) DOI: 10.21538/0134-4889-2022-28-2-215-248.
4. Wu J., Li M., Gao C., *et al.* Research on Deployment Scheme and Routing Optimization Algorithm of Distribution Cable Condition Monitoring Devices. Energies. 2023. Vol. 16. P. 852–879. DOI: 10.3390/en16196930.
5. Chung S.H., Sah B., Lee J. Optimization for drone and drone-truck combined operations: A review of the state of the art and future directions. Computers & Operations Research. 2020. Vol. 123. DOI: 10.1016/j.cor.2020.105004.
6. Petunin A.A., Chentsov A.G., Chentsov P.A. Optimal routing in problems of sequential traversal of megapolises in the presence of constraints. Chelyab. Phys.-Math. Journal. 2022. Vol. 7, no. 2. P. 209–233. (in Russian) DOI: 10.47475/2500-0101-2022-17205.
7. Makarovskikh T., Panyukov A., Abotaleb M., *et al.* Optimal Route for Drone for Monitoring of Crop Yields. Advances in Optimization and Applications. OPTIMA 2023. Vol. 1913 / ed. by N. Olenev, Y. Evtushenko, M. Jaćimović, *et al.* Springer International Publishing, 2024. P. 228–240. Communications in Computer and Information Science. DOI: 10.1007/978-3-031-48751-4_17.
8. Gimadi E.K., Khachai M.Y. Extreme Tasks on Sets of Permutations. Ekaterinburg: Izdatelstvo UMC UPI, 2016. 220 p. (in Russian).

УДК 004.942 **DOI: 10.14529/cmse240303**

ОПТИМАЛЬНОЕ УПРАВЛЕНИЕ ТРЕМЯ WORK-STEALING ДЕКАМИ В ДВУХУРОВНЕВОЙ ПАМЯТИ

© 2024 Е.А. Аксёнова, Е.А. Барковский, А.В. Соколов

Институт прикладных математических исследований
Карельского научного центра Российской академии наук
(185910 Петрозаводск, ул. Пушкинская, д. 11)
E-mail: aksenova@krc.karelia.ru, barkevgen@gmail.com, avs@krc.karelia.ru

Поступила в редакцию: 26.07.2024

При выполнении параллельных вычислений возникает проблема равномерного разделения задач между потоками. Одним из способов решения этой проблемы является применение распределенной динамической балансировки нагрузки. При таком способе балансировки каждый рабочий поток имеет свою очередь задач и потоки сами занимаются дальнейшем распределением задач. Широкое распространение получил метод балансировки «work-stealing», в котором один поток, у которого закончились задачи, может перехватывать задачи других потоков. Для реализации такого метода у каждого потока должен быть свой специализированный дек, в котором хранятся указатели на задачи. В этой статье предлагается и исследуется новый метод представления трех work-stealing деков в двухуровневой памяти. Рассматривается случай работы с тремя деками, когда в одном разделе быстрой памяти расположены две LIFO-части деков — два стека, которые растут навстречу друг другу, в другом разделе быстрой памяти расположены три FIFO-части, которые объединены в одну FIFO-очередь, из которой элементы только исключаются (кражи), и третья LIFO-часть. Средние части деков находятся в медленной памяти, обращение к ним происходит при переполнении или опустошении LIFO-частей или опустошении FIFO-частей деков, расположенных в быстрой памяти. Рассматривается задача оптимального разделения быстрой памяти для трех деков с заранее заданными вероятностями выполнения операций. Этот выбор зависит от характеристик уровней памяти, вероятностей операций и критерия оптимальности. В качестве критерия оптимальности рассматривается максимальное среднее время работы системы (среднее количество операций) до перераспределения памяти.

Ключевые слова: двухуровневая память, метод Монте-Карло, структуры данных, work-stealing деки.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Аксёнова Е.А., Барковский Е.А., Соколов А.В. Оптимальное управление тремя work-stealing деками в двухуровневой памяти // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2024. Т. 13, № 3. С. 47–60. DOI: 10.14529/cmse240303.

Введение

При выполнении параллельных вычислений возникает проблема равномерного разделения задач между потоками. Эту проблему решают с помощью статической и динамической балансировки задач [1]. Если свойства и особенности вычисляемых задач известны заранее, то применяют статическую балансировку. Такие задачи являются NP-полными [2], и для их решения можно составить расписание. Если информации о вычисляемых задач недостаточно, то целесообразно применять динамическую балансировку. В таком случае балансировка задач происходит во время работы системы [3]. Стратегию динамической балансировки, в свою очередь, можно разделить на централизованную и распределенную [4, 5]. В централизованной балансировке назначением задач занят специально для этого отведенный поток.

В распределенной балансировке каждый рабочий поток имеет свою очередь задач и потоки сами занимаются дальнейшем распределением задач. Существуют различные ме-

тоды распределенной балансировки задач. По методу «work-dealing» поток, у которого накопилось много задач, перераспределяет их среди других потоков [6, 7]. По методу «work-requesting» поток, у которого заканчиваются задачи, запрашивает их у других потоков [8]. Суть метода «work-stealing» в том, что если у потока заканчиваются задачи, он начинает перехватывать их у других потоков [9]. Метод work-stealing имеет широкое распространение [10] и применяется в следующих балансировщиках задач: Cilk; Intel TBB; dotNET TPL; X10 и других. Для реализации этого метода, у каждого потока должен быть свой специализированный дек, в котором хранятся указатели на задачи. Дек (англ. «double ended queue», сокращенно «deque») — это структура данных, в которой операции добавления (операция «push») и удаления (операция «pop») элементов происходят с обоих концов по принципу LIFO (рис. 1).

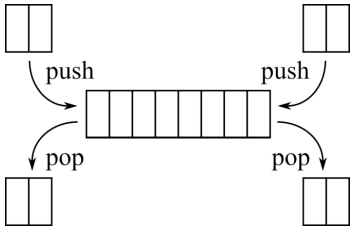


Рис. 1. Принцип работы дека

Во время работы системы, реализованной на основе метода work-stealing, потоки обращаются к своим декам по следующему принципу: после создания новой задачи, поток добавляет указатель на нее в вершину своего дека; если потоку требуется задача для выполнения и его дек не пуст, он берет указатель из вершины дека; если потоку требуется задача для выполнения и его дек пуст, он перехватывает указатель из основания дека другого потока. Таким образом, первые две операции выполняются с одного конца по принципу LIFO, а перехват элементов происходит из основания структуры данных (принцип FIFO). Дек, работающий таким образом, называется деком с ограниченным входом [11] или work-stealing деком (рис. 2).

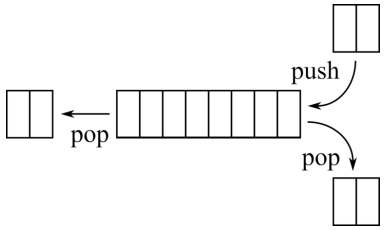


Рис. 2. Принцип работы work-stealing дека

Так как задачи балансировки нагрузки возникают в многопроцессорных системах [12] и в вычислительных сетях [13, 14], требуется эффективно использовать метод work-stealing. Для этого предлагают не только различные способы его реализации [15], но и способы управления work-stealing деками [16, 17].

Модель work-stealing балансировщика, основанная на аппарате теории массового обслуживания, была предложена в [18]. В [19, 20] предлагается и исследуется реализация экспериментального динамического work-stealing балансировщика. Деки можно представить в памяти с помощью различных методов [21]. Например, метод связного представления [22].

Для стеков и очередей модель на основе этого метода уже построена [23]. Также можно воспользоваться методом двухсвязного страничного представления [24]. Модель на основе этого метода для деков будет похожа на уже построенную модель для других структур данных [25].

В [26, 27] предлагались модели и методы оптимального управления двумя work-stealing деками в общей памяти одного уровня. Было рассмотрено использование распространенного метода раздельного последовательного циклического представления и применение нового метода, где деки двигаются друг за другом по кругу [28]. В [29] были предложены и проанализированы модели и методы оптимального управления тремя work-stealing деками в общей памяти одного уровня.

В двухуровневой памяти дек представлен таким образом: в быстрой памяти первого уровня расположены концы дека с которыми будут происходить операции; в медленной памяти второго уровня находится середина дека (рис. 3). В [30, 31] предлагались модели работы дека в двухуровневой памяти и решалась задача о поиске оптимального числа элементов концов дека, которые остаются в быстрой части после перераспределения двухуровневой памяти. Задача оптимального управления двумя work-stealing деками в двухуровневой памяти была исследована в [32].

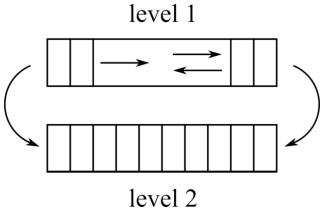


Рис. 3. Принцип работы work-stealing дека в двухуровневой памяти

В этой статье предлагается и исследуется новый метод представления трех work-stealing деков в двухуровневой памяти. В разделе 1 описан новый метод представления трех деков в двухуровневой памяти и сформулирована задача оптимального управления деками. Решение задачи и результаты исследований даны в разделе 2. В заключении предложен вариант использования этого метода и приведены некоторые направления дальнейших исследований.

1. Постановка задачи

Рассмотрим компьютерную систему, имеющую двухуровневую память. В медленной памяти второго уровня расположены середины трех деков (De_1, De_2, De_3), в которых хранятся указатели на задачи. В быстрой памяти первого уровня расположены концы (Sk_n, Qe_n) трех деков (De_n): Sk_n — рабочий конец дека De_n, в который происходят добавления и удаления указателей; Qe_n — work-stealing конец дека De_n, из которого посторонние (не рассматриваемые здесь) деки перехватывают указатели.

Память первого уровня размером m единиц разделена на две части s и $m - s$ (рис. 4): в части s расположены work-stealing концы всех трех деков (Qe_1, Qe_2, Qe_3), объединенные в одну очередь Qe и рабочий конец первого дека Sk_1, работающий как стек; в части $m - s$ расположены рабочие концы остальных двух деков (Sk_2, Sk_3), растущие навстречу друг другу.

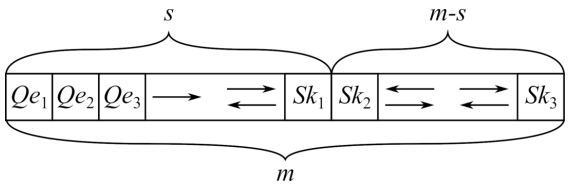


Рис. 4. Расположение деков в памяти первого уровня

Один указатель на задачу равен одной единице памяти. Пусть начальные размеры концов деков $Sk_1 = Qe_1 = Sk_2 = Qe_2 = Sk_3 = Qe_3 = 10$ указателей, общий размер памяти первого уровня $m = 100$ указателей. Таким образом, разделение s может принимать значения $s = 40 \dots 80$ единиц памяти. Начальные размеры концов деков и размер памяти являются теоретическими значениями, взятыми для упрощения дальнейших расчетов. В этой статье не рассматривается задача нахождения оптимальных начальных размеров. В табл. 1 приведены операции, происходящие с каждым деком на каждом шаге дискретного времени с заданной вероятностью. Сумма вероятностей возникновения операций для дека De_n: $p_n + q_n + w_n + pw_n + qw_n + r_n = 1$.

Таблица 1. Вероятности операций с деками

Обозначение	Описание вероятности
p_n	Добавление указателя в рабочий конец Sk_n дека De_n
q_n	Удаление указателя из рабочего конца Sk_n дека De_n
w_n	Перехват указателя из общей work-stealing очереди Qe, в то время как дек De_n обрабатывает задачу
pw_n	Параллельное добавление указателя в рабочий конец Sk_n дека De_n и перехват указателя из общей work-stealing очереди Qe
qw_n	Параллельное удаление указателя из рабочего конца Sk_n дека De_n и перехват указателя из общей work-stealing очереди Qe
r_n	Операция не изменяющая размеры концов дека De_n (например, обработка задачи)

Система работает бесконечно и прекращает свою работу (происходит перераспределение быстрой памяти) при опустошении рабочего конца любого дека $Sk_n < 0$, при опустошении общей work-stealing очереди Qe или при переполнении любой части быстрой памяти $Qe + Sk_1 > s$, $Sk_2 + Sk_3 > (m - s)$. В этой статье решается задача о нахождении оптимального значения разделения s быстрой памяти m для трех деков с заранее заданными вероятностями выполнения операций. Критерий оптимальности: максимальное среднее время работы системы (среднее количество операций) до перераспределения памяти.

2. Численный анализ

Для решения поставленной задачи была разработана имитационная модель процесса на основе метода Монте-Карло. Входными данными модели значатся: размер памяти первого уровня (m); часть памяти отведенная паре Qe—Sk_1 (s); вероятности возникновения операций с деками (p_n, q_n, w_n, pw_n, qw_n, r_n). Результатом имитационного моделирования является среднее время работы системы (t) до перераспределения памяти. С помощью имитационной модели был проведен ряд исследований. Взятые здесь вероятности операций являются теоретическими, для упрощения расчетов и повышения наглядности результатов.

В табл. 2 и 3 рассматривается ситуации, в которых указатели добавляются в один из деков чаще, чем в остальные деки ($p_1 > p_2$, $p_1 > p_3$ и $p_2 > p_1$, $p_2 > p_3$). В табл. 4 и 5 анализируются случаи, где указатели добавляются в два дека чаще, чем в оставшийся дек ($p_1 > p_3$, $p_2 > p_3$ и $p_2 > p_1$, $p_3 > p_1$). В табл. 6 рассматривается случай, где указатели добавляются во все деки одинаково часто ($p_1 = p_2 = p_3$).

Таблица 2. Среднее время работы системы, $p_1 > p_2$ и p_3

Вероятности операций	Метод разделения памяти	
	Разделение пополам	Оптимальное разделение
$p_1 = 0.50$, $p_2 = p_3 = 0.26$, $q_1 = 0.02$, $q_2 = q_3 = 0.26$, $r_1 = r_2 = r_3 = 0.45$	27.90	66.72 ($s = 69$)
$p_1 = 0.60$, $p_2 = p_3 = 0.31$, $q_1 = 0.02$, $q_2 = q_3 = 0.31$, $r_1 = r_2 = r_3 = 0.35$	22.24	54.09 ($s = 70$)
$p_1 = 0.70$, $p_2 = p_3 = 0.36$, $q_1 = 0.02$, $q_2 = q_3 = 0.36$, $r_1 = r_2 = r_3 = 0.25$	18.49	45.55 ($s = 70$)
$p_1 = 0.80$, $p_2 = p_3 = 0.41$, $q_1 = 0.02$, $q_2 = q_3 = 0.41$, $r_1 = r_2 = r_3 = 0.15$	15.82	39.34 ($s = 70$)
$p_1 = 0.90$, $p_2 = p_3 = 0.46$, $q_1 = 0.02$, $q_2 = q_3 = 0.46$, $r_1 = r_2 = r_3 = 0.05$	13.83	34.63 ($s = 70$)

Таблица 3. Среднее время работы системы, $p_2 > p_1$ и p_3

Вероятности операций	Метод разделения памяти	
	Разделение пополам	Оптимальное разделение
$p_2 = 0.50$, $p_1 = p_3 = 0.26$, $q_2 = 0.02$, $q_1 = q_3 = 0.26$, $r_1 = r_2 = r_3 = 0.45$	60.16	64.55 ($s = 46$)
$p_2 = 0.60$, $p_1 = p_3 = 0.31$, $q_2 = 0.02$, $q_1 = q_3 = 0.31$, $r_1 = r_2 = r_3 = 0.35$	49.94	53.29 ($s = 46$)
$p_2 = 0.70$, $p_1 = p_3 = 0.36$, $q_2 = 0.02$, $q_1 = q_3 = 0.36$, $r_1 = r_2 = r_3 = 0.25$	42.69	45.36 ($s = 46$)
$p_2 = 0.80$, $p_1 = p_3 = 0.41$, $q_2 = 0.02$, $q_1 = q_3 = 0.41$, $r_1 = r_2 = r_3 = 0.15$	37.29	39.48 ($s = 46$)
$p_2 = 0.90$, $p_1 = p_3 = 0.46$, $q_2 = 0.02$, $q_1 = q_3 = 0.46$, $r_1 = r_2 = r_3 = 0.05$	33.10	34.94 ($s = 46$)

Таблица 4. Среднее время работы системы, p_1 и $p_2 > p_3$

Вероятности операций	Метод разделения памяти	
	Разделение пополам	Оптимальное разделение
$p_1 = p_2 = 0.50$, $p_3 = 0.26$, $q_1 = q_2 = 0.02$, $q_3 = 0.26$, $r_1 = r_2 = r_3 = 0.45$	27.86	41.01 ($s = 58$)
$p_1 = p_2 = 0.60$, $p_3 = 0.31$, $q_1 = q_2 = 0.02$, $q_3 = 0.31$, $r_1 = r_2 = r_3 = 0.35$	22.24	33.73 ($s = 58$)
$p_1 = p_2 = 0.70$, $p_3 = 0.36$, $q_1 = q_2 = 0.02$, $q_3 = 0.36$, $r_1 = r_2 = r_3 = 0.25$	18.49	28.75 ($s = 59$)
$p_1 = p_2 = 0.80$, $p_3 = 0.41$, $q_1 = q_2 = 0.02$, $q_3 = 0.41$, $r_1 = r_2 = r_3 = 0.15$	15.83	25.12 ($s = 59$)
$p_1 = p_2 = 0.90$, $p_3 = 0.46$, $q_1 = q_2 = 0.02$, $q_3 = 0.46$, $r_1 = r_2 = r_3 = 0.05$	13.83	22.35 ($s = 59$)

Таблица 5. Среднее время работы системы, p_2 и $p_3 > p_1$

Вероятности операций	Метод разделения памяти	
	Разделение пополам	Оптимальное разделение
$p_2 = p_3 = 0.50$, $p_1 = 0.26$, $q_2 = q_3 = 0.02$, $q_1 = 0.26$, $r_1 = r_2 = r_3 = 0.45$	32.47	36.45 ($s = 45$)
$p_2 = p_3 = 0.60$, $p_1 = 0.31$, $q_2 = q_3 = 0.02$, $q_1 = 0.31$, $r_1 = r_2 = r_3 = 0.35$	26.92	30.13 ($s = 45$)
$p_2 = p_3 = 0.70$, $p_1 = 0.36$, $q_2 = q_3 = 0.02$, $q_1 = 0.36$, $r_1 = r_2 = r_3 = 0.25$	23.01	25.69 ($s = 45$)
$p_2 = p_3 = 0.80$, $p_1 = 0.41$, $q_2 = q_3 = 0.02$, $q_1 = 0.41$, $r_1 = r_2 = r_3 = 0.15$	20.09	22.39 ($s = 45$)
$p_2 = p_3 = 0.90$, $p_1 = 0.46$, $q_2 = q_3 = 0.02$, $q_1 = 0.46$, $r_1 = r_2 = r_3 = 0.05$	17.85	19.86 ($s = 45$)

Таблица 6. Среднее время работы системы, $p_1 = p_2 = p_3$

Вероятности операций	Метод разделения памяти	
	Разделение пополам	Оптимальное разделение
$p_1 = p_2 = p_3 = 0.50$, $q_1 = q_2 = q_3 = 0.02$, $r_1 = r_2 = r_3 = 0.45$	25.87	27.90 ($s = 53$)
$p_1 = p_2 = p_3 = 0.60$, $q_1 = q_2 = q_3 = 0.02$, $r_1 = r_2 = r_3 = 0.35$	21.14	23.17 ($s = 53$)
$p_1 = p_2 = p_3 = 0.70$, $q_1 = q_2 = q_3 = 0.02$, $r_1 = r_2 = r_3 = 0.25$	17.88	19.88 ($s = 53$)
$p_1 = p_2 = p_3 = 0.80$, $q_1 = q_2 = q_3 = 0.02$, $r_1 = r_2 = r_3 = 0.15$	15.49	17.49 ($s = 53$)
$p_1 = p_2 = p_3 = 0.90$, $q_1 = q_2 = q_3 = 0.02$, $r_1 = r_2 = r_3 = 0.05$	13.67	15.68 ($s = 53$)

На основе проведенных исследований можно сделать вывод, что среднее время работы системы возрастает, если делить память оптимально. Например, по результатам табл. 2 для общей памяти $m = 100$ единиц и вероятностях $p_1 = 0.50$, $p_2 = p_3 = 0.26$, оптимальные размеры памяти составляют: $s = 69$ единиц для пары Qe—Sk_1 и $m - s = 31$ единиц для пары Sk_2—Sk_3. При таком разделении памяти система в среднем совершает на 38.82 операций больше, чем при разделении памяти пополам. График зависимости среднего времени работы (t) от разделения памяти (s) приведен на рис. 5.

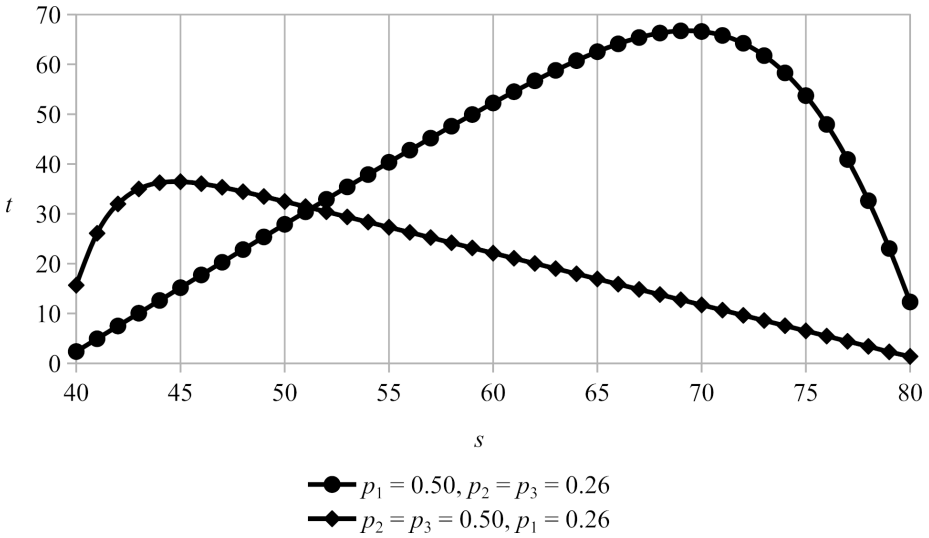


Рис. 5. Зависимость времени работы от разделения памяти

Аналогичная ситуация наблюдается и для остальных случаев. Так, анализируя табл. 5 можно увидеть, что оптимальные размеры памяти для вероятностей $p_2 = p_3 = 0.50$,

$p_1 = 0.26$ составляют: $s = 45$ единиц и $m - s = 55$ единиц. При оптимальном разделении памяти система в среднем совершает 36.45 операций, что на 3.98 операций больше, чем при разделении памяти пополам. График зависимости среднего времени работы (t) от разделения памяти (s) приведен на рис. 5.

Заключение

В статье решается задача о поиске оптимального значения разделения памяти первого уровня между тремя work-stealing деками, в качестве критерия оптимальности рассматривалось максимальное среднее время работы системы до перераспределения памяти. Имитационная модель этого процесса была построена на основе метода Монте-Карло и проанализирована.

Предложенную модель можно использовать для оптимизации двухуровневой памяти при разработке системного программного обеспечения. Для этого требуется провести предварительный статистический анализ работы системы и на основе полученных данных рассчитать значение оптимального разделения памяти.

В будущем можно рассмотреть другие критерии оптимальности, такие как минимизация суммы средних затрат или минимизация наибольших средних затрат на перераспределение быстрой памяти. Также можно обобщить эту задачу на произвольное число деков. В таком случае надо будет рассматривать разные варианты совместного расположения LIFO- и FIFO-частей деков в быстрой памяти.

Литература

1. Herlihy M., Shavit N. The Art of Multiprocessor Programming. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008. DOI: 10.5555/1734069.
2. Kwok Y.-K., Ahmad I. Static scheduling algorithms for allocating directed task graphs to multiprocessors // ACM Comput. Surv. New York, NY, USA, 1999. Dec. Vol. 31, no. 4. P. 406–471. DOI: 10.1145/344588.344618.
3. Alakeel A.M. A Guide to Dynamic Load Balancing in Distributed Computer Systems // International Journal of Computer Science and Network Security. 2010. Vol. 10, no. 6. P. 153–160.
4. Beaumont O., Carter L., Ferrante J., *et al.* Centralized versus distributed schedulers for multiple bag-of-task applications // Proceedings 20th IEEE International Parallel & Distributed Processing Symposium. 2006. P. 10. DOI: 10.1109/IPDPS.2006.1639262.
5. Xia Y., Prasanna V.K., Li J. Hierarchical Scheduling of DAG Structured Computations on Manycore Processors with Dynamic Thread Grouping // Job Scheduling Strategies for Parallel Processing / ed. by E. Frachtenberg, U. Schwiegelshohn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. P. 154–174. DOI: 10.1007/978-3-642-16505-4_9.
6. Hendler D., Shavit N. Non-blocking steal-half work queues // Proceedings of the Twenty-First Annual Symposium on Principles of Distributed Computing. Monterey, California: Association for Computing Machinery, 2002. P. 280–289. DOI: 10.1145/571825.571876.
7. Hendler D., Shavit N. Work dealing // Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures. Winnipeg, Manitoba, Canada: Association for Computing Machinery, 2002. P. 164–172. DOI: 10.1145/564870.564900.

8. Acar U.A., Charguéraud A., Rainey M. Scheduling parallel programs by work stealing with private deques // SIGPLAN Not. New York, NY, USA, 2013. Feb. Vol. 48, no. 8. P. 219–228. DOI: 10.1145/2517327.2442538.
9. Arora N.S., Blumofe R.D., Plaxton C.G. Thread scheduling for multiprogrammed multiprocessors // Proceedings of the Tenth Annual ACM Symposium on Parallel Algorithms and Architectures. Puerto Vallarta, Mexico: Association for Computing Machinery, 1998. P. 119–129. DOI: 10.1145/277651.277678.
10. Yang J., He Q. Scheduling Parallel Computations by Work Stealing: A Survey // International Journal of Parallel Programming. 2018. Apr. Vol. 46, no. 2. P. 173–197. DOI: 10.1007/s10766-016-0484-8.
11. Knuth D.E. The art of computer programming, volume 1 (3rd ed.): fundamental algorithms. USA: Addison Wesley Longman Publishing Co., Inc., 1997. DOI: 10.5555/260999.
12. Alam M., Varshney A.K. A New Approach of Dynamic Load Balancing Scheduling Algorithm for Homogeneous Multiprocessor System // International Journal of Applied Evolutionary Computation. 2016. Vol. 7, no. 2. P. 61–75. DOI: 10.4018/IJAEC.2016040104.
13. Амелина Н.О., Корнивец А.Д., Иванский Ю.В., Тюшев К.И. Применение консенсусного протокола для балансировки загрузки стохастической децентрализованной сети при передаче данных // XII Всероссийское совещание по проблемам управления: Труды научной конференции, Москва, 16–19 июня, 2014. Москва: ИПУ РАН, 2014. С. 8902–8911.
14. Amelina N., Fradkov A., Jiang Y., Vergados D.J. Approximate Consensus in Stochastic Networks with Application to Load Balancing // IEEE Transactions on Information Theory. 2015. Vol. 61, no. 4. P. 1739–1752. DOI: 10.1109/TIT.2015.2406323.
15. Li J., Agrawal K., Elnikety S., *et al.* Work stealing for interactive services to meet target latency // SIGPLAN Not. New York, NY, USA, 2016. Feb. Vol. 51, no. 8. Article 14. DOI: 10.1145/3016078.2851151.
16. Wimmer M., Versaci F., Traff J.L., *et al.* Data structures for task-based priority scheduling // SIGPLAN Not. New York, NY, USA, 2014. Feb. Vol. 49, no. 8. P. 379–380. DOI: 10.1145/2692916.2555278.
17. Gmys J., Leroy R., Mezmaz M., *et al.* Work stealing with private integer–vector–matrix data structure for multi-core branch-and-bound algorithms // Concurrency and Computation: Practice and Experience. 2016. Vol. 28, no. 18. P. 4463–4484. DOI: https://doi.org/10.1002/cpe.3771.
18. Mitzenmacher M. Analyses of load stealing models based on differential equations // Proceedings of the Tenth Annual ACM Symposium on Parallel Algorithms and Architectures. Puerto Vallarta, Mexico: Association for Computing Machinery, 1998. P. 212–221. DOI: 10.1145/277651.277687.
19. Kuchumov R., Sokolov A., Korkhov V. Staccato: Cache-Aware Work-Stealing Task Scheduler for Shared-Memory Systems // Computational Science and Its Applications – ICCSA 2018 / ed. by O. Gervasi, B. Murgante, S. Misra, *et al.* Cham: Springer International Publishing, 2018. P. 91–102. DOI: 10.1007/978-3-319-95171-3_8.
20. Kuchumov R., Sokolov A., Korkhov V. Staccato: shared-memory work-stealing task scheduler with cache-aware memory management // International Journal of Web and Grid Services. 2019. Vol. 15, no. 4. P. 394–407. DOI: 10.1504/IJWGS.2019.103233.

21. Аксенова Е.А., Соколов А.В. Методы управления work-stealing деками в динамических планировщиках многопроцессорных параллельных вычислений // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2023. Т. 12, № 4. С. 76–93. DOI: 10.14529/cmse230403.

22. Chase D., Lev Y. Dynamic circular work-stealing deque // Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures. Las Vegas, Nevada, USA: Association for Computing Machinery, 2005. P. 21–28. DOI: 10.1145/1073970.1073974.

23. Sokolov A., Drac A. The linked list representation of n LIFO-stacks and/or FIFO-queues in the single-level memory // Information Processing Letters. 2013. Vol. 113, no. 19. P. 832–835. DOI: 10.1016/j.ipl.2013.07.021.

24. Hendler D., Lev Y., Moir M., Shavit N. A dynamic-sized nonblocking work stealing deque // Distributed Computing. 2006. Feb. Vol. 18, no. 3. P. 189–207. DOI: 10.1007/s00446-005-0144-5.

25. Аксенова Е.А., Лазутина А.А., Соколов А.В. Об оптимальных методах представления динамических структур данных // Обозрение прикладной и промышленной математики. 2003. Т. 10, № 2. С. 375–376.

26. Sokolov A., Barkovsky E. The Mathematical Model and the Problem of Optimal Partitioning of Shared Memory for Work-Stealing Deques // Parallel Computing Technologies / ed. by V. Malyshkin. Cham: Springer International Publishing, 2015. P. 102–106. DOI: 10.1007/978-3-319-21909-7_11.

27. Барковский Е.А., Кучумов Р.И., Соколов А.В. Оптимальное управление двумя work-stealing деками в общей памяти при различных стратегиях перехвата работы // Программные системы: теория и приложения. 2017. Т. 8, № 1. С. 83–103. DOI: 10.25209/2079-3316-2017-8-1-83-103.

28. Барковский Е.А., Лазутина А.А., Соколов А.В. Построение и анализ модели процесса работы с двумя деками, двигающимися друг за другом в общей памяти // Программные системы: теория и приложения. 2019. Т. 10, № 1. С. 3–17. DOI: 10.25209/2079-3316-2019-10-1-3-17.

29. Aksenova E.A., Barkovsky E.A., Sokolov A.V. The Models and Methods of Optimal Control of Three Work-Stealing Deques Located in a Shared Memory // Lobachevskii Journal of Mathematics. 2019. Nov. Vol. 40, no. 11. P. 1763–1770. DOI: 10.1134/S1995080219110052.

30. Лазутина А.А., Соколов А.В. Об оптимальном управлении Work-Stealing деками в двухуровневой памяти // Вестник компьютерных и информационных технологий. 2020. Т. 17, № 4. С. 51–60. DOI: 10.14489/vkit.2020.04.pp.051-060.

31. Аксенова Е.А., Лазутина А.А., Соколов А.В. Минимизация средних затрат на перераспределение при работе с work-stealing деком в двухуровневой памяти // Программные системы: теория и приложения. 2021. Т. 12, № 2. С. 53–71. DOI: 10.25209/2079-3316-2021-12-2-53-71.

32. Aksenova E.A., Lazutina A.A., Sokolov A.V. About Optimal Management of Work-Stealing Deques in Two-Level Memory // Lobachevskii Journal of Mathematics. 2021. July. Vol. 42, no. 7. P. 1475–1482. DOI: 10.1134/S1995080221070027.

Аксёнова Елена Алексеевна, к.ф.-м.н., лаборатория информационных компьютерных технологий, Институт прикладных математических исследований Карельского научного центра Российской академии наук (Петрозаводск, Российская Федерация)

Барковский Евгений Александрович, независимый исследователь (Петрозаводск, Российская Федерация)

Соколов Андрей Владимирович, д.ф.-м.н., профессор, лаборатория информационных компьютерных технологий, Институт прикладных математических исследований Карельского научного центра Российской академии наук (Петрозаводск, Российская Федерация)

DOI: 10.14529/cmse240303

OPTIMAL CONTROL OF THREE WORK-STEALING DEQUES LOCATED IN TWO-LEVEL MEMORY

© 2024 E.A. Aksenova, E.A. Barkovsky, A.V. Sokolov

Institute of Applied Mathematical Research
of the Karelian Research Centre of the Russian Academy of Sciences
(str. Pushkinskaya 11, Petrozavodsk, 185910 Russia)
E-mail: aksenova@krc.karelia.ru, barkevgen@gmail.com, avs@krc.karelia.ru

Received: 26.07.2024

The problem of even balancing of tasks between threads may arise during parallel computations. One way to resolve this issue is to implement distributed dynamic load balancing. In this balancing method each worker thread has its own task queue, and threads themselves distribute further tasks. One of the widely used distributed dynamic balancing methods is "work-stealing": a thread that runs out of tasks begins to "steal" them from another thread. To utilize this method, each thread must have its own specialized deque where pointers to tasks are stored. In this paper, we propose and examine a new method for representing three work stealing deques in two-level memory. We consider the case of working with three deques, where one section of fast memory contains two LIFO parts of the deques, i.e., two stacks that grow towards each other. Third LIFO part and three FIFO parts are located in the other section of fast memory. FIFO parts are combined into one FIFO queue, from which elements are only deleted (being stolen). The middle parts of the deques are located in slow memory. They are accessed when LIFO or FIFO parts located in fast memory are empty or overflowed. We consider the problem of finding optimal partition of fast memory for three deques with predetermined probabilities of operations. This choice depends on the characteristics of memory levels, operation probabilities, and optimality criteria. The optimality criterion is the maximum mean system operating time (the average number of operations) before memory reallocation.

Keywords: data structures, Monte Carlo methods, two-level memory, work stealing deques.

FOR CITATION

Aksenova E.A., Barkovsky E.A., Sokolov A.V. Optimal Control of Three Work-Stealing Deques Located in Two-Level Memory. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2024. Vol. 13, no. 3. P. 47–60. (in Russian) DOI: 10.14529/cmse240303.

References

1. Herlihy M., Shavit N. The Art of Multiprocessor Programming. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008. DOI: 10.5555/1734069.
2. Kwok Y.-K., Ahmad I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv. New York, NY, USA, 1999. Dec. Vol. 31, no. 4. P. 406–471. DOI: 10.1145/344588.344618.
3. Alakeel A.M. A Guide to Dynamic Load Balancing in Distributed Computer Systems. International Journal of Computer Science and Network Security. 2010. Vol. 10, no. 6. P. 153–160.
4. Beaumont O., Carter L., Ferrante J., *et al.* Centralized versus distributed schedulers for multiple bag-of-task applications. Proceedings 20th IEEE International Parallel & Distributed Processing Symposium. 2006. P. 10. DOI: 10.1109/IPDPS.2006.1639262.
5. Xia Y., Prasanna V.K., Li J. Hierarchical Scheduling of DAG Structured Computations on Manycore Processors with Dynamic Thread Grouping. Job Scheduling Strategies for Parallel Processing / ed. by E. Frachtenberg, U. Schwiegelshohn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. P. 154–174. DOI: 10.1007/978-3-642-16505-4_9.
6. Hendler D., Shavit N. Non-blocking steal-half work queues. Proceedings of the Twenty-First Annual Symposium on Principles of Distributed Computing. Monterey, California: Association for Computing Machinery, 2002. P. 280–289. DOI: 10.1145/571825.571876.
7. Hendler D., Shavit N. Work dealing. Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures. Winnipeg, Manitoba, Canada: Association for Computing Machinery, 2002. P. 164–172. DOI: 10.1145/564870.564900.
8. Acar U.A., Charguéraud A., Rainey M. Scheduling parallel programs by work stealing with private deques. SIGPLAN Not. New York, NY, USA, 2013. Feb. Vol. 48, no. 8. P. 219–228. DOI: 10.1145/2517327.2442538.
9. Arora N.S., Blumofe R.D., Plaxton C.G. Thread scheduling for multiprogrammed multiprocessors. Proceedings of the Tenth Annual ACM Symposium on Parallel Algorithms and Architectures. Puerto Vallarta, Mexico: Association for Computing Machinery, 1998. P. 119–129. DOI: 10.1145/277651.277678.
10. Yang J., He Q. Scheduling Parallel Computations by Work Stealing: A Survey. International Journal of Parallel Programming. 2018. Apr. Vol. 46, no. 2. P. 173–197. DOI: 10.1007/s10766-016-0484-8.
11. Knuth D.E. The art of computer programming, volume 1 (3rd ed.): fundamental algorithms. USA: Addison Wesley Longman Publishing Co., Inc., 1997. DOI: 10.5555/260999.
12. Alam M., Varshney A.K. A New Approach of Dynamic Load Balancing Scheduling Algorithm for Homogeneous Multiprocessor System. International Journal of Applied Evolutionary Computation. 2016. Vol. 7, no. 2. P. 61–75. DOI: 10.4018/IJAEC.2016040104.
13. Amelina N.O., Kornivec A.D., Ivanskij J.V., Tjushev K.I. Primenenie konsensusnogo protokola dlja balansirovki zagruzki stohasticheskoj decentralizovannoj seti pri peredache dannyh. XII Vserossijskoe soveshhanie po problemam upravlenija: Scientific conference proceedings, Moscow, 16–19 June, 2014. Moscow: ICS of RAS, 2014. P. 8902–8911. (in Russian).

14. Amelina N., Fradkov A., Jiang Y., Vergados D.J. Approximate Consensus in Stochastic Networks with Application to Load Balancing. IEEE Transactions on Information Theory. 2015. Vol. 61, no. 4. P. 1739–1752. DOI: 10.1109/TIT.2015.2406323.

15. Li J., Agrawal K., Elnikety S., *et al.* Work stealing for interactive services to meet target latency. SIGPLAN Not. New York, NY, USA, 2016. Feb. Vol. 51, no. 8. Article 14. DOI: 10.1145/3016078.2851151.

16. Wimmer M., Versaci F., Traff J.L., *et al.* Data structures for task-based priority scheduling. SIGPLAN Not. New York, NY, USA, 2014. Feb. Vol. 49, no. 8. P. 379–380. DOI: 10.1145/2692916.2555278.

17. Gmys J., Leroy R., Mezmaz M., *et al.* Work stealing with private integer–vector–matrix data structure for multi-core branch-and-bound algorithms. Concurrency and Computation: Practice and Experience. 2016. Vol. 28, no. 18. P. 4463–4484. DOI: https://doi.org/10.1002/cpe.3771.

18. Mitzenmacher M. Analyses of load stealing models based on differential equations. Proceedings of the Tenth Annual ACM Symposium on Parallel Algorithms and Architectures. Puerto Vallarta, Mexico: Association for Computing Machinery, 1998. P. 212–221. DOI: 10.1145/277651.277687.

19. Kuchumov R., Sokolov A., Korkhov V. Staccato: Cache-Aware Work-Stealing Task Scheduler for Shared-Memory Systems. Computational Science and Its Applications – ICCSA 2018 / ed. by O. Gervasi, B. Murgante, S. Misra, *et al.* Cham: Springer International Publishing, 2018. P. 91–102. DOI: 10.1007/978-3-319-95171-3_8.

20. Kuchumov R., Sokolov A., Korkhov V. Staccato: shared-memory work-stealing task scheduler with cache-aware memory management. International Journal of Web and Grid Services. 2019. Vol. 15, no. 4. P. 394–407. DOI: 10.1504/IJWGS.2019.103233.

21. Aksenova E.A., Sokolov A.V. Control Methods of Work-stealing Deques in Dynamic Schedulers of Multiprocessor Parallel Computations. Bulletin of the SUSU. Series: Computational Mathematics and Software Engineering. 2023. Vol. 12, no. 4. P. 76–93. (in Russian) DOI: 10.14529/cmse230403.

22. Chase D., Lev Y. Dynamic circular work-stealing deque. Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures. Las Vegas, Nevada, USA: Association for Computing Machinery, 2005. P. 21–28. DOI: 10.1145/1073970.1073974.

23. Sokolov A., Drac A. The linked list representation of n LIFO-stacks and/or FIFO-queues in the single-level memory. Information Processing Letters. 2013. Vol. 113, no. 19. P. 832–835. DOI: 10.1016/j.ipl.2013.07.021.

24. Hendler D., Lev Y., Moir M., Shavit N. A dynamic-sized nonblocking work stealing deque. Distributed Computing. 2006. Feb. Vol. 18, no. 3. P. 189–207. DOI: 10.1007/s00446-005-0144-5.

25. Aksenova E.A., Lazutina A.A., Sokolov A.V. Ob optimal'nyh metodah predstavlenija dinamicheskih struktur dannyh. Obozrenie prikladnoj i promyshlennoj matematiki. 2003. Vol. 10, no. 2. P. 375–376. (in Russian).

26. Sokolov A., Barkovsky E. The Mathematical Model and the Problem of Optimal Partitioning of Shared Memory for Work-Stealing Deques. Parallel Computing Technologies / ed. by V. Malyshkin. Cham: Springer International Publishing, 2015. P. 102–106. DOI: 10.1007/978-3-319-21909-7_11.

27. Barkovksy E., Kuchumov R., Sokolov A. Optimal Control of Two Deques in Shared Memory with Various Work-Stealing Strategies. Program Systems: Theory and Applications. 2017. Vol. 8, no. 1. P. 83–103. (in Russian) DOI: 10.25209/2079-3316-2017-8-1-83-103.

28. Barkovsky E., Lazutina A., Sokolov A. The Optimal Control of Two Work-Stealing Deques, Moving One After Another in a Shared Memory. Program Systems: Theory and Applications. 2019. Vol. 10, no. 1. P. 19–32. DOI: 10.25209/2079-3316-2019-10-1-19-32.

29. Aksenova E.A., Barkovsky E.A., Sokolov A.V. The Models and Methods of Optimal Control of Three Work-Stealing Deques Located in a Shared Memory. Lobachevskii Journal of Mathematics. 2019. Nov. Vol. 40, no. 11. P. 1763–1770. DOI: 10.1134/S1995080219110052.

30. Lazutina A.A., Sokolov A.V. About Optimal Management of Work-Stealing Deques in Two-Level Memory. Herald of Computer and Information Technologies. 2020. Vol. 17, no. 4. P. 51–60. (in Russian) DOI: 10.14489/vkit.2020.04.pp.051-060.

31. Aksenova E.A., Lazutina A.A., Sokolov A.V. About Optimal Managment of Work-Stealing Deques in Two-Level Memory. Program Systems: Theory and Applications. 2021. Vol. 12, no. 2. P. 53–71. (in Russian) DOI: 10.25209/2079-3316-2021-12-2-53-71.

32. Aksenova E.A., Lazutina A.A., Sokolov A.V. About Optimal Management of Work-Stealing Deques in Two-Level Memory. Lobachevskii Journal of Mathematics. 2021. July. Vol. 42, no. 7. P. 1475–1482. DOI: 10.1134/S1995080221070027.

DOI: 10.14529/cmse240304

TASC SOFTWARE FOR HPC PERFORMANCE ANALYSIS: CURRENT STATE AND LATEST DEVELOPMENTS

© 2024 V.V. Voevodin, D.I. Shaikhislamov, V.A. Serov

Lomonosov Moscow State University

(Leninskie Gory 1, Moscow, 119991 Russia)

E-mail: vadim@parallel.ru, sdenis1995@gmail.com, sva@srcc.msu.ru

Received: 04.09.2024

To ensure high operating efficiency of modern supercomputers, it is necessary to constantly analyze and control various aspects of their behavior, paying special attention to the flow of supercomputer applications running on these machines. To solve this problem, the TASC (Tuning Applications for SuperComputers) software suite was previously developed. It automatically detects performance issues in HPC applications and evaluates the efficiency of using supercomputer resources, provides supercomputer administrators with a flexible report tool for analyzing different aspects of supercomputer functioning with the desired level of detail, and estimates the noise level on compute nodes. This paper provides full-scale description of current TASC structure and capabilities, including the stages of data processing and storing, as well as performing different types of analysis. It also describes new results obtained and methods developed within one of the main TASC components — assessment system for quick and accurate evaluation of HPC resources usage efficiency.

Keywords: high-performance computing, supercomputer, performance analysis, workload analysis, operational data analytics, monitoring.

FOR CITATION

Voevodin V.V., Shaikhislamov D.I., Serov V.A. TASC Software for HPC Performance Analysis: Current State and Latest Developments. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2024. Vol. 13, no. 3. P. 61–78. DOI: 10.14529/cmse240304.

Introduction

Supercomputers are used to substantially increase the speed of performing computational experiments, which are needed in a wide variety of different scientific and commercial researches in any major subject area. This means that the speed of supercomputer computations is one of the key characteristics, and the task of increasing it is among the most important ones. However, it turns out in practice that the performance of many supercomputer applications is quite low [1], since they efficiently utilize only a fraction of resources provided to them. This is due to both the extreme complexity of the structure of modern supercomputers (which leads to the fact that correctly taking into account all the hardware peculiarities to achieve maximum performance has become a very difficult task), and the complexity of writing high-performance parallel applications in general. And this becomes more and more topical, since the HPC market is growing [2, 3].

The problem is complicated by the fact that many users are unaware that their applications are not working efficiently. And performance issues can arise for various reasons: inefficient implementation of the application itself; poor matching of the selected software implementation and the target hardware; external factors caused by the surrounding supercomputer environment, and many others. One possible way to solve this problem of unawareness about inefficient use of supercomputer resources is to constantly analyze the flow of running applications, notifying

both the users about their inefficient applications and the supercomputer administrators about the situation in general.

With this in mind, the TASC (Tuning Applications for SuperComputers) software suite was previously developed at the Research Computing Center of Lomonosov Moscow State University. This suite includes several software components that use different approaches for solving the stated task. A tool for automatic detection of job-level performance issues [4] allows to promptly notify users about potential performance issues in their applications, which is done based on a set of predefined rules. An assessment system [1] is designed to automatically provide a quick and accurate evaluation of the efficiency of using supercomputer resources. An "endless" web report system [5] provides administrators with a very flexible set of performance-related dashboards that allows them to analyze only the required data subset with desired level of detail. And a noise level estimation system [6] helps to understand to what extent system processes (which are always running in the background on compute nodes) actively interfere with the execution of user applications. In this case, "noise" is defined as an external influence of the software and hardware environment leading to a change in the execution time or other properties of user applications running on the supercomputer, and one of the most common noise sources is operating system processes.

This paper provides for the first time a full-scale description of the current state of the TASC software suite, as well as presents the latest methods developed within it and results obtained using this solution.

The question of the efficiency analysis of applications running on supercomputers is touched upon in several existing studies. There are a number of papers like [7–9] presenting holistic reports on the operation of specific HPC systems. In these papers, many major aspects of supercomputer functioning are considered: job launch properties (duration, number of allocated processors, used software libraries, etc.); job efficiency in terms of CPU or GPU load, intensity of communication network usage; distribution of job, node-hours or efficiency characteristics between partitions or subject areas; and many others. However, these were mostly manually performed studies, therefore not portable and not directly applicable in other supercomputer centers. There are also works that study the efficiency of individual aspects of supercomputer behavior in more detail. For example, the authors of [10] in detail discuss how the job properties and trends in resource consumption of two large supercomputers, Intrepid and Mira, have changed over several years of operation. In paper [11], authors present a software solution for automatic detection of problems with job efficiency and behavior. Another paper [12] discusses the causes of low I/O throughput and provides a dedicated software solution to detect them. There are other similar works as well, but the difference in TASC is that it aims at holistic automated analysis of many different aspects, also focusing on thorough analysis of monitoring data.

The main paper is organized as follows. Section 1 shows the current TASC architecture and capabilities, listing the input data being used and describing approaches to storing this data (subsection 1.1), as well as showing main tools for performance analysis (subsection 1.2). Section 2 is devoted to the description of new methods, tools and results that have been obtained recently, which are all related to the assessment system. In subsection 2.1, we describe new visualization tools for the assessment system, new methods for informing users about the results obtained are described in 2.2, and in 2.3 we describe the data mining method for continuous prediction of assessments. We show interesting examples of using this system in section 3. Conclusions summarize the work done.

1. TASC Current State

TASC software suite is intended for carrying out ODA (Operational Data Analytics, e.g., see [13, 14]) on a supercomputer. It performs continuous performance analysis and visualization of data on behavior of supercomputer in general and applications running on in particular. In this section, we will describe TASC architecture and functionality, using its existing implementation running on the Lomonosov-2 supercomputer [15] as an example.

The overall working algorithm of TASC software suite is quite straightforward. Most input data needed for performance analysis carried out by TASC is collected using monitoring systems running on compute and service nodes of a supercomputer. The data is sent via network to the dedicated virtual machine running main TASC services, where it is processed and stored into databases. Analysis tools are launched on this virtual machine at the selected frequency, retrieving data from databases, studying it and storing results back to databases. The obtained insights are available to supercomputer users and administrators via websites and notifications using email or messengers. Thus, two main stages of TASC operation can be distinguished: 1) processing and storing the required input data, and 2) subsequent analysis of this data. We will describe these stages in more detail below.

1.1. Processing and Storing Data for Subsequent Analysis

TASC does not directly collect raw data (monitoring systems are mainly used for this purpose), so this process is not described here. However, the variety of collected input data is of interest. Figure 1 lists the types and sources of data that are being regularly collected on the Lomonosov-2 supercomputer. Each grey rectangle corresponds to a specific target (hardware or software) being monitored, a yellow rounded rectangle — to a specific tool and the data it collects. Within the rounded rectangle, it is shown what tool is used for data collection (top right), what types of data is collected (center) and what the granularity of data collection is (bottom right).

The main source of performance data is compute nodes, and we use DiMMon monitoring system [16] developed in RCC MSU to collect it. This data is collected once per second, then it is aggregated and sent to TASC once per minute. For each job running on Lomonosov-2, TASC gets the following information:

- CPU load and load average;
- intensity of sent/received bytes and packets via MPI network;
- GPU load and memory usage;
- Lustre file system usage (amount of read/write bytes and opened/closed files);
- performance monitoring counters describing CPU usage activity and cache misses;
- amount of available RAM;
- existence of ECC errors.

On compute nodes, we additionally collect information about noise level, which helps us to detect cases when noise becomes notable. For this purpose, a special script was implemented, which regularly runs quick noise measurement in the epilogue of user jobs (see [6] for more details).

There is no need to study service nodes in such detail, however, it is also useful to collect information about them, especially regarding their reliability. That is why we obtain such basic data as server availability using ping command, load average and available disk space. On head

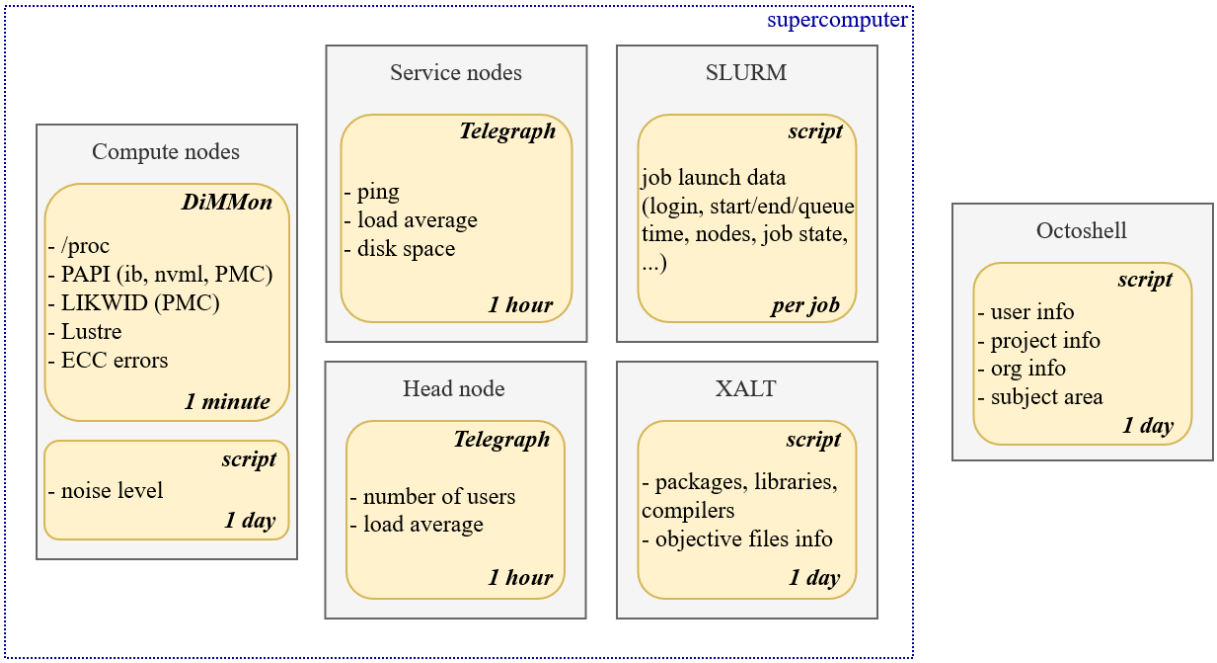


Figure 1. Monitoring data collected on the Lomonosov-2 supercomputer

node, we are interested in the number of active users currently working on Lomonosov-2, as well as in load average.

In order to get a detailed picture of supercomputer behavior, information from compute and service nodes alone is not enough. We additionally collect data from different system software. Of course, SLURM is used to get information about launches of user jobs — who, when and where launched each job, how many nodes were allocated, what the job exit state was, etc. We also use XALT [17] to get useful information about popular software packages (like NAMD, GROMACS, etc.) and compilers being used in jobs.

There is one source of data for TASC that does not run on the supercomputer itself. We regularly request Octoshell [18] (supercomputer center management system developed in RCC MSU) located on a separate server to obtain information about users as well as scientific projects and organizations they work in.

All this variety of data needs to be properly stored, so that TASC analysis modules can easily and quickly retrieve all the information they need. The overall data storage organization scheme is shown in Fig. 2. The main database used to store almost all types of data is a MongoDB database. This DB type was initially chosen because we work with very diverse data, and we also need to be able to easily add new data sources. For this purpose also, we have developed our own unified API, which significantly simplifies this process of importing different types of data in our MongoDB. All the data in this database is never deleted, enabling us to work with all the historical information we want.

The only exception is the main performance-related data from compute nodes. The reason why it is imported differently is the volume of this data. More than 40 characteristics are collected on each compute node using DiMMon, and for each of them minimum, average and maximum value is sent every minute. Taking into account that the number of nodes in the Lomonosov-2 supercomputer is more than 1500, and also considering our desire to store this information for a long time, it is obvious that the main database will very quickly become too large and slow. That

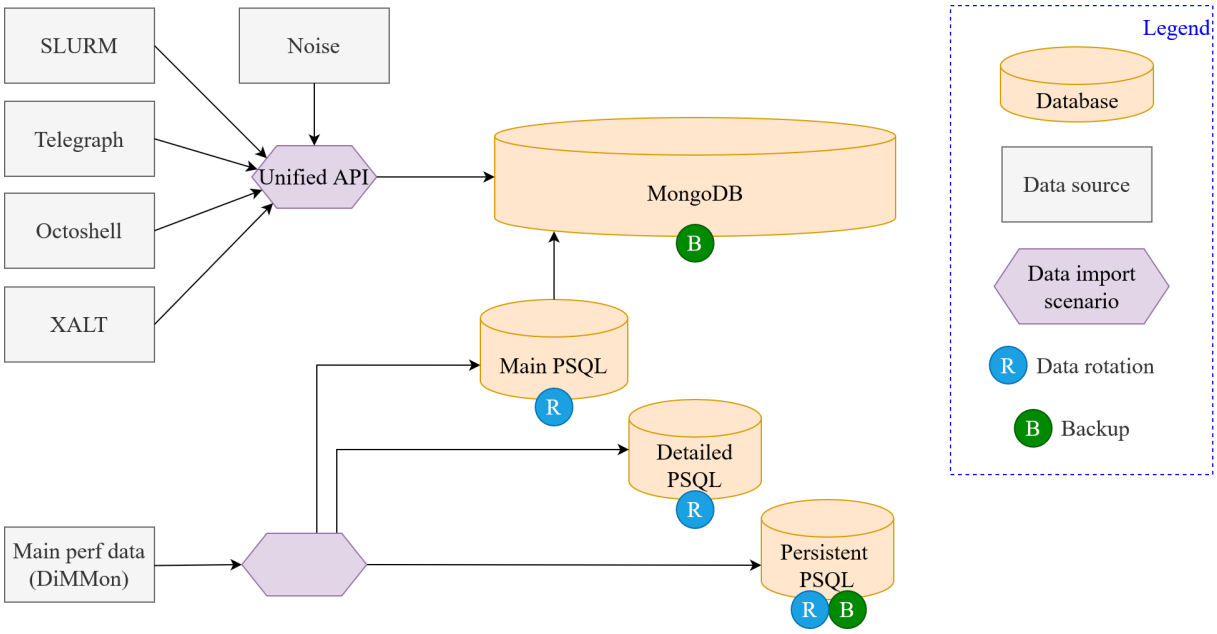


Figure 2. Data storage organization in TASC

is why we decided to use the following approach. All DiMMon data with 1-minute granularity is stored to PostgreSQL database called Main PSQL. This data is regularly aggregated per job and sent to MongoDB. Since this database is used frequently and queries to it must be executed quickly, it can not be very large, so it rotates the data and stores data for the last ∼3 weeks. But we want the data not to disappear so quickly, so it was decided to add two more databases — Detailed and Persistent PSQL. The first one stores the same data, but for a much longer time period (∼5 months). It is rarely used, usually to perform heavy queries involving large amounts of data. The Persistent PSQL stores data almost endlessly (for 3 years now, planned to make it even longer), but since the database would become unacceptably large over such a period of time, the data granularity was reduced from 1 minute to 10 minutes.

The described scheme has proven itself well in practice, allowing to easily deal with large amounts of real-life data.

1.2. Performing Data Analysis

The part of TASC responsible for performance analysis consists of several generally independent solutions, each of which is aimed at carrying out ODA for studying different aspects of supercomputer behavior. The main analysis software tools implemented and used in practice are:

- tool for automatic detection of job-level performance issues;
- web report system of “endless” workload analysis;
- assessment system for evaluating HPC resource efficiency;
- noise measurement tool.

The first solution automatically detects performance issues in all user jobs running on the supercomputer. This is performed using a set of predefined rules; each rule describes one particular issue with job performance and includes issue description, criteria for its automatic detection using monitoring data, supposition of its root cause as well as recommendation on possible further steps for its elimination. On the Lomonosov-2 supercomputer, there are currently ∼30 rules

implemented that are aimed at capturing MPI usage inefficiency, imbalance, usage of wrong partitions, abnormally inefficient jobs, etc. The examples and more detailed description of these rules can be found in [4].

"Endless" workload analysis [5] is a web-based system for HPC system administrators that can provide performance-related reports with a desired level of detail. This flexible tool can show only the needed subset of data for a specified level of consideration, starting from particular job launches and to the HPC system as a whole. The ability to fine-tune both the subset of data being considered and the object of consideration (job launch, user, project, organization, software package, scientific area) generates a tremendous number of possible report variants, which gives this system its "endless" name.

One way to assess the performance of an HPC application is to determine how efficiently it uses resources allocated to it. However, commonly used metrics, which are collected on-the-fly for all applications running on the supercomputer (i.e., without the need to restart the application in profiling, tracing, or debugging mode), usually do not allow estimating this with sufficient accuracy. For example, CPU user load shows processor utilization, but it does not indicate if CPU is busy with useful calculations. Further, common memory-related characteristics like the frequency of cache misses or read/write operations also do not allow one to accurately assess whether memory is being used efficiently. In order to solve this problem, a set of assessments was proposed [1], which automatically calculate accurate measurements of resource usage efficiency (CPU, memory, MPI network, I/O, GPU) for all jobs being executed on the supercomputer. The main idea is to estimate to what extent dealing with a specific resource interferes with useful computations. This allows both identifying applications that show low efficiency and determining directions for further performance analysis using existing profilers, debuggers, etc.

Most modern supercomputers contain so-called noise on compute nodes, which was previously defined as an external influence of the software and hardware environment leading to a change in the behavior of user applications. It is usually quite insignificant, but in some cases it can notably affect the behavior of user applications. In order to assess the noise level, a tool within TASC was proposed and developed [6] that allows regular estimation of the noise level on compute nodes. This tool is based on Netgauge software [19, 20] aimed at measuring OS noise. It starts in the SLURM epilogue after each job completion (but no more than once a day, so as not to impact user applications itself) and provides a noise level metric ranging from 0 (no noise at all) to 100 (maximum possible noise level). This helps, for example, to identify dysfunctional nodes and send them to system administrators for further detailed study and repair.

2. TASC Recent Developments

The main recent developments within TASC are aimed at enhancing one analysis component — the assessment system. In this section, we will describe them as well as provide real-life examples of useful insights obtained using this system.

As mentioned earlier, the assessment system allows for automatic estimation of the efficiency of using various resource types for all jobs running on the supercomputer. Previously, an approach for creating such assessment system was proposed [1], as well as a software implementation of the assessments and its evaluation in practice were performed. This system provides assessment scores for 4 main types of resources (GPU scores were proposed but not implemented due to limitations of currently available hardware): CPU, memory subsystem, MPI network, I/O. Each score estimates resource usage efficiency ranging from 0 (working with this resource did not

hinder useful computations in any way during job execution) to 100 (it constantly hindered useful computations during job execution).

This section describes further work in this direction. An approach to visualizing the collected information on assessments was developed, a method for informing supercomputer users about detected efficiency problems was proposed, and an analysis of the collected statistics on the assessments obtained on the Lomonosov-2 supercomputer was conducted. Also, we implemented a solution for continuous prediction of assessment values in cases when the main method of their calculation is not applicable.

2.1. Dashboards for Assessment Results Visualization

The main goal of this service is to provide a convenient web tool for studying all the main results obtained by the assessment system. It should be noted that it is only available to supercomputer administrators; users receive results from the assessment system in a much smaller volume and in a simpler and more convenient way (see subsection 2.2). This is done, in particular, because excessively detailed information is not required by users and will only confuse them; moreover, users should not have access to information about the performance of other users and projects.

To provide detailed information about the quality of supercomputer resources usage, this web service uses so-called dashboards — web reports dedicated to a certain part of results available in the assessment system. In total, three main dashboards are implemented: 1) general information about resource usage efficiency; 2) detailed information about particular user; 3) detailed information about particular scientific project. Each dashboard can be flexibly configured using different filters, such as specific time period, user, project, organization, used software package, supercomputer partition, number of nodes or node-hours, presence of particular performance issues automatically detected by rules in TASC (briefly described in subsection 1.2). It should be noted that all figures shown below in the paper are screenshots obtained from these dashboards.

The general information dashboard is designed to provide primary insights about what is happening on the supercomputer in general. The dashboard provides the following information:

- integral values of different assessment scores;
- dynamics of change in score values over time;
- distribution of score values for different jobs;
- top worst values for particular jobs, users and projects.

For example, Fig. 3 shows the distribution of $\text{score}_{\text{cpu}}$ (axis X) and $\text{score}_{\text{mem}}$ (axis Y) for individual jobs, where each point corresponds to one job and its color — to different users. Consider, for example, the point in the upper left corner, highlighted by the green circle in the figure. This job has a $\text{score}_{\text{cpu}}$ of 45, which is generally within the normal range, and a $\text{score}_{\text{mem}}$ of 80, which is much higher (i.e., worse) than normal. In this case, we can conclude that the processor often waits for data from memory during the execution of this application, though this is not a significant bottleneck, since the share of useful calculations is still high. The situation is quite different for the application highlighted by the red circle: here, the $\text{score}_{\text{cpu}}$ is above 90, and the $\text{score}_{\text{mem}}$ is slightly above 40. Such a high $\text{score}_{\text{cpu}}$ value indicates serious performance problems, and memory usage is likely one of the reasons for this (although clearly not the only one, since the $\text{score}_{\text{mem}}$ value is not so high). This job definitely requires a more detailed study, as can be concluded based on this graph.

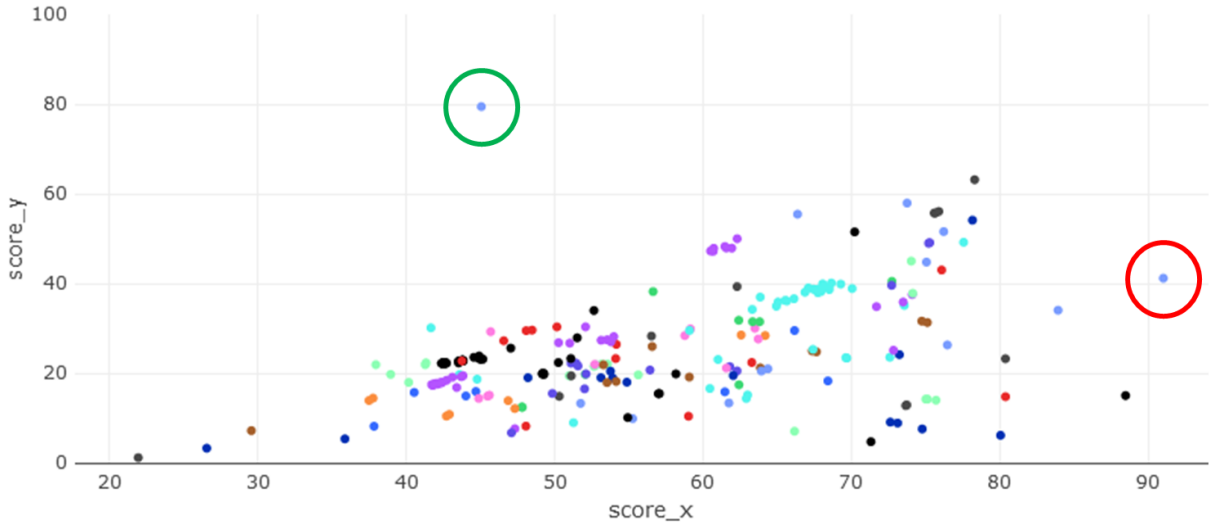


Figure 3. 2D graph showing distribution of values for $\text{score}_{\text{cpu}}$ and $\text{score}_{\text{mem}}$ for individual jobs (screenshot from web dashboard)

The dashboard with detailed information about particular user shows the following information:

- overall user activity: number of job launches and consumed node-hours for the selected time period;
- aggregated score values for the selected period, as well as user rank according to these scores (in general and for the selected software package);
- distribution of score values, separately for each resource type;
- detailed information on each job;
- the same information as for the first dashboard (but filtered for this user only): dynamics of change in score values over time and distribution of score values for different jobs.

As an example, Fig. 4 shows aggregated score values with user ranks (the lower the ranking, the worse). This figure shows that the user has no problems with I/O (the average score is 0, so no I/O-related issues at all, which obviously corresponds to the 1st place in the rating), but the share of useful calculations is not so great — $\text{score}_{\text{cpu}}$ is 67, and this is the 91st place out of 115 (this is the total number of users who launched their tasks during the time period under review). At the same time, note that the average $\text{score}_{\text{mem}}$ value is less than 40, which is generally considered a normal value for supercomputer applications. But the low rank, according to this assessment, suggests that memory usage can be probably optimized, since most of other users show better memory usage efficiency.

These dashboards can be useful for quick but multi-sided primary analysis of resource usage efficiency, starting from the most general level and gradually getting a more detailed picture. Of course, to get conclusions about specific performance issues and ways to deal with them, one need to use specialized software for more detailed performance analysis, but this solution helps to take the first step and understand whether there are issues in general and in what direction to move further during performance analysis.

2.2. Informing Users about Performance Issues

As said earlier, users do not need to be provided with such detailed information. In the vast majority of cases, users are not interested in values of some scores they are not aware of. They

name	value	ranking
score_cpu	66.89	№ 91 / 115
score_mem	35.71	№ 102 / 115
score_mpi	0.62	№ 67 / 115
score_io	0.00	№ 1 / 115

Figure 4. Score values and user rank according to these assessments (screenshot from web dashboard)

want to know whether there are performance issues in their applications and how to fix them. For this purpose, automatic notification of users about such cases based on the results of the assessment system was developed.

This is implemented as follows. Within the framework of TASC, rules were previously proposed that automatically detect certain performance issues. In order to notify users of the Lomonosov-2 supercomputer about these issues, a reporting system was implemented within the Octoshell web-site (used as a "single window" for organizing the work with Lomonosov-2 for all users). Among other, this reporting system shows a list of detected issues for each launched job, along with an assumption about their occurrence and recommendations for their elimination. It was decided to add additional rules to this solution that notify users about issues detected by the assessment system. To do this, it was necessary to draw up criteria for rule triggering, including defining threshold values for each score, which was done both on the basis of our own experience and existing materials [21].

As a result, each user always gets notifications about whether his applications have significant issues with the efficiency of using various resources, and what software can be further used to study these issues in more detail. Figure 5 shows an example of such output for a job. The top record is associated with $\text{score}_{\text{cpu}}$, middle one — with score_{io}, lower one — with $\text{score}_{\text{mem}}$. In all cases, the corresponding rule triggered due to the score being so high that it indicates significant impact on overall job performance, leading to this notification to appear. It should be noted that such notifications appear only if resource usage efficiency was really low, which means that false positives are rare in this case.

2.3. Development of Method for Continuous Assessment Prediction

All assessments within the proposed system are calculated based on monitoring data. Two of them — $\text{score}_{\text{cpu}}$ and $\text{score}_{\text{mem}}$ — require collecting additional data from hardware performance monitoring counters, which leads to overheads due to the usage of multiplexing mode (see [22] for details on estimation of such overheads). Therefore, on the Lomonosov-2 supercomputer this data collection is currently organized only for a part of the jobs (for every third job at this moment, in the future this will likely be increased). In addition, technical problems with the monitoring itself may arise. This leads to the fact that assessments are not available for all jobs. Therefore, the task of predicting scores for the remaining jobs arise.

Description	Supposition	Recommendation
The share of useful CPU computations is small.	Not enough information to determine the reasons for this issue.	Try performing the following type of more detailed analysis: General program analysis VTune Amplifier
The usage of I/O network presumably interferes with useful computations.	It is likely that the file system usage is not organized efficiently.	Try performing the following type of more detailed analysis: I/O analysis VTune Amplifier
Processor are idle for a notable time while waiting for memory operations to be performed.	Memory usage is presumably organized inefficiently and requires optimization.	Try performing the following type of more detailed analysis: Analysis of memory usage efficiency Valgrind (Callgrind)

Figure 5. Examples of user notification about resource usage efficiency being too low, notably interfering with useful computations (screenshot from web dashboard for users)

A general method has been earlier proposed that allows for highly accurate score predictions based on data mining methods [1]. However, this method cannot be used on a regular basis because it is too computationally expensive, in particular because it analyzes data on all previously completed jobs. We propose an adapted, faster version of this method that works as follows:

- The prediction process is started every 30 minutes. At each iteration, a list of jobs that were completed after the previous iteration and the duration of which was more than 30 minutes is extracted (if duration is less, the volume of collected monitoring data is insufficient to accurately assess job behavior).
- The knowledge base is loaded. The knowledge base includes the data on N previous jobs with known score values that were calculated using main method without prediction (we will further talk about selecting the optimal value of N).
- The jobs are traversed from older to newer ones, for each of them the following is performed:
  - Prediction is performed based on static information about program source code and system monitoring data collected during program execution. Prediction is based on searching for similar supercomputer jobs in the knowledge base and subsequent aggregation of assessments for the list of detected similar jobs (this has not changed from the original method, see [1]).
  - If it was possible to predict scores, these assessment values are saved.
  - Also, if the true values of the predicted assessments (obtained from the main method) are known for the current job, the knowledge base is updated in FIFO (first in first out) mode.

One of the main questions that arises here is the selection of the N value, which sets the number of jobs under consideration, on the basis of which the prediction will be made. To solve the problem of selecting N, an experiment was conducted in which the method accuracy was evaluated depending on the value of N. In this experiment, the prediction of scores was carried out for the jobs, for which the true score values (obtained using the main method) were known.

Accuracy was evaluated based on 2 metrics: Mean Absolute Error (MAE) и Mean Squared Error (MSE). Both MSE and MAE are positive numbers, and the lesser the value — the better. The task of predicting assessment values is the regression task, and MAE shows average error of the regression model. MSE squares the error, and only then an average value is computed. This way it ensures that predictions with high error have greater effect on the score, which, in combination with MAE, allows for a more precise accuracy evaluation.

Within the experiment, jobs for several months of the Lomonosov-2 supercomputer operation were studied. We only considered jobs with known assessments that lasted more than 30 minutes. The total number of such jobs was ∼4600. Accuracy comparison based on MAE and MSE metrics showed that the best results are achieved with $N = 1250$. Table 1 shows the corresponding metric values. Note that for assessment scores an error of ±10 is considered acceptable, so the proportion of predictions for which the error was more than 10 is evaluated separately.

Table 1. Accuracy results for the continuous assessment prediction method, $N = 1250$

	% of jobs with predicted values	MAE	MSE	% of jobs with AE>10
$\text{score}_{\text{cpu}}$	81.39	2.18	24.83	3.68
$\text{score}_{\text{mem}}$	81.41	2.53	32.59	4.51

It is worth noting that the proposed method for continuous prediction can predict 7% fewer jobs than the original method. However, the proportion of jobs for which scores were predicted, as well as the accuracy of these predictions, are quite high. For example, the average error is less than 3, and the proportion of notable errors (greater than 10) does not exceed 5%.

The proposed method was implemented and is used on a regular basis in everyday practice on the Lomonosov-2 supercomputer. It should be also noted that if both the true and predicted scores (calculated using main and proposed methods, respectively) are available for a certain job, then the true value is used for further analysis as the more accurate one. However, the predicted value is also saved; this will be useful, in particular, for subsequent accuracy evaluations of the prediction method in the future.

We also evaluated the performance of the proposed method on the same experiment data as described earlier. This evaluation was performed on a single Intel Xeon Silver 4214 (1 core was used) with 8 GB RAM (usually less than 2 GB was used). It showed that on average it takes only 1.6 seconds to predict estimates for one job, which is orders of magnitude faster than current real-life needs of the Lomonosov-2 supercomputer.

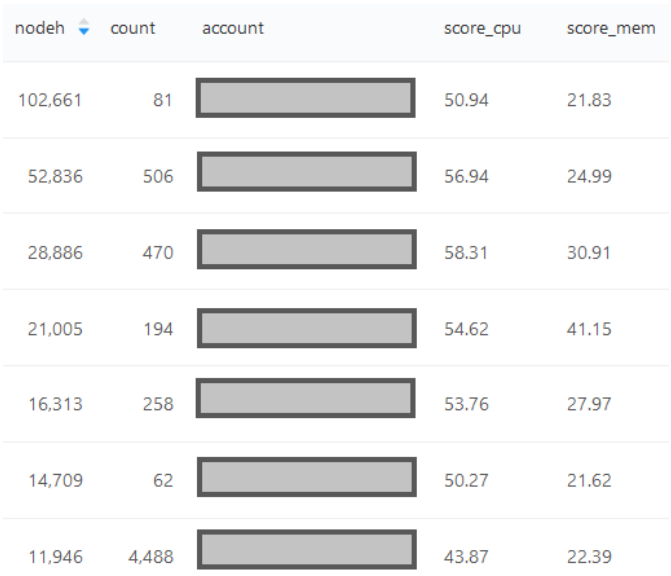
3. Real-life Application of the Assessment System

In this section, we will describe several examples obtained on real-life data from the Lomonosov-2 supercomputer to demonstrate practical application of the proposed assessment system. All these examples were found out using dashboards described earlier.

We have studied a list of individual job launches with the highest $\text{score}_{\text{cpu}}$ scores for May-June of this year. Of interest were the jobs with high $\text{score}_{\text{cpu}}$ combined with high CPU user load — these are the cases when the processor is actively utilized, but at the same time the share of useful calculations is small. As a result, we found launches by two users from the same project with an average CPU load of about 95% (HyperThreading is enabled on most nodes of the Lomonosov-2 supercomputer, so such a load means that not only all physical, but also all logical cores were almost completely utilized in these programs), while the $\text{score}_{\text{cpu}}$ was ∼85

(which means that the processor was fully occupied by useful calculations only 15% of the time during job execution).

Further analysis of these jobs showed that the GROMACS package [23] was used in all cases. So we analyzed the table showing the most active GROMACS users within the year of 2024 (Fig. 6). For each user, this table shows total amount of consumed node-hours with GROMACS launches, number of job launches, as well as average values of $\text{score}_{\text{cpu}}$ and $\text{score}_{\text{mem}}$. It is clear from this table that the $\text{score}_{\text{cpu}}$ assessments are usually much better (less than 60, which is typically OK for HPC programs), meaning that $\text{score}_{\text{cpu}}$ equal to 85 is not typical for this package. Moreover, jobs showing high CPU load together with high $\text{score}_{\text{cpu}}$ appeared for these two GROMACS users only.



nodeh	count	account	score_cpu	score_mem
102,661	81		50.94	21.83
52,836	506		56.94	24.99
28,886	470		58.31	30.91
21,005	194		54.62	41.15
16,313	258		53.76	27.97
14,709	62		50.27	21.62
11,946	4,488		43.87	22.39

Figure 6. Top users of GROMACS software packages sorted by consumed node-hours (screenshot from web dashboard described in 2.1)

It was decided to take a more detailed look at the list of all last GROMACS package launches by these two users. It turned out that they each have several such launches, and these launches are notably different from the rest, since the activity of using other resources (except for the CPU) in them is noticeably lower. In particular, they show very low GPU load at less than 2%, while these users (and other users as well) usually launch GROMACS jobs that show much higher GPU load — up to 50% and higher. Other performance indicators like frequency of cache misses or amount of MPI data sent were also significantly lower. Based on this information, we can conclude that most likely these job launches for some reason did not execute properly. The exact reason can not be determined using the assessment system only, since its purpose is to provide a quick view of whether resources are being utilized efficiently or not. But further analysis, performed by other parts of TASC ("endless" report system and noise measurement tool), showed that these jobs were executed on two noisy compute nodes that were misbehaving at that moment of time. These nodes were sent to repair, and the problem was solved.

This information is surely of interest for both users and supercomputer administrators. It should be noted that these jobs in general showed "normal" performance indicators, did not fail and even completed with a successful finish state, meaning that it would be difficult to determine the root causes of these performance issues without the assessment system.

Let us switch to another example. We considered launches of the popular LAMMPS software package [24] for a couple of months on the Lomonosov-2 supercomputer. Sorting the jobs by the maximum $\text{score}_{\text{mpi}}$ value showed that the top 35 job launches were all performed by 3 users of the same project. This being said, the MPI network usage activity in these jobs was far from the highest — in many cases less than 200 MB/sec on average, while for other users of this package it is more than 1.5 GB/sec. Further study of these jobs with unusually high $\text{score}_{\text{mpi}}$ showed that almost all of them had two performance issues — too small average size of MPI packages, as well as too low network locality (compute nodes allocated for the job were located far from each other). The first issue often leads to additional overheads due to the need to transfer system data and due to higher impact of latency, the second one leads to increased latency. The presence of these issues can be noticeable, which seems to be the case here due to the high $\text{score}_{\text{mpi}}$. These cases should be reported to users so that they can be aware of such situations and try, if necessary, to optimize data transfer in these applications.

Here is another useful, though somewhat unexpected result obtained using the assessment system. Studying the list of individual job launches with the highest $\text{score}_{\text{mem}}$ values, it was found that almost all jobs with a score above 80 (which corresponds to CPU almost constantly stalled waiting data from memory) have a distinctive feature — only one of the nodes allocated to such job is actively used during execution. For example, a job with 45 allocated nodes was found, and the average CPU user load on 44 of them was less than 1% (the 45th node shows fairly high activity). All other such jobs behave similarly. In this case, it can be assumed that either users incorrectly submitted these jobs (for example, several nodes were requested while starting a single-node program), or some technical issues occurred during job launch process. It is not yet entirely clear why this behavior results in extremely high $\text{scores}_{\text{mem}}$, but detecting such incorrect launches is certainly useful, and measures will be taken to promptly identify and fix such situations.

Conclusions

This paper describes the current state and functionality of the TASC software suite. It mentions the data types being collected and analyzed, as well as how this variety of information is stored. The main analytical TASC capabilities are also briefly presented: 1) tool for automatic detection of job-level performance issues for all jobs running on a supercomputer; 2) "endless" workload analysis solution for detailed study of supercomputer performance with a needed level of detail; 3) assessment system for initial, but quick and accurate evaluation of HPC resource efficiency; 4) system for regular measurement of noise level on compute nodes.

The second part of the paper is devoted to the description of recent developments and updates within TASC, which are all related to the assessment system. An approach to visualizing the collected information on assessments was developed, a method for automatic informing supercomputer users about detected performance issues was proposed, and an analysis of the collected statistics on the assessments obtained on the Lomonosov-2 supercomputer was conducted. Also, we proposed a solution for continuous prediction of assessment values, which demonstrates high accuracy and performance.

The results shown in Section 2 were achieved at Lomonosov Moscow State University with the financial support of the Russian Science Foundation, agreement No. 21-71-30003. The research is carried out using the equipment of shared research facilities of HPC computing resources at Lomonosov Moscow State University.

References

1. Voevodin V.V., Shaikhislamov D.I., Nikitenko D.A. How to assess the quality of supercomputer resource usage. Supercomputing Frontiers and Innovations. 2022. Vol. 9, no. 3. P. 4–18. DOI: 10.14529/jsfi220301.
2. High Performance Computing Market Size to Surpass USD 64.65. URL: https://www.globenewswire.com/news-release/2022/04/04/2415844/0/en/High-Performance-Computing-Market-Size-to-Surpass-USD-64-65-Bn-by-2030.html (accessed: 14.08.2024).
3. High Performance Computing Market Size, Growth Report. URL: https://www.fortunebusinessinsights.com/industry-reports/high-performance-computing-hpc-and-high-performance-data-analytics-hpda-market-100636 (accessed: 14.08.2024).
4. Shvets P., Voevodin V., Zhumatiy S. Primary automatic analysis of the entire flow of supercomputer applications. CEUR Workshop Proceedings. 2018. P. 20–32.
5. Shvets P., Voevodin V. "Endless" Workload Analysis of Large-Scale Supercomputers. Lobachevskii Journal of Mathematics. 2021. Vol. 42. P. 184–194. DOI: 10.1134/s1995080221010236.
6. Voevodin V.V., Nikitenko D.A. Recurrent Monitoring of Supercomputer Noise. Supercomputing Frontiers and Innovations. 2023. Vol. 10, no. 3. P. 27–35. DOI: 10.14529/jsfi230304.
7. Jones M.D., White J.P., Innus M., *et al.* Workload Analysis of Blue Waters. 2017. DOI: 10.48550/arXiv.1703.00924. arXiv: 1703.00924.
8. Simakov N.A., White J.P., DeLeon R.L., *et al.* A Workload Analysis of NSF's Innovative HPC Resources Using XDMoD. 2018. DOI: 10.48550/arXiv.1801.04306. arXiv: 1801.04306.
9. Hart D.L. Measuring TeraGrid: workload characterization for a high-performance computing federation. The International Journal of High Performance Computing Applications. 2011. Nov. Vol. 25, no. 4. P. 451–465. DOI: 10.1177/1094342010394382.
10. Patel T., Liu Z., Kettimuthu R., *et al.* Job characteristics on large-scale systems: long-term analysis, quantification, and implications. SC20: International conference for high performance computing, networking, storage and analysis. IEEE, 2020. P. 1–17. DOI: 10.1109/SC41405.2020.00088.
11. Kostenetskiy P., Shamsutdinov A., Chulkevich R., *et al.* HPC TaskMaster- Task Efficiency Monitoring System for the Supercomputer Center. International Conference on Parallel Computational Technologies. Springer, 2022. P. 17–29. DOI: 10.1007/978-3-031-11623-0_2.
12. Isakov M., Del Rosario E., Madireddy S., *et al.* HPC I/O throughput bottleneck analysis with explainable local models. SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2020. P. 1–13. DOI: 10.1109/SC41405.2020.00037.

13. Netti A., Shin W., Ott M., *et al.* A conceptual framework for HPC operational data analytics. 2021 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 2021. P. 596–603. DOI: 10.1109/Cluster48925.2021.00086.

14. Ott M., Shin W., Bourassa N., *et al.* Global experiences with HPC operational data measurement, collection and analysis. 2020 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 2020. P. 499–508. DOI: 10.1109/CLUSTER49012.2020.00071.

15. Voevodin V.V., Antonov A.S., Nikitenko D.A., *et al.* Supercomputer Lomonosov-2: large scale, deep monitoring and fine analytics for the user community. Supercomputing Frontiers and Innovations. 2019. Vol. 6, no. 2. P. 4–11. DOI: 10.14529/jsfi190201.

16. Stefanov K., Voevodin V., Zhumatiy S., Voevodin V. Dynamically reconfigurable distributed modular monitoring system for supercomputers (DiMMon). Procedia Computer Science. 2015. Vol. 66. P. 625–634. DOI: 10.1016/j.procs.2015.11.071.

17. Agrawal K., Fahey M.R., McLay R., James D. User Environment Tracking and Problem Detection with XALT. 2014 First International Workshop on HPC User Support Tools. IEEE, Nov. 2014. P. 32–40. DOI: 10.1109/HUST.2014.6.

18. Nikitenko D., Zhumatiy S., Paokin A., *et al.* Evolution of the Octoshell HPC center management system. International Conference on Parallel Computational Technologies. Springer, 2019. P. 19–33. DOI: 10.1007/978-3-030-28163-2_2.

19. Hoefler T., Mehlan T., Lumsdaine A., Rehm W. Netgauge: A network performance measurement framework. International Conference on High Performance Computing and Communications. Springer, 2007. P. 659–671.

20. Netgauge - Operating System Noise Measurement. URL: https://htor.inf.ethz.ch/research/netgauge/osnoise/ (accessed: 25.09.2024).

21. Top-down Microarchitecture Analysis Method. URL: https://www.intel.com/content/www/us/en/docs/vtune-profiler/cookbook/2023-0/top-down-microarchitecture-analysis-method.html#GUID-FEA77CD8-F9F1-446A-8102-07D3234CDB68 (accessed: 14.08.2024).

22. Voevodin V., Stefanov K., Zhumatiy S. Overhead analysis for performance monitoring counters multiplexing. Russian Supercomputing Days. Springer, 2022. P. 461–474. DOI: 10.1007/978-3-031-22941-1_34.

23. Abraham M.J., Murtola T., Schulz R., *et al.* GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX. 2015. Sept. Vol. 1–2. P. 19–25. DOI: 10.1016/j.softx.2015.06.001.

24. Thompson A.P., Aktulga H.M., Berger R., *et al.* LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. Comp. Phys. Comm. 2022. Vol. 271. P. 108171. DOI: 10.1016/j.cpc.2021.108171.

УДК 519.68 DOI: 10.14529/cmse240304

ПРОГРАММНЫЙ КОМПЛЕКС TASC ДЛЯ АНАЛИЗА ПРОИЗВОДИТЕЛЬНОСТИ СУПЕРКОМПЬЮТЕРОВ: ТЕКУЩЕЕ СОСТОЯНИЕ И ПОСЛЕДНИЕ РАЗРАБОТКИ

© 2024 В.В. Воеводин, Д.И. Шайхисламов, В.А. Серов

Московский государственный университет имени М.В. Ломоносова

(119991 Москва, Ленинские горы, д. 1)

E-mail: vadim@parallel.ru, sdenis1995@gmail.com, sva@srcc.msu.ru

Поступила в редакцию: 04.09.2024

Для обеспечения высокой эффективности работы современных суперкомпьютеров необходимо постоянно анализировать и контролировать различные аспекты их функционирования, уделяя особое внимание изучению потока суперкомпьютерных приложений, выполняющихся на таких системах. Для решения этой задачи ранее был разработан программный комплекс TASC (Tuning Applications for SuperComputers). Он автоматически обнаруживает проблемы с производительностью в суперкомпьютерных приложениях и оценивает эффективность использования ресурсов суперкомпьютера, предоставляет администраторам гибкий инструмент создания отчетов для анализа различных аспектов работы суперкомпьютера с требуемым уровнем детализации, а также оценивает уровень шума на вычислительных узлах. В данной работе приведено детальное описание текущей структуры и возможностей TASC, включая этапы обработки и хранения данных, а также выполнения различных видов анализа. Также описаны новые полученные результаты и разработанные методы в рамках одного из основных компонентов TASC — системы оценок для быстрого и точного определения эффективности использования суперкомпьютерных ресурсов.

Ключевые слова: высокопроизводительные вычисления, суперкомпьютер, анализ производительности, анализ качества работы, анализ операционных данных, мониторинг.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Voevodin V.V., Shaikhislamov D.I., Serov V.A. TASC Software for HPC Performance Analysis: Current State and Latest Developments // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2024. Т. 13, № 3. С. 61–78. DOI: 10.14529/cmse240304.

Литература

1. Voevodin V.V., Shaikhislamov D.I., Nikitenko D.A. How to assess the quality of supercomputer resource usage // Supercomputing Frontiers and Innovations. 2022. Vol. 9, no. 3. P. 4–18. DOI: 10.14529/jsfi220301.
2. High Performance Computing Market Size to Surpass USD 64.65. URL: https://www.globenewswire.com/news-release/2022/04/04/2415844/0/en/High-Performance-Computing-Market-Size-to-Surpass-USD-64-65-Bn-by-2030.html (дата обращения: 14.08.2024).
3. High Performance Computing Market Size, Growth Report. URL: https://www.fortunebusinessinsights.com/industry-reports/high-performance-computing-hpc-and-high-performance-data-analytics-hpda-market-100636 (дата обращения: 14.08.2024).

4. Shvets P., Voevodin V., Zhumatiy S. Primary automatic analysis of the entire flow of supercomputer applications // CEUR Workshop Proceedings. 2018. P. 20–32.

5. Shvets P., Voevodin V. "Endless" Workload Analysis of Large-Scale Supercomputers // Lobachevskii Journal of Mathematics. 2021. Vol. 42. P. 184–194. DOI: 10.1134/s1995080221010236.

6. Voevodin V.V., Nikitenko D.A. Recurrent Monitoring of Supercomputer Noise // Supercomputing Frontiers and Innovations. 2023. Vol. 10, no. 3. P. 27–35. DOI: 10.14529/jsfi230304.

7. Jones M.D., White J.P., Innus M., *et al.* Workload Analysis of Blue Waters. 2017. DOI: 10.48550/arXiv.1703.00924. arXiv: 1703.00924.

8. Simakov N.A., White J.P., DeLeon R.L., *et al.* A Workload Analysis of NSF's Innovative HPC Resources Using XDMoD. 2018. DOI: 10.48550/arXiv.1801.04306. arXiv: 1801.04306.

9. Hart D.L. Measuring TeraGrid: workload characterization for a high-performance computing federation // The International Journal of High Performance Computing Applications. 2011. Nov. Vol. 25, no. 4. P. 451–465. DOI: 10.1177/1094342010394382.

10. Patel T., Liu Z., Kettimuthu R., *et al.* Job characteristics on large-scale systems: long-term analysis, quantification, and implications // SC20: International conference for high performance computing, networking, storage and analysis. IEEE, 2020. P. 1–17. DOI: 10.1109/SC41405.2020.00088.

11. Kostenetskiy P., Shamsutdinov A., Chulkevich R., *et al.* HPC TaskMaster- Task Efficiency Monitoring System for the Supercomputer Center // International Conference on Parallel Computational Technologies. Springer, 2022. P. 17–29. DOI: 10.1007/978-3-031-11623-0_2.

12. Isakov M., Del Rosario E., Madireddy S., *et al.* HPC I/O throughput bottleneck analysis with explainable local models // SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2020. P. 1–13. DOI: 10.1109/SC41405.2020.00037.

13. Netti A., Shin W., Ott M., *et al.* A conceptual framework for HPC operational data analytics // 2021 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 2021. P. 596–603. DOI: 10.1109/Cluster48925.2021.00086.

14. Ott M., Shin W., Bourassa N., *et al.* Global experiences with HPC operational data measurement, collection and analysis // 2020 IEEE International Conference on Cluster Computing (CLUSTER). 2020. P. 499–508. DOI: 10.1109/CLUSTER49012.2020.00071.

15. Voevodin V.V., Antonov A.S., Nikitenko D.A., *et al.* Supercomputer Lomonosov-2: large scale, deep monitoring and fine analytics for the user community // Supercomputing Frontiers and Innovations. 2019. Vol. 6, no. 2. P. 4–11. DOI: 10.14529/jsfi190201.

16. Stefanov K., Voevodin V., Zhumatiy S., Voevodin V. Dynamically reconfigurable distributed modular monitoring system for supercomputers (DiMMon) // Procedia Computer Science. 2015. Vol. 66. P. 625–634. DOI: 10.1016/j.procs.2015.11.071.

17. Agrawal K., Fahey M.R., McLay R., James D. User Environment Tracking and Problem Detection with XALT // 2014 First International Workshop on HPC User Support Tools. IEEE, Nov. 2014. P. 32–40. DOI: 10.1109/HUST.2014.6.

18. Nikitenko D., Zhumatiy S., Paokin A., *et al.* Evolution of the Octoshell HPC center management system // International Conference on Parallel Computational Technologies. Springer, 2019. P. 19–33. DOI: 10.1007/978-3-030-28163-2_2.

19. Hoefler T., Mehlan T., Lumsdaine A., Rehm W. Netgauge: A network performance measurement framework // International Conference on High Performance Computing and Communications. Springer, 2007. P. 659–671.

20. Netgauge - Operating System Noise Measurement. URL: https://htor.inf.ethz.ch/research/netgauge/osnoise/ (дата обращения: 25.09.2024).

21. Top-down Microarchitecture Analysis Method. URL: https://www.intel.com/content/www/us/en/docs/vtune-profiler/cookbook/2023-0/top-down-microarchitecture-analysis-method.html#GUID-FEA77CD8-F9F1-446A-8102-07D3234CDB68 (дата обращения: 14.08.2024).

22. Voevodin V., Stefanov K., Zhumatiy S. Overhead analysis for performance monitoring counters multiplexing // Russian Supercomputing Days. Springer, 2022. P. 461–474. DOI: 10.1007/978-3-031-22941-1_34.

23. Abraham M.J., Murtola T., Schulz R., *et al.* GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers // SoftwareX. 2015. Sept. Vol. 1–2. P. 19–25. DOI: 10.1016/j.softx.2015.06.001.

24. Thompson A.P., Aktulga H.M., Berger R., *et al.* LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales // Comp. Phys. Comm. 2022. Vol. 271. P. 108171. DOI: 10.1016/j.cpc.2021.108171.

Воеводин Вадим Владимирович, к.ф.-м.н., заведующий лабораторией, Научно-исследовательский вычислительный центр, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация)

Шайхисламов Денис Ильгизович, техник, Научно-исследовательский вычислительный центр, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация)

Серов Владимир Александрович, научный сотрудник, Научно-исследовательский вычислительный центр, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация)

УДК 004.272.25, 004.048, 519.254 DOI: 10.14529/cmse240305

КЛАССИФИКАЦИЯ ПОТОКОВОГО ВРЕМЕННОГО РЯДА НА ОСНОВЕ НЕЙРОСЕТЕВЫХ ТЕХНОЛОГИЙ И ПОВЕДЕНЧЕСКИХ ШАБЛОНОВ

© 2024 А.И. Гоглачев

Южно-Уральский государственный университет
(454080 Челябинск, пр. им. В.И. Ленина, д. 76)
E-mail: goglachevai@susu.ru

Поступила в редакцию: 25.08.2024

В статье представлен метод SALTO (Snippet and Autoencoder-based Labeling of Time series coming Online), позволяющий выполнять классификацию подпоследовательностей временного ряда, элементы которого поступают для обработки непрерывным потоком в режиме реального времени. Областью применения разработанного метода являются приложения персональной медицины, промышленного Интернета вещей и цифровой индустрии, в которых предъявляются высокие требования ко времени реакции системы: не более 10 мс в соответствии со стандартом URLLC (Ultra-Reliable Low Latency Communications, сверхнадежная связь с малой задержкой). Метод SALTO предполагает предварительную обработку предварительно сохраненного репрезентативного фрагмента потокового временного ряда и распознавание подпоследовательностей этого ряда, поступающих в реальном времени, с помощью нейросетевой модели. Предобработка выполняется без участия учителя с помощью параллельного алгоритма, который автоматизирует поиск поведенческих шаблонов (сниппетов) ряда, используемых для формирования обучающей выборки. Нейросетевая классификационная модель использует архитектуру автоэнкодеров. Энкодер модели преобразует входную подпоследовательность в скрытое представление и включает в себя два сверточных слоя и один рекуррентный слой. Декодер модели состоит из одного рекуррентного слоя и двух транспонированных сверточных слоев, зеркально отражающих параметры Энкодера. В вычислительных экспериментах на стандартных тестах метод SALTO более чем в полтора раза опережает в среднем передовые аналоги по быстродействию, вписываясь в рамки стандарта URLLC, и при этом показывает в среднем более высокую точность, чем большинство указанных аналогов.

Ключевые слова: временной ряд, классификация временных рядов, автоэнкодер, поведенческие шаблоны (сниппеты) временного ряда, нейронные сети.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Гоглачев А.И. Классификация потокового временного ряда на основе нейросетевых технологий и поведенческих шаблонов // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2024. Т. 13, № 3. С. 79–94. DOI: 10.14529/cmse240305.

Введение

В настоящее время классификация потоковых временных рядов является одной из актуальных задач, востребованных в широком спектре приложений Интернета вещей [1], цифровой индустрии [2], персональной медицины [3]. В указанных приложениях исследуемый субъект демонстрирует поведение с фиксированным количеством предопределенных видов активности и снабжается датчиком, который непрерывно снимает показания деятельности данного субъекта с высокой частотой. Классификация получаемого таким образом временного ряда подразумевает распознавание в реальном времени вида активности субъекта. Примером подобной задачи в области цифровой индустрии может служить случай клети прокатного стана металлургического завода [4], на которой устанавливаются датчики, измеряющие деформацию, механическое напряжение и проч. показатели несколько раз в секунду, и в реальном времени выполняется распознавание режима работы клети с целью

контроля ее остаточного ресурса. Более того, в настоящее время для приложений цифровой индустрии разработан стандарт сверхнадежной связи с малой задержкой URLLC (Ultra-Reliable Low Latency Communications), который предписывает обработку данных за время от 1 до 10 мс [5].

На сегодня разработан ряд нейросетевых моделей для решения задачи классификации данных потокового временного ряда: FCN [6], Rocket [7], ResNet [8], InceptionTime [9] и др., применение которых, однако, требует накладных расходов, связанных с вовлечением эксперта в предметной области для ручной разметки данных обучающей выборки.

В данной статье представлен новый метод решения рассматриваемой задачи, получивший название SALTO (Snippet and Autoencoder-based Labeling of Time series coming Online), который объединяет в себе разметку обучающей выборки при минимальном участии эксперта в предметной области и нейросетевую модель классификации подпоследовательностей потокового временного ряда. Разметка выполняется с помощью параллельного алгоритма поиска поведенческих шаблонов [10] (называемых сниппетами [11]), которые представляют собой подпоследовательности, отражающие типичные активности субъекта. При этом точность классификации снижается незначительно по сравнению с использованием разметки, подготовленной экспертом. Нейросетевая классификационная модель использует простую архитектуру автоэнкодеров, что обеспечивает высокое быстродействие классификации в соответствии со стандартом URLLC.

Статья организована следующим образом. Раздел 1 содержит краткий обзор работ по тематике исследования. В разделе 2 приводятся формальные определения понятий и нотация, используемые в статье. Предложенный метод описан в разделе 3. Результаты вычислительных экспериментов, исследующих точность и быстродействие SALTO по сравнению с передовыми аналогами, приведены в разделе 4. Заключение содержит сводку полученных результатов и направления будущих исследований.

1. Обзор связанных работ

В настоящее время существует множество как аналитических методов, так и нейросетевых подходов к решению задачи классификации временных рядов. Среди аналитических методов [12] можно выделить шейплеты [13], рекуррентные диаграммы (recurrence plots) [14] и методы, основанные на дискретном преобразовании Фурье [15]. В настоящее время одним из основных подходов к классификации временных рядов считается использование нейросетевых технологий [16]. Нейросетевые методы классификации временных рядов используют широкий спектр архитектур: сверточные нейронные сети [6], энкодеры [17], рекуррентные нейронные сети [18] и др. Среди нейросетевых архитектур в данной области наиболее распространены сверточные нейронные сети, поэтому далее в обзоре кратко будут рассмотрены нейросетевые модели этого типа.

Модель FCN (Fully Convolutional Network) [6] состоит из трех сверточных блоков, каждый из которых выполняет свертку и далее пакетную нормализация, результат которой передается в функцию активации Линейный выпрямитель (ReLU, Rectified linear unit) [19]. Особенностью модели является отсутствие локальных слоев пулинга, что делает длину временного ряда неизменной на протяжении всех операций свертки. Данная модель имеет высокое быстродействие и представляет собой хороший базовый уровень для оценки других нейросетевых моделей классификации временных рядов.

Модель Rocket [7] представляет собой однослойную сверточную сеть, использующую ядра свертки со случайными значениями длины, веса и смещения. Размер ядер свертки и смещения выбираются случайным образом, подчиняясь равномерному распределению, веса выбираются случайно, подчиняясь нормальному распределению. Случайные ядра свертки преобразуют признаки, которые используются для обучения модели, на основе логистической регрессии. Совместное использование нейросетевой модели и логистической регрессии позволяет сократить время обучения и вывода модели.

Модель ResNet [8] использует концепцию остаточных соединений (residual connection), которые представляют собой дополнительные связи, обеспечивающие прямое прохождение градиентов по слоям сети без вычисления нелинейных функций активации. Архитектура ResNet обычно делится на четыре части, каждая из которых содержит несколько остаточных блоков (residual block) с различной глубиной. Первая такая часть сети включает один сверточный слой, за которым следует операция подвыборки по максимальному значению (MaxPooling) для уменьшения пространственных размеров входных данных. Вторая часть сети содержит 64 фильтра, а третья и четвертая части содержат 128 и 256 фильтров соответственно. Последняя часть сети обеспечивает операцию глобальной подвыборки по среднему значению (global average pooling) и полносвязный слой, который производит выходные данные. Данная архитектура решает проблему затухания градиента [19], что позволяет ускорить обучение модели.

Модель InceptionTime [9] вместо традиционных сверточных слоев применяет Inception модули, которые, как и в случае с моделью ResNet, используют остаточные соединения для решения проблемы затухающего градиента. Основным компонентом Inception модуля является слой «бутылочного горлышка» (bottleneck), который представляет собой скользящий фильтр длины 1 с шагом 1. Его задача заключается в уменьшении размерности данных, что позволяет уменьшить сложность модели и предотвратить переобучение. Вторым важным компонентом данного модуля являются скользящие фильтры различных длин, которые применяются ко входным данным. Данный подход делает модель InceptionTime более устойчивой к аномалиям в данных.

2. Теоретический базис

Временной ряд (time series) T представляет собой последовательность хронологически упорядоченных вещественных значений:

$$T = (t_1, \ldots, t_n),\ t_i \in \mathbb{R}. \tag{1}$$

Число n обозначается как $|T|$ и называется длиной ряда.

Подпоследовательность (subsequence) $T_{i,m}$ временного ряда T представляет собой непрерывное подмножество T из m элементов, начиная с позиции i:

$$T_{i,m} = (t_i, \ldots, t_{i+m-1}),\ 1 \leqslant m \ll n,\ 1 \leqslant i \leqslant n - m + 1. \tag{2}$$

Временной ряд T может быть логически разбит на сегменты — непересекающиеся подпоследовательности заданной длины m. Здесь и далее без существенного ограничения общности мы можем считать, что n кратно m, поскольку $m \ll n$. Множество сегментов ряда, имеющих длину $m \ll n$, обозначим как S_T^m, элементы этого множества как $S_1, \ldots, S_{n/m}$:

$$S_T^m = (S_1, \ldots, S_{n/m}),\ S_i = T_{m\cdot(i-1)+1,\,m}. \tag{3}$$

Концепция *сниппетов (snippet)* предложена Кеогом и др. в работе [20] и уточняет понятие типичных подпоследовательностей временного ряда следующим образом. Каждый сниппет представляет собой один из сегментов временного ряда. Со сниппетом ассоциируются его ближайшие соседи — подпоследовательности ряда, имеющие ту же длину, что и сниппет, которые более похожи на данный сниппет, чем на другие сегменты. Для вычисления схожести подпоследовательностей используется специализированное расстояние MPdist, основанное на евклидовом расстоянии. Сниппеты упорядочиваются по убыванию мощности множества своих ближайших соседей. Множество сниппетов ряда T, имеющих длину m, назовем словарем сниппетов и обозначим как C_T^m, а элементы этого множества — как $C_1, \ldots, C_K$:

$$C_T^m = (C_1, \ldots, C_K),\ C_i \in S_T^m. \tag{4}$$

Число K ($1 \leqslant K \leqslant n/m$) представляет собой параметр, задаваемый экспертом в предметной области, и отражает соответствующее количество наиболее типичных активностей субъекта. С каждым сниппетом ассоциированы следующие атрибуты: индекс сниппета (порядковый номер сегмента), ближайшие соседи (подпоследовательности, наиболее похожие на сниппет) и значимость данного сниппета (доля подпоследовательностей временного ряда, которые репрезентует сниппет). В словаре сниппеты упорядочиваются по убыванию их значимости.

Для вычисления схожести временных рядов при нахождении сниппетов используется расстояние MPdist [21]. Оно пропорционально количеству подпоследовательностей наперед заданной экспертом длины ℓ ($3 \leqslant \ell \leqslant m$), близких к сниппету в смысле z-нормализованного евклидового расстояния, которое определяется следующим образом. Пусть имеются временные ряды X и Y, $|X| = |Y| = m$. Тогда z-нормализованное евклидово расстояние между X и Y обозначается $\mathrm{ED}_{\mathrm{norm}}(X, Y)$ и

$$\mathrm{ED}_{\mathrm{norm}}(X, Y) = \mathrm{ED}(\hat{X}, \hat{Y}) = \sqrt{\sum_{i=1}^{m} (\hat{x}_i - \hat{y}_i)^2}, \tag{5}$$

$$\hat{x}_i = \frac{x_i - \mu_x}{\sigma_x}, \quad \mu_x = \frac{1}{m}\sum_{i=1}^{m} x_i, \quad \sigma_x^2 = \frac{1}{m}\sum_{i=1}^{m} x_i^2 - \mu^2. \tag{6}$$

3. Модель классификации потоковых временных рядов

3.1. Архитектура

На рис. 1 представлена архитектура предложенного метода, которая предполагает два этапа: предобработку и классификацию. Препроцессор обрабатывает предварительно сохраненный фрагмент временного ряда, который адекватно отражает все активности субъекта. В данном методе участие эксперта сведено к минимуму, поскольку он задает лишь диапазон длины сниппетов $[m_{\min}, m_{\max}]$. Далее Препроцессор выполняет поиск значения длины сниппета, дающее наибольшую точность автоматической разметки временного ряда. Полученная разметка используется для создания обучающей выборки и словаря сниппетов C_T^m, которые будут использованы для обучения Экстрактора. По сравнению с аналогами предложенный метод самостоятельно размечает данные, что позволяет избежать накладных расходов, связанных с ручной разметкой временных рядов.

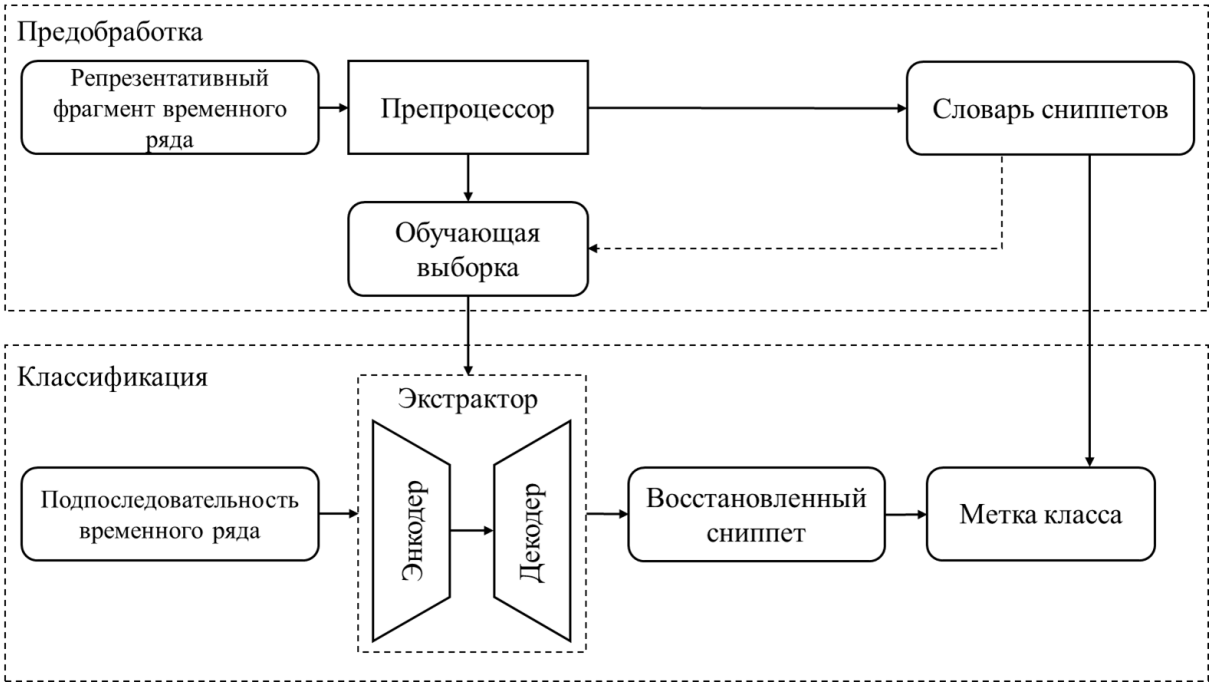


Рис. 1. Архитектура метода SALTO

Полученная выборка используется для обучения Экстрактора, который представляет собой автоэнкодер со стандартной архитектурой [22]. Задача Экстактора заключается в извлечении сниппета из входной подпоследовательности обрабатываемого временного ряда. Полученный сниппет сравнивается с каждый элементом словаря C_T^m, и в качестве метки класса подпоследовательности присваивается порядковый номер ближайшего соседа из словаря сниппетов. Особенностью данной модели по сравнению с аналогами является более простая архитектура, позволяющая добиться высокого быстродействия модели как при обучении, так и при выводе.

3.2. Препроцессор

Препроцессор выполняет разметку предварительно сохраненного репрезентативного фрагмента исходного временного ряда и формирует словарь сниппетов C_T^m, что позволяет подготовить выборку для обучения Экстрактора. На первом шаге работы Препроцессора выполняется z-нормализация всех подпоследовательностей репрезентативного фрагмента по формуле (6).

Дальнейшая обработка входных данных осуществляется при помощи предложенного автором данной статьи алгоритма PaSTiLa [10] (Parallel Snippet-based Time series Labeling), который выполняет поиск сниппетов и разметку фрагмента на высокопроизводительном кластере с узлами, оснащенными графическими процессорами. Алгоритм автоматически подбирает значение длины сниппета m_{best} из указанного диапазона $[m_{\min}, m_{\max}]$ с помощью эвристического критерия. Неформально данный критерий можно описать следующим образом: наилучшим значением длины сниппета $m_{\text{best}} \in [m_{\min}, m_{\max}]$ будет то, которое даст максимально различающиеся сниппеты в смысле расстояния MPdist.

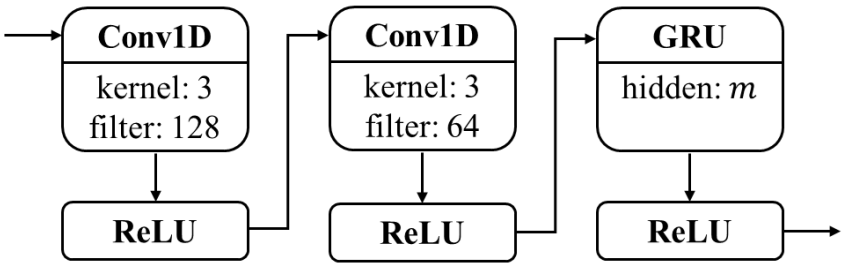
Для ускорения вычислений PaSTiLa задействует ранее предложенный параллельный алгоритм поиска сниппетов PSF (Parallel Snippet-Finder) [23]. Данный алгоритм выполняется на различных узлах кластера для каждого значения из диапазона $[m_{\min}, m_{\max}]$, после

чего результаты, полученные каждым узлом, пересылаются на узел-мастер, который с помощью вышеуказанного критерия вычисляет значение m_{best}.

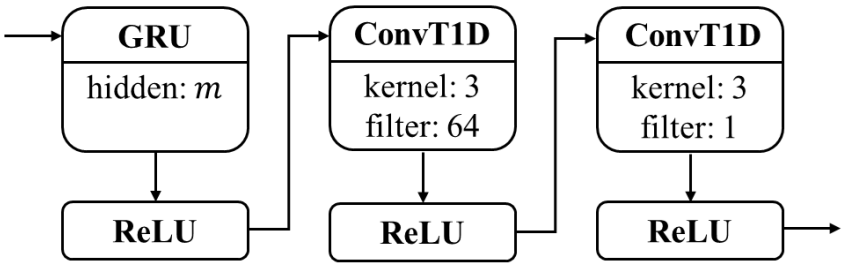
Результатом работы Препроцессора является обучающая выборка Экстрактора, которая обозначается далее как D и представляет собой множество пар $\langle X, Y\rangle$, где X — входная подпоследовательность фрагмента, имеющая длину m_{best}, а Y — соответствующий сниппет той же длины. Полученная выборка формально определяется следующим образом:

$$D = \{\langle X, Y\rangle \mid X = T_{i,m_{\text{best}}},\ Y = C_j, 1 \leqslant i \leqslant n - m_{\text{best}} + 1,\ j = \arg \min_{C_k \in C_T^{m_{\text{best}}}} \text{MPdist}(T_{i,m}, C_k)\}. \tag{7}$$

3.3. Экстрактор



а) Энкодер



б) Декодер

Рис. 2. Структура Экстрактора

Задачей Экстрактора является извлечение сниппетов из входных подпоследовательностей, его архитектура представлена на рис. 2. На вход сети поступает подпоследовательность временного ряда длины m_{best}. Выводом сети является извлеченный из входной подпоследовательности сниппет C_{recov} той же длины.

Нейронная сеть состоит из Энкодера и Декодера. Энкодер выполняет преобразование входных данных в скрытое представление (hidden state). Энкодер состоит из двух сверточных слоев и одного рекуррентного слоя. Первый сверточный слой содержит 128 фильтров, второй — 64 фильтра, оба слоя имеют размер ядра свертки 3. Сверточные слои используются для выявления значимых признаков, определяющих схожесть входной подпоследовательности и одного из элементов $C_T^{m_{\text{best}}}$. Рекуррентный слой реализуется при помощи управляемого рекуррентного блока (Gated Recurrent Units, GRU) [24] и имеет в скрытом состоянии m_{best} признаков. Использование управляемого рекуррентного блока позволяет сохранять долгосрочные зависимости в последовательных данных, тем самым повышая качество извлечение сниппетов. Каждый из слоев использует в качестве функции активации Линейный выпрямитель (ReLU, Rectified linear unit) [19]. Задача Декодера заключается в

преобразовании скрытого представления в сниппет. Декодер состоит из одного рекуррентного слоя и двух транспонированных сверточных слоев. Транспонированный сверточный слой (transposed convolutional layer) [25] используется для увеличения пространственного разрешения входных данных. Этот слой выполняет преобразование, противоположное нормальной свертке, путем перемножения элементов входных данных и ядра свертки.

Для извлеченного при помощи Энкодера сниппета C_{recov} выполняется поиск его ближайшего соседа в словаре сниппетов $C_T^{m_{\text{best}}}$ в смысле расстояния MPdist. Обозначим результирующую метку класса как *label*, тогда *label* принимает значение номера ближайшего соседа C_{recov} в словаре $C_T^{m_{\text{best}}}$:

$$label = \arg \min_{C_i \in C_T^{m_{\text{best}}}} \text{MPdist}(C_{\text{recov}}, C_i). \tag{8}$$

4. Вычислительные эксперименты

Для исследования эффективности предложенного метода были проведены вычислительные эксперименты, в которых точность классификации, время обучения нейросетевой модели и быстродействие ее вывода сравнивались с показателями передовых аналогов, рассмотренных выше в разделе 1.

4.1. Описание экспериментов

Наборы данных. Для экспериментов были использованы стандартные наборы данных TSSB и HAR. TSSB (Time Series Segmentation Benchmark) [26] представляет собой набор из 75 временных рядов разной длины из различных предметных областей, где каждый временной ряд представляет некоторую деятельность субъекта, где количество видов деятельности варьируется от 2 до 7, и субъект меняет вид деятельности с одного на другой до 8 раз.

Набор данных HAR (Human Activity Recognition Dataset) [27] был собран у 30 субъектов, выполняющих шесть различных видов деятельности: ходьба, подъем по лестнице, спуск по лестнице, сидение, стояние, лежание. Набор включает в себя временные ряды данных инерционных датчиков, которые были собраны с помощью смартфонов, носимых субъектами.

Аппаратная платформа и гиперпараметры. Эксперименты были проведены на оборудовании Лаборатории суперкомпьютерного моделирования ЮУрГУ [28]. Аппаратная платформа экспериментов резюмирована в табл. 1.

Таблица 1. Аппаратная платформа экспериментов

Характеристика	Центральный процессор	Графический процессор
Модель	Intel Xeon Gold 6254	NVIDIA Tesla V100 SXM2
Количество ядер	18	5120
Тактовая частота ядра, GHz	4.0	1.3
Оперативная память, Gb	64	32
Пиковая произв-ть, TFLOPS	1.2	15.7

В табл. 2 даны гиперпараметры модели, использованные в экспериментах. Прочие параметры (диапазон длины подпоследовательности, количество активностей субъекта и др.) зависят от предметной области входного временного ряда.

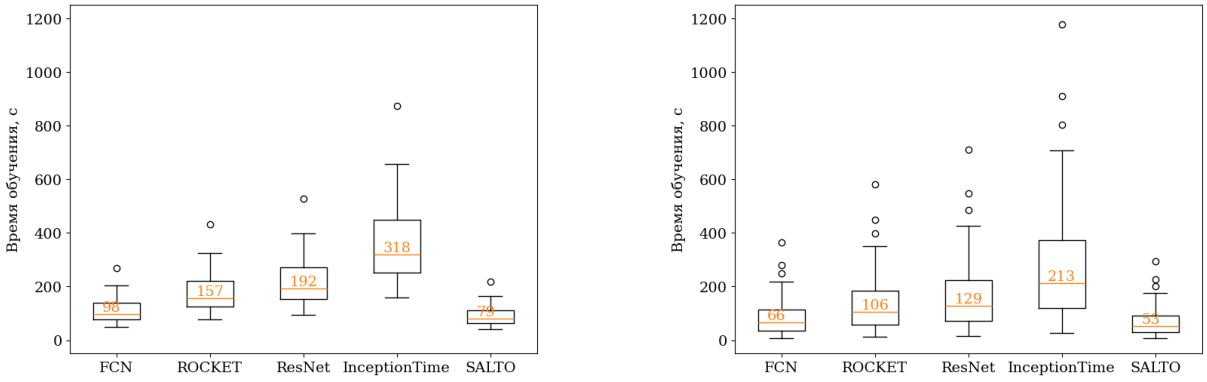
Таблица 2. Значения гиперпараметров модели

Гиперпараметр	Значение
Оптимизатор	Adam
Начальная скорость обучения	0.01
Размер батча	32
Количество эпох обучения	100
Функция потерь	RMSE

Для оценки точности работы Экстрактора в качестве функции потерь использован корень средней квадратичной ошибки RMSE (Root Mean Squared Error), который является одной из наиболее применяемых метрик для оценки моделей автоэнкодеров [29]. Формально данная метрика определяется следующим образом:

$$\mathrm{RMSE} = \sqrt{\frac{1}{m_{\mathrm{best}}} \sum_{i=1}^{m_{\mathrm{best}}} (c_i - \hat{c}_i)^2}, \tag{9}$$

где m_{best} — длина сниппета, c_i и $\hat{c}_i$ – фактическое и предсказанное значения сниппета соответственно.



а) набор данных TSSB б) набор данных HAR

Рис. 3. Время обучения моделей

Метрика точности. Для оценки точности разработанного метода SALTO использована стандартная мера качества классификации F1. Для рассматриваемой в данной статье задачи мультиклассовой классификации данная мера определяется следующим образом:

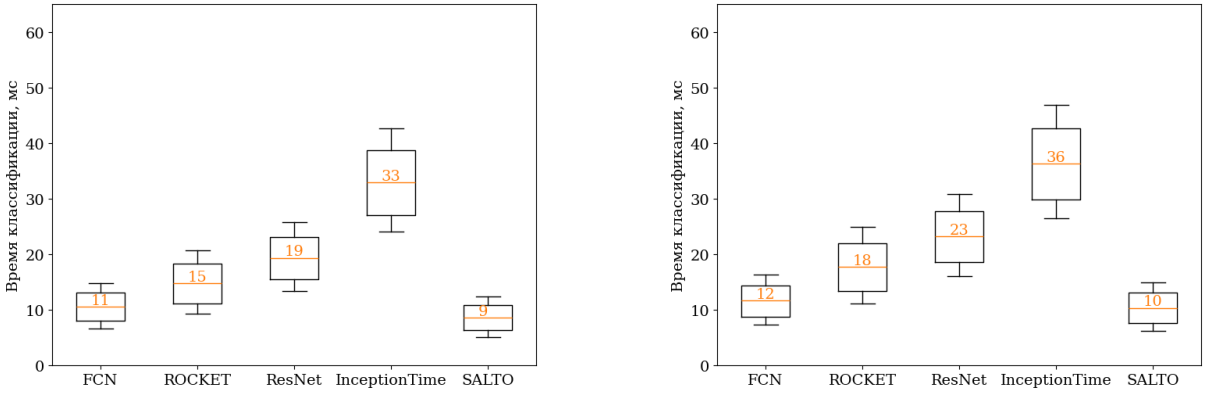
$$\mathrm{Precision}_i = \frac{TP_i}{TP_i + FP_i}, \ \mathrm{Recall}_i = \frac{TP_i}{TP_i + FN_i}, \ \mathrm{F1}_i = 2 \cdot \frac{\mathrm{Precision}_i \cdot \mathrm{Recall}_i}{\mathrm{Precision}_i + \mathrm{Recall}_i}, \tag{10}$$

$$\mathrm{F1} = \frac{1}{K} \sum_{i=1}^{K} \mathrm{F1}_i, 1 \leqslant i \leqslant K,$$

где TP_i, FP_i, TN_i и FN_i — количество истинно положительных, ложно положительных, истинно отрицательных и ложно отрицательных элементов ряда соответственно при сравнении истинной и полученной при помощи алгоритма разметок ряда для i-го класса, K — количество классов (поведений), содержащихся в репрезентативном фрагменте временного ряда.

4.2. Анализ результатов

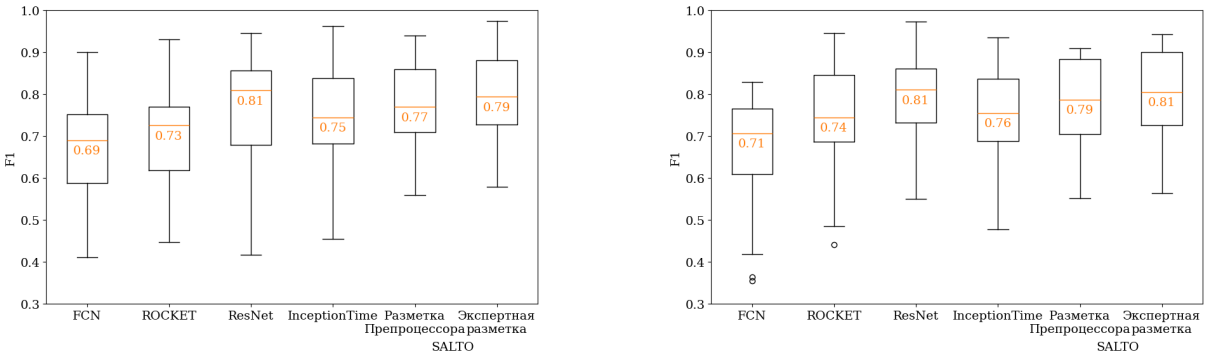
Оценка производительности. На рис. 3 представлены результаты по полному времени обучения рассматриваемых моделей. Можно видеть, что предложенная модель SALTO имеет наименьшее время обучения.



а) набор данных TSSB б) набор данных HAR

Рис. 4. Время классификации одной подпоследовательности

На рис. 4 даны результаты по времени классификации одной подпоследовательности рассматриваемыми моделями. По полученным результатам можно сделать вывод, что предложенная модель имеет наилучшую производительность по сравнению с аналогами. Метод SALTO опережает аналоги по производительности более чем в полтора раза. Предложенный метод выполняет классификацию одной подпоследовательности менее чем за 10 мс, что позволяет применять SALTO, в отличие от большинства конкурентов, в приложениях промышленного интернета вещей с высокими требованиями к производительности в соответствии со стандартом URLLC [5].



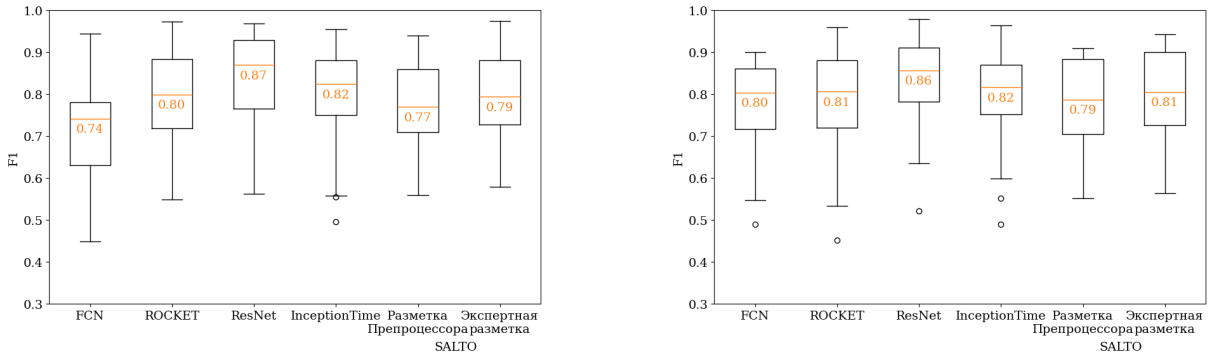
а) набор данных TSSB б) набор данных HAR

Рис. 5. Точность классификации при использовании длины подпоследовательности m_{rand}

Оценка точности. На рис. 5 представлены результаты по оценке точности работы моделей в случае, когда для конкуренты используют случайное значение длины подпоследовательности m_{rand}, которое случайным образом выбиралось как $0.85m_{\text{best}}$ или $1.15m_{\text{best}}$. Для метода SALTO даны результаты с использованием m_{best} при обучении на разметке, полученной Препроцессором, и экспертной разметке. На основе полученных результатов можно сделать вывод, что без рекомендации Препроцессора большинство конкурентов по-

казывают меньшую точность по сравнению с предложенным методом. При сравнении результатов метода SALTO на сгенерированной и экспертных разметках можно видеть, что использование разметки, полученной Препроцессором, дает незначительное отставание в 2% по точности классификации

На рис. 6 представлены результаты по оценке точности работы моделей в случае, когда конкуренты использовали для обучения значение длины подпоследовательности m_{best}. Можно видеть, что использование длины подпоследовательности, найденной Препроцессором, позволяет передовым аналогам добиться значительного увеличения точности: вплоть до 8%.



а) набор данных TSSB

б) набор данных HAR

Рис. 6. Точность классификации при использовании длины подпоследовательности m_{best}

Заключение

В статье рассмотрена задача классификации подпоследовательностей потокового временного ряда, элементы которого поступают для обработки в режиме реального времени. Классификация подразумевает распознавание одного из предопределенных видов активности субъекта, поведение которого отражает обрабатываемый временной ряд. В настоящее время данная задача является актуальной в широком спектре приложений Интернета вещей, цифровой индустрии, персональной медицины и др. Постановка задачи предполагает высокие требования к времени реакции системы: не более 10 мс в соответствии с промышленным стандартом URLLC (Ultra-Reliable Low Latency Communications, сверхнадежная связь с малой задержкой).

В статье предложен метод решения рассматриваемой задачи, получивший название SALTO (Snippet and Autoencoder-based Labeling of Time series coming Online), предполагающий этапы предобработки и классификации. На первом этапе в предварительно сохраненном репрезентативном фрагменте потокового временного ряда осуществляется поиск поведенческих шаблонов (сниппетов), которые представляют собой подпоследовательности, отражающие типичные активности субъекта. Поиск сниппетов выполняется с помощью разработанного автором статьи параллельного алгоритма. При этом участие эксперта в предметной области сведено к минимуму, поскольку он задает лишь диапазон длин сниппетов, что существенно снижает накладные расходы на подготовку обучающих данных. Алгоритм автоматизирует подбор длины сниппета на основе эвристического критерия, предписывающего выбирать такую длину сниппета, при которой полученные сниппеты, соответствующие различным активностям, максимально отличны друг от друга. Найденные сниппеты

используются для разметки репрезентативного фрагмента: подпоследовательность фрагмента, наиболее похожая на один из сниппетов, получает метку соответствующей активности субъекта. Итогом этапа предобработки является обучающая выборка для нейросетевой модели, выполняющей классификацию на втором этапе. Элемент обучающей выборки представляет собой кортеж, состоящий из подпоследовательности и соответствующего ей сниппета. Нейросетевая классификационная модель использует архитектуру автоэнкодеров. Энкодер модели преобразует входную подпоследовательность в скрытое представление и включает в себя два сверточных слоя и один рекуррентный слой. Декодер модели состоит из одного рекуррентного слоя и двух транспонированных сверточных слоев, зеркально отражающих параметры Энкодера. Указанная архитектура не использует вычислительно емкие операции, что снижает время обучения модели и обеспечивает высокое быстродействие классификации в соответствии со стандартом URLLC.

В вычислительных экспериментах на стандартных тестах TSSB (Time Series Segmentation Benchmark) и HAR (Human Activity Recognition) метод SALTO выполняет обучение модели от двух раз быстрее, а классификацию — в среднем более чем в полтора раза быстрее, чем передовые аналоги (модели FCN [6], Rocket [7], ResNet [8], InceptionTime [9]). При выполнении классификации быстродействие SALTO, в отличие от большинства конкурентов, вписывается в рамки стандарта URLLC. Разработанный метод дает в среднем более высокую точность классификации, чем большинство передовых аналогов, когда при их запуске в качестве входного параметра используется случайная длина подпоследовательности, отличная от длины сниппета, которая в методе SALTO выбирается с помощью препроцессора. При этом средняя точность классификации конкурентов повышается, если ими используется указанное значение входного параметра. Исходные коды реализации свободно доступны в сети Интернет по адресу `https://github.com/goglachevai/SALTO`.

В будущих исследованиях планируется изучить возможность использования в методе SALTO иных функций расстояния [30] для поиска сниппетов временного ряда, основанных на вычислении статистических характеристик подпоследовательностей ряда.

Литература

1. Kumar S., Tiwari P., Zymbler M. Internet of Things is a revolutionary approach for future technology enhancement: a review // Journal of Big Data. 2019. Dec. Vol. 6, no. 1. DOI: 10.1186/s40537-019-0268-2.

2. Ali Nemer M., Azar J., Demerjian J., *et al.* A Review of Research on Industrial Time Series Classification for Machinery based on Deep Learning // 2022 4th IEEE Middle East and North Africa COMMunications Conference (MENACOMM). IEEE, Dec. 2022. P. 89–94. DOI: 10.1109/menacomm57252.2022.9998277.

3. Gratius N., Wang Z., Hwang M.Y., *et al.* Digital Twin Technologies for Autonomous Environmental Control and Life Support Systems // Journal of Aerospace Information Systems. 2024. Apr. Vol. 21, no. 4. P. 332–347. DOI: 10.2514/1.i011320.

4. Краева Я.А. Поиск аномалий в сенсорных данных цифровой индустрии с помощью параллельных вычислений // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2023. Т. 12, № 2. С. 47–61. DOI: 10.14529/cmse230202.

5. Serrà J., Pascual S., Karatzoglou A. TS 123 501 - V15.2.0 - 5G; System Architecture for The 5G System (5GS) (3GPP TS 23.501 Version 15.2.0 Release 15). 2018.

6. Wang Z., Yan W., Oates T. Time series classification from scratch with deep neural networks: A strong baseline // 2017 International Joint Conference on Neural Networks (IJCNN). IEEE, May 2017. DOI: 10.1109/ijcnn.2017.7966039.
7. Dempster A., Petitjean F., Webb G.I. ROCKET: Exceptionally Fast and Accurate Time Series Classification Using Random Convolutional Kernels // Data Mining and Knowledge Discovery. 2020. Vol. 34, no. 5. P. 1454–1495. DOI: 10.1007/s10618-020-00701-z.
8. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition // 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, June 2016. DOI: 10.1109/cvpr.2016.90.
9. Ismail Fawaz H., Lucas B., Forestier G., *et al.* InceptionTime: Finding AlexNet for time series classification // Data Mining and Knowledge Discovery. 2020. Sept. Vol. 34, no. 6. P. 1936–1962. DOI: 10.1007/s10618-020-00710-y.
10. Zymbler M.L., Goglachev A.I. PaSTiLa: Scalable Parallel Algorithm for Unsupervised Labeling of Long Time Series // Lobachevskii Journal of Mathematics. 2024. Mar. Vol. 45, no. 3. P. 1333–1347. DOI: 10.1134/s1995080224600766.
11. Imani S., Madrid F., Ding W., *et al.* Introducing time series snippets: a new primitive for summarizing long time series // Data Mining and Knowledge Discovery. 2020. July. Vol. 34, no. 6. P. 1713–1743. DOI: 10.1007/s10618-020-00702-y.
12. Faouzi J. Time Series Classification: A review of Algorithms and Implementations // Machine Learning (Emerging Trends and Applications) / ed. by K. Kotecha. Proud Pen, 2022. URL: https://inria.hal.science/hal-03558165.
13. Ye L., Keogh E. Time series shapelets: a new primitive for data mining // Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, June 2009. P. 947–956. KDD09. DOI: 10.1145/1557019.1557122.
14. Marwan N., Carmenromano M., Thiel M., Kurths J. Recurrence plots for the analysis of complex systems // Physics Reports. 2007. Jan. Vol. 438, 5–6. P. 237–329. DOI: 10.1016/j.physrep.2006.11.001.
15. Schäfer P., Leser U. Fast and Accurate Time Series Classification with WEASEL // Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. ACM, Nov. 2017. CIKM '17. DOI: 10.1145/3132847.3132980.
16. Ismail Fawaz H., Forestier G., Weber J., *et al.* Deep learning for time series classification: a review // Data Mining and Knowledge Discovery. 2019. Mar. Vol. 33, no. 4. P. 917–963. DOI: 10.1007/s10618-019-00619-1.
17. Serrà J., Pascual S., Karatzoglou A. Towards a universal neural network encoder for time series. 2018. DOI: 10.48550/ARXIV.1805.03908.
18. Hüsken M., Stagge P. Recurrent neural networks for time series classification // Neurocomputing. 2003. Jan. Vol. 50. P. 223–235. DOI: 10.1016/s0925-2312(01)00706-8.
19. Hochreiter S. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions // International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems. 1998. Apr. Vol. 6, no. 2. P. 107–116. DOI: 10.1142/s0218488598000094.

20. Imani S., Madrid F., Ding W., *et al.* Matrix Profile XIII: Time Series Snippets: A New Primitive for Time Series Data Mining // 2018 IEEE International Conference on Big Knowledge, ICBK 2018, Singapore, November 17-18, 2018 / ed. by X. Wu, Y. Ong, C.C. Aggarwal, H. Chen. IEEE Computer Society, 2018. P. 382–389. DOI: 10.1109/ICBK.2018.00058.

21. Gharghabi S., Imani S., Bagnall A.J., *et al.* An ultra-fast time series distance measure to allow data mining in more complex real-world deployments // Data Min. Knowl. Discov. 2020. Vol. 34, no. 4. P. 1104–1135. DOI: 10.1007/s10618-020-00695-8.

22. Li P., Pei Y., Li J. A comprehensive survey on design and application of autoencoder in deep learning // Applied Soft Computing. 2023. May. Vol. 138. P. 110176. DOI: 10.1016/j.asoc.2023.110176.

23. Zymbler M., Goglachev A. Fast Summarization of Long Time Series with Graphics Processor // Mathematics. 2022. May. Vol. 10, no. 10. P. 1781. DOI: 10.3390/math10101781.

24. Chung J., Gulcehre C., Cho K., Bengio Y. Gated Feedback Recurrent Neural Networks // Proceedings of the 32nd International Conference on Machine Learning. Vol. 37 / ed. by F. Bach, D. Blei. Lille, France: PMLR, July 2015. P. 2067–2075. Proceedings of Machine Learning Research. URL: https://proceedings.mlr.press/v37/chung15.html.

25. Dumoulin V., Visin F. A guide to convolution arithmetic for deep learning. 2016. DOI: 10.48550/ARXIV.1603.07285.

26. Ermshaus A., Schäfer P., Leser U. ClaSP: parameter-free time series segmentation // Data Mining and Knowledge Discovery. 2023. Vol. 37. P. 1262–1300. DOI: 10.1007/s10618-023-00923-x.

27. Jorge Reyes-Ortiz D.A. Human Activity Recognition Using Smartphones. 2013. DOI: 10.24432/C54S4K.

28. Биленко Р.В., Долганина Н.Ю., Иванова Е.В., Рекачинский А.И. Высокопроизводительные вычислительные ресурсы Южно-Уральского государственного университета // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2022. Т. 11, № 1. С. 15–30. DOI: 10.14529/cmse220102.

29. Minor B.D., Doppa J.R., Cook D.J. Learning Activity Predictors from Sensor Data: Algorithms, Evaluation, and Applications // IEEE Transactions on Knowledge and Data Engineering. 2017. Dec. Vol. 29, no. 12. P. 2744–2757. DOI: 10.1109/tkde.2017.2750669.

30. Lubba C.H., Sethi S.S., Knaute P., *et al.* catch22: CAnonical Time-series CHaracteristics: Selected through highly comparative time-series analysis // Data Mining and Knowledge Discovery. 2019. Aug. Vol. 33, no. 6. P. 1821–1852. DOI: 10.1007/s10618-019-00647-x.

Гоглачев Андрей Игоревич, программист отдела интеллектуального анализа данных и виртуализации Лаборатории суперкомпьютерного моделирования, Южно-Уральский государственный университет (национальный исследовательский университет) (Челябинск, Российская Федерация)

DOI: 10.14529/cmse240305

CLASSIFICATION OF STREAMING TIME SERIES BASED ON NEURAL NETWORK TECHNOLOGIES AND BEHAVIORAL PATTERNS

© 2024 A.I. Goglachev

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

E-mail: goglachevai@susu.ru

Received: 25.08.2024

Data mining of streaming time series is a topical task that occurs in a wide range of subject areas. This article presents the SALTO method (Snippet and Autoencoder based Labeling of Time series coming Online), which combines a neural network model for classifying streaming time series and an analytical method for automatic labeling of training set based on behavioral patterns (snippets). The lightweight architecture of the neural network model used makes it possible to achieve low latency when classifying streaming data. The method involves two stages: preprocessing and classification. At the preprocessing stage, the Preprocessor searches for the optimal value of the length of the subsequence of the input series and performs its labeling. The resulting labels are used to create a training set for the Extractor. The Extractor performs the extraction of the snippet of the input subsequence and assigns it a class based on the similarity of the extracted snippet with snippets from the training set. The experimental results showed that the proposed method performs the classification of a single subsequence in less than 10 ms, which allows SALTO, unlike other competitors, to be used in industrial Internet of Things applications with high performance requirements in accordance with the URLLC (Ultra-Reliable Low Latency Communications) standard.

Keywords: time series, time series classification, autoencoder, time series behavioral patterns (snippets), neural networks.

FOR CITATION

Goglachev A.I., Classification of Streaming Time Series Based on Neural Network Technologies and Behavioral Patterns. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2024. Vol. 13, no. 3. P. 79–94. (in Russian) DOI: 10.14529/cmse240305.

References

1. Kumar S., Tiwari P., Zymbler M. Internet of Things is a revolutionary approach for future technology enhancement: a review. Journal of Big Data. 2019. Dec. Vol. 6, no. 1. DOI: 10.1186/s40537-019-0268-2.
2. Ali Nemer M., Azar J., Demerjian J., *et al.* A Review of Research on Industrial Time Series Classification for Machinery based on Deep Learning. 2022 4th IEEE Middle East and North Africa COMMunications Conference (MENACOMM). IEEE, Dec. 2022. P. 89–94. DOI: 10.1109/menacomm57252.2022.9998277.
3. Gratius N., Wang Z., Hwang M.Y., *et al.* Digital Twin Technologies for Autonomous Environmental Control and Life Support Systems. Journal of Aerospace Information Systems. 2024. Apr. Vol. 21, no. 4. P. 332–347. DOI: 10.2514/1.i011320.

4. Kraeva Y.A. Anomaly Detection in Digital Industry Sensor Data Using Parallel Computing. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2023. Vol. 12, no. 2. P. 47–61. DOI: 10.14529/cmse230202.

5. Serrà J., Pascual S., Karatzoglou A. TS 123 501 - V15.2.0 - 5G; System Architecture for The 5G System (5GS) (3GPP TS 23.501 Version 15.2.0 Release 15). 2018.

6. Wang Z., Yan W., Oates T. Time series classification from scratch with deep neural networks: A strong baseline. 2017 International Joint Conference on Neural Networks (IJCNN). IEEE, May 2017. DOI: 10.1109/ijcnn.2017.7966039.

7. Dempster A., Petitjean F., Webb G.I. ROCKET: Exceptionally Fast and Accurate Time Series Classification Using Random Convolutional Kernels. Data Mining and Knowledge Discovery. 2020. Vol. 34, no. 5. P. 1454–1495. DOI: 10.1007/s10618-020-00701-z.

8. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, June 2016. DOI: 10.1109/cvpr.2016.90.

9. Ismail Fawaz H., Lucas B., Forestier G., *et al.* InceptionTime: Finding AlexNet for time series classification. Data Mining and Knowledge Discovery. 2020. Sept. Vol. 34, no. 6. P. 1936–1962. DOI: 10.1007/s10618-020-00710-y.

10. Zymbler M.L., Goglachev A.I. PaSTiLa: Scalable Parallel Algorithm for Unsupervised Labeling of Long Time Series. Lobachevskii Journal of Mathematics. 2024. Mar. Vol. 45, no. 3. P. 1333–1347. DOI: 10.1134/s1995080224600766.

11. Imani S., Madrid F., Ding W., *et al.* Introducing time series snippets: a new primitive for summarizing long time series. Data Mining and Knowledge Discovery. 2020. July. Vol. 34, no. 6. P. 1713–1743. DOI: 10.1007/s10618-020-00702-y.

12. Faouzi J. Time Series Classification: A review of Algorithms and Implementations. Machine Learning (Emerging Trends and Applications) / ed. by K. Kotecha. Proud Pen, 2022. URL: https://inria.hal.science/hal-03558165.

13. Ye L., Keogh E. Time series shapelets: a new primitive for data mining. Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, June 2009. P. 947–956. KDD09. DOI: 10.1145/1557019.1557122.

14. Marwan N., Carmenromano M., Thiel M., Kurths J. Recurrence plots for the analysis of complex systems. Physics Reports. 2007. Jan. Vol. 438, 5–6. P. 237–329. DOI: 10.1016/j.physrep.2006.11.001.

15. Schäfer P., Leser U. Fast and Accurate Time Series Classification with WEASEL. Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. ACM, Nov. 2017. CIKM '17. DOI: 10.1145/3132847.3132980.

16. Ismail Fawaz H., Forestier G., Weber J., *et al.* Deep learning for time series classification: a review. Data Mining and Knowledge Discovery. 2019. Mar. Vol. 33, no. 4. P. 917–963. DOI: 10.1007/s10618-019-00619-1.

17. Serrà J., Pascual S., Karatzoglou A. Towards a universal neural network encoder for time series. 2018. DOI: 10.48550/ARXIV.1805.03908.

18. Hüsken M., Stagge P. Recurrent neural networks for time series classification. Neurocomputing. 2003. Jan. Vol. 50. P. 223–235. DOI: 10.1016/s0925-2312(01)00706-8.

19. Hochreiter S. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems. 1998. Apr. Vol. 6, no. 2. P. 107–116. DOI: 10.1142/s0218488598000094.

20. Imani S., Madrid F., Ding W., *et al.* Matrix Profile XIII: Time Series Snippets: A New Primitive for Time Series Data Mining. 2018 IEEE International Conference on Big Knowledge, ICBK 2018, Singapore, November 17-18, 2018 / ed. by X. Wu, Y. Ong, C.C. Aggarwal, H. Chen. IEEE Computer Society, 2018. P. 382–389. DOI: 10.1109/ICBK.2018.00058.

21. Gharghabi S., Imani S., Bagnall A.J., *et al.* An ultra-fast time series distance measure to allow data mining in more complex real-world deployments. Data Min. Knowl. Discov. 2020. Vol. 34, no. 4. P. 1104–1135. DOI: 10.1007/s10618-020-00695-8.

22. Li P., Pei Y., Li J. A comprehensive survey on design and application of autoencoder in deep learning. Applied Soft Computing. 2023. May. Vol. 138. P. 110176. DOI: 10.1016/j.asoc.2023.110176.

23. Zymbler M., Goglachev A. Fast Summarization of Long Time Series with Graphics Processor. Mathematics. 2022. May. Vol. 10, no. 10. P. 1781. DOI: 10.3390/math10101781.

24. Chung J., Gulcehre C., Cho K., Bengio Y. Gated Feedback Recurrent Neural Networks. Proceedings of the 32nd International Conference on Machine Learning. Vol. 37 / ed. by F. Bach, D. Blei. Lille, France: PMLR, July 2015. P. 2067–2075. Proceedings of Machine Learning Research. URL: https://proceedings.mlr.press/v37/chung15.html.

25. Dumoulin V., Visin F. A guide to convolution arithmetic for deep learning. 2016. DOI: 10.48550/ARXIV.1603.07285.

26. Ermshaus A., Schäfer P., Leser U. ClaSP: parameter-free time series segmentation. Data Mining and Knowledge Discovery. 2023. Vol. 37. P. 1262–1300. DOI: 10.1007/s10618-023-00923-x.

27. Jorge Reyes-Ortiz D.A. Human Activity Recognition Using Smartphones. 2013. DOI: 10.24432/C54S4K.

28. Bilenko R., Dolganina N., Ivanova E., Rekachinsky A. High-performance Computing Resources of South Ural State University. Bulletin of the South Ural State University. Computational Mathematics and Software Engineering. 2022. Vol. 11, no. 1. P. 15–30. (in Russian) DOI: 10.14529/cmse220102.

29. Minor B.D., Doppa J.R., Cook D.J. Learning Activity Predictors from Sensor Data: Algorithms, Evaluation, and Applications. IEEE Transactions on Knowledge and Data Engineering. 2017. Dec. Vol. 29, no. 12. P. 2744–2757. DOI: 10.1109/tkde.2017.2750669.

30. Lubba C.H., Sethi S.S., Knaute P., *et al.* catch22: CAnonical Time-series CHaracteristics: Selected through highly comparative time-series analysis. Data Mining and Knowledge Discovery. 2019. Aug. Vol. 33, no. 6. P. 1821–1852. DOI: 10.1007/s10618-019-00647-x.

СВЕДЕНИЯ ОБ ИЗДАНИИ

Научный журнал «Вестник ЮУрГУ. Серия «Вычислительная математика и информатика» основан в 2012 году.

Учредитель — Федеральное государственное автономное образовательное учреждение высшего образования «Южно-Уральский государственный университет» (национальный исследовательский университет).

Главный редактор — Л.Б. Соколинский.

Свидетельство о регистрации ПИ ФС77-57377 выдано 24 марта 2014 г. Федеральной службой по надзору в сфере связи, информационных технологий и массовых коммуникаций.

Журнал включен в Реферативный журнал и Базы данных ВИНИТИ; индексируется в библиографической базе данных РИНЦ. Журнал размещен в открытом доступе на Всероссийском математическом портале MathNet. Сведения о журнале ежегодно публикуются в международной справочной системе по периодическим и продолжающимся изданиям «Ulrich's Periodicals Directory».

Решением Президиума Высшей аттестационной комиссии Министерства образования и науки Российской Федерации журнал включен в «Перечень рецензируемых научных изданий, в которых должны быть опубликованы основные научные результаты на соискание ученой степени кандидата наук, на соискание ученой степени доктора наук» по научным специальностям и соответствующим им отраслям науки: 1.2.3 – Теоретическая информатика, кибернетика (физико-математические науки), 2.3.5 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей (физико-математические науки).

Подписной индекс научного журнала «Вестник ЮУрГУ», серия «Вычислительная математика и информатика»: 10244, каталог «Пресса России». Периодичность выхода — 4 выпуска в год.

Адрес редакции, издателя: 454080, г. Челябинск, проспект Ленина, 76, Издательский центр ЮУрГУ, каб. 32.

ПРАВИЛА ДЛЯ АВТОРОВ

1. Правила подготовки рукописей и пример оформления статей можно загрузить с сайта серии `https://vestnikvmi.susu.ru`. **Статьи, оформленные без соблюдения правил, к рассмотрению не принимаются.**
2. Адрес редакционной коллегии научного журнала «Вестник ЮУрГУ», серия «Вычислительная математика и информатика»:

   Россия 454080, г. Челябинск, пр. им. В.И. Ленина, 76, ЮУрГУ, кафедра СП,
   зам. главного редактора Цымблеру М.Л.
3. Адрес электронной почты редакции: `vestnikvmi@susu.ru`
4. **Плата с авторов за публикацию рукописей не взимается, и гонорары авторам не выплачиваются.**

ВЕСТНИК
ЮЖНО-УРАЛЬСКОГО
ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА
Серия
«ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА И ИНФОРМАТИКА»
2024 Том 13, № 3

16+

Техн. редактор *А.В. Миних*

Издательский центр Южно-Уральского государственного университета

Подписано в печать 27.09.2024. Дата выхода в свет 28.10.2024. Формат 60×84 1/8. Печать цифровая.
Усл. печ. л. 11,16. Тираж 500 экз. Заказ 306/396. Цена свободная.

Отпечатано в типографии Издательского центра ЮУрГУ.
454080, г. Челябинск, проспект Ленина, 76.