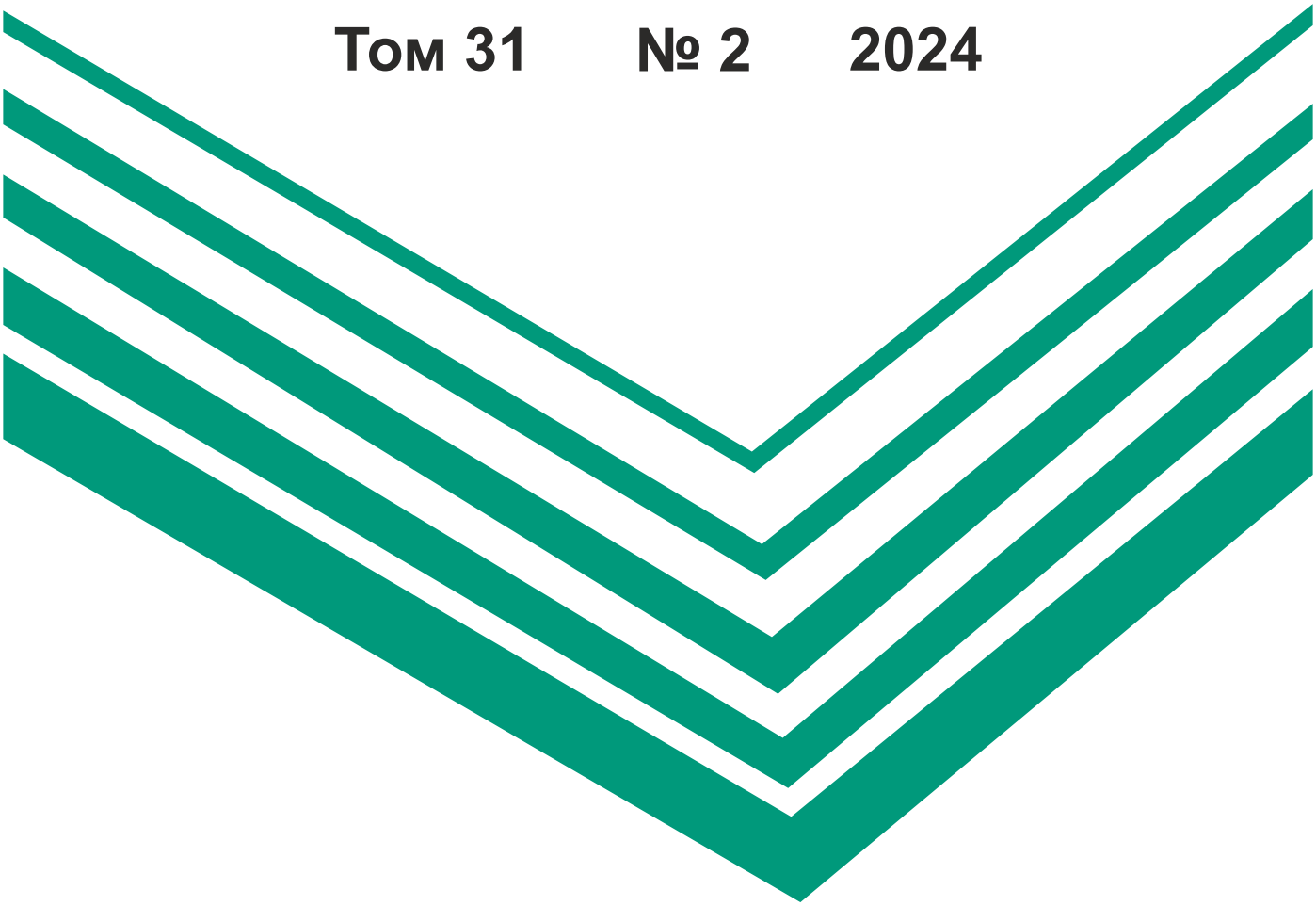
ISSN 2949-5598

ДИСКРЕТНЫЙ АНАЛИЗ И ИССЛЕДОВАНИЕ ОПЕРАЦИЙ

Том 31 № 2 2024



Новосибирск
Издательство Института математики

ДИСКРЕТНЫЙ АНАЛИЗ И ИССЛЕДОВАНИЕ ОПЕРАЦИЙ
Апрель–июнь 2024. Т. 31, № 2. С. 5–27

УДК 519.854.2 DOI: 10.33048/daio.2024.31.760

МУРАВЬИНЫЙ АЛГОРИТМ ДЛЯ ПОСТРОЕНИЯ ОДНОПРОЦЕССОРНОГО РАСПИСАНИЯ С МИНИМИЗАЦИЕЙ ПИКОВОГО ИСПОЛЬЗОВАНИЯ РЕСУРСА

В. В. Балашов[1, a], *А. В. Абрамов*[1, b], *А. А. Чупахин*[1, c], *А. В. Туркин*[2, d], *Ц. Гао*[2, e], *Ш. Сунь*[3, f], *Л. Чжоу*[3, g], *Ц. Сунь*[3, h]

[1] Московский гос. университет им. М. В. Ломоносова, Ленинские горы, 1, 119991 Москва, Россия

[2] Московский исследовательский центр Хуавэй, Смоленская пл., 7-9, 121099 Москва, Россия

[3] Гонконгский исследовательский центр Хуавэй, Сайнс Парк Вест Авеню, 2, 8/F, 999077 Гонконг, КНР

E-mail: [a] hbd@cs.msu.ru, [b] avabramov@lvk.cs.msu.ru, [c] andrewchup@lvk.cs.msu.ru, [d] andrei.turkin@huawei.com, [e] gaojiexing@huawei.com, [f] sunchumin@huawei.com, [g] zhouli107@huawei.com, [h] j.sun@huawei.com

Аннотация. Рассматривается задача построения однопроцессорного расписания с минимизацией пикового использования ресурса вычислителя. В качестве ресурса может выступать оперативная память. Набор заданий для планирования представлен в виде ориентированного ациклического графа, в каждой вершине которого указан объём занимаемого соответствующим заданием ресурса. Высвобождение ресурса, выделенного заданию, происходит по завершении последнего (согласно расписанию) непосредственного потомка этого задания в графе. Ограничением корректности на расписание является соблюдение частичного порядка, определённого графом заданий. Длительности заданий не рассматриваются. Приводится формальная постановка задачи. В качестве алгоритма для её решения предлагается муравьиный алгоритм, модифицированный так, что матрица феромона отражает желательность взаимного расположения (порядка) в расписании для любой пары заданий, а не только для пар соседних заданий. При построении расписания алгоритм для каждого задания выбирает его позицию в расписании, в отличие от известных муравьиных алгоритмов, которые строят расписание в порядке увеличения номеров позиций («слева направо») и для каждой очередной позиции выбирают задание.

Выполнено экспериментальное исследование алгоритма на двух наборах заданий. Графы из первого набора построены так, чтобы априорно была известна оценка оптимального значения целевой функции. Графы из второго набора «слоистые», что соответствует структуре приложений по многостадийной обработке данных. В рамках каждого из наборов графы сформированы случайным образом, с соблюдением заданных параметров генерации и ограничений на структуру графа. Экспериментальное исследование показало высокую точность и стабильность предложенного муравьиного алгоритма. Табл. 1, ил. 12, библиогр. 17.

Ключевые слова: комбинаторная оптимизация, однопроцессорное расписание, минимизация ресурса, муравьиный алгоритм.

Введение

Эффективность использования ресурсов в вычислительных системах (ВС) тесно связана с планированием заданий. Частью указанного процесса является решение задачи составления расписаний. Традиционно решение таких задач направлено на минимизацию общего времени выполнения всех заданий, но в ряде случаев наиболее критичными являются другие ресурсы, такие как оперативная память. Таким образом, при построении ВС возникает вопрос: как определить необходимый для ВС объём ресурсов, которого точно хватит для организации вычислительного процесса и не окажется в избытке?

В настоящей работе рассматривается проблема построения расписания, оптимального с точки зрения уменьшения пикового потребления памяти ВС. ВС состоит из одного процессора, способного в каждый момент времени обрабатывать лишь одно задание, и некоторого объёма памяти. На вход алгоритму планирования подаётся связный ориентированный ациклический граф заданий, в котором рёбра представляют собой зависимости по данным между ними. Таким образом, граф задаёт отношение частичного порядка на множестве заданий. Каждой вершине в графе сопоставлено неотрицательное целое число — объём памяти, занимаемый результатом выполнения задания, соответствующего этой вершине. Результат выполнения задания хранится в памяти до тех пор, пока не выполнятся все непосредственные потомки данного задания, т. е. такие вершины в графе, которые связаны ребром с вершиной, соответствующей данному заданию. Все задания обрабатываются на процессоре без прерываний. Необходимо определить сквозной порядок заданий, который минимизирует пиковое потребление памяти ВС. При этом сквозной порядок должен удовлетворять отношению частичного порядка, заданному графом. Длительности заданий при планировании не учитываются.

Описанная задача является задачей о минимизации максимальной совокупной стоимости (ММСС) [1]. В [2, 3] показано, что в общем случае задача ММСС NP-трудна. В настоящий момент неизвестны полиномиальные алгоритмы точного решения NP-трудных задач. В этой работе предложен алгоритм, относящийся к классу алгоритмов муравьиных колоний, которые берут за основу принципы взаимодействия муравьёв в естественной среде, а именно их роевой интеллект. Каждый отдельный муравей обладает информацией только о локальной обстановке, ни один из них не имеет представления обо всей ситуации в целом — только о том, что узнал от своих сородичей явно или неявно. На неявных взаимодействиях муравьёв основаны механизмы поиска кратчайшего пути от муравейника до источника пищи. Каждый раз, проходя по такому пути, муравей оставляет за собой дорожку феромонов. Другие муравьи, почувствовав такие следы на земле, будут рефлекторно устремляться к нему. Эти муравьи тоже оставляют за собой дорожки феромонов, а значит, чем больше муравьёв проходит по определённому пути, тем более привлекательным он становится для их сородичей. При этом чем короче путь до источника пищи, тем меньше времени требуется муравьям на него, а следовательно, тем быстрее оставленные на нём следы становятся заметными.

Муравьиные алгоритмы (МА) нашли широкое применение в приближённом решении NP-трудных задач, особенно задач на графах. Так, в работах [4, 5] рассматривается задача минимизации суммарного запаздывания на одном приборе при помощи муравьиного алгоритма, описываются подходы локального поиска (перестановки и замены) для улучшения решений, найденных при помощи МА. В [6] приведено описание сбалансированного алгоритма муравьиных колоний для планирования вычислений в Grid-системах. В [7] рассматривается гибридный муравьиный алгоритм для проектного планирования при условии ограниченности ресурсов. В [8] предложен муравьиный алгоритм для отображения графов ресурсных запросов на граф физических ресурсов центра обработки данных.

Часто МА показывает хорошие результаты, которые можно улучшить при помощи различных модификаций, например, локального поиска или правил исключения [5]. В [9] приведено сравнение МА с различными метаэвристическими алгоритмами, в числе которых алгоритм имитации отжига и генетический алгоритм, и установлено, что для задач минимизации суммарного запаздывания больших размерностей МА показывает наиболее хорошие результаты.

Ниже приводится муравьиный алгоритм для решения задачи минимизации максимальных затрат. Разд. 1 отведён математической постановке задачи. В разд. 2 приводится описание предложенного алгоритма.

Результаты численных экспериментов содержатся в разд. 3. В заключении подведены итоги и указаны направления дальнейших исследований.

1. Постановка задачи

Модель прикладной программы может быть представлена как ориентированный ациклический граф $G = (V, E)$, $|V| = n$, $|E| = m$. Каждой вершине графа соответствует задание p_i, $1 \leqslant i \leqslant n$, каждой дуге — передача данных между заданиями. Если $(p_i, p_j) \in E$, $1 \leqslant i, j \leqslant n$, то для выполнения задания p_j необходим результат выполнения задания p_i.

Вычислительная среда состоит из одного процессора SP, который в каждый момент времени способен выполнять только одно задание, и уникального ресурса SR. Каждое задание выполняется на процессоре SP без прерываний. В момент старта задание p_i, $1 \leqslant i \leqslant n$, захватывает $r_{p_i} \geqslant 0$ ресурса SR для хранения результата. При завершении задания p_i происходит высвобождение ресурса, занимаемого такими заданиями p_j, $1 \leqslant j \leqslant n$, $(p_j, p_i) \in E$, для которых p_i является последним выполненным потомком.

Расписание HP сформировано, если для всех заданий из G определён порядок их выполнения на процессоре SP. Фактически HP представляет собой перестановку π номеров заданий из G (будем считать, что задания пронумерованы), поэтому ниже понятия «расписание» и «перестановка заданий» будем считать взаимозаменяемыми:

$$\pi = (j_1, j_2, \ldots, j_n),$$

где j_i — номер задания, находящегося в i-й позиции расписания.

Расписание HP корректно, если выполнены следующие ограничения.

1^*. Каждое задание назначено на процессор SP.

2^*. Процессор SP в каждый момент времени выполняет не более одного задания.

3^*. Частичный порядок, заданный графом зависимостей G, сохранён в HP.

Далее будем говорить, что расписание допустимо $\mathrm{HP} \in \mathrm{HP}^*_{1-3}$, если оно удовлетворяет ограничениям 1^*–3^*.

Минимизируемой целевой функцией (стоимостью в терминах задачи MMCC) является максимальное по всему расписанию количество занятого в BC ресурса. Длительности заданий в рамках задачи не рассматриваются.

Пусть задано расписание $\mathrm{HP} \in \mathrm{HP}^*_{1-3}$. Опишем процедуру расчёта целевой функции для HP. Обозначим через f^k_{HP}, $1 \leqslant k \leqslant n$, количество занятого в BC ресурса для k-й позиции в расписании HP. Пусть A — множество заданий, расположенных в расписании HP на позициях $1, 2, \ldots, k$. Пусть $B \subset A$ — множество заданий, расположенных в расписании HP

на позициях $1, 2, \ldots, k-1$, у каждого из которых все потомки расположены на позициях от 1 до $k-1$. Тогда f^k_{HP} вычисляется по формуле

$$f^k_{\text{HP}} = \sum_{p_j \in A} r_{p_j} - \sum_{p_i \in B} r_{p_i}.$$

Пусть известны значения $\{f^k_{\text{HP}}\}_{k=1}^n$ для расписания HP. Тогда значение целевой функции f_{HP} для такого расписания равно

$$f_{\text{HP}} = \max_{1 \leqslant k \leqslant n} f^k_{\text{HP}}. \tag{1}$$

Требуется для модели прикладной программы G найти такое расписание $\text{HP}_* \in \text{HP}^*_{1-3}$, для которого достигается минимальное значение целевой функции:

$$f_{\text{HP}_*} = \min_{HP \in \text{HP}^*_{1-3}} f_{\text{HP}}.$$

Прежде чем переходить к предлагаемому алгоритму, ответим на вопрос о применимости для решения поставленной задачи ряда известных программ-решателей задач, как коммерческих (с доступной академической лицензией), так и свободно распространяемых. Ограничения задачи задаются графом зависимостей G, определяющим частичный порядок заданий. Если в качестве переменных задачи взять позиции заданий в расписании, то ограничения частичного порядка принимают линейный вид. Однако авторам задачи не известно представления целевой функции задачи ни в линейном, ни в квадратичном виде. В связи с указанной спецификой целевой функции для решения задачи не подходят такие программы-решатели, как Gurobi [10], MOSEK [11], SCIP [12], решатели от COIN-OR [13].

2. Описание предложенного алгоритма

2.1. Структура матрицы феромонного следа. Прежде чем перейти к описанию предложенного алгоритма, необходимо рассмотреть некоторые особенности решаемой задачи и обсудить применимость стандартного муравьиного алгоритма.

Стандартные муравьиные алгоритмы хорошо зарекомендовали себя в решении задач на графах таких, как, например, задача коммивояжёра [14]. Главной причиной, по которой МА оказываются столь эффективными, является дизайн и применение структуры, хранящей информацию о количестве феромона на рёбрах графа. Если рассматривать большинство стандартных задач построения расписаний, решаемых при помощи МА, то можно обратить внимание на то, что переход по каждому ребру вносит вполне определённый вклад в итоговую целевую функцию (ЦФ), например в длительность расписания. Другими словами, точно известно, что ребро из вершины A в вершину B имеет определённую длину,

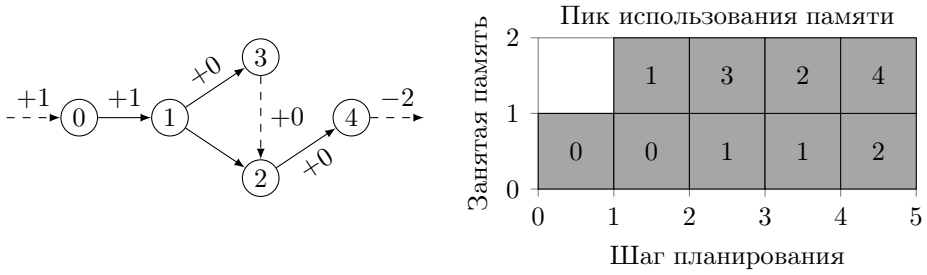


Рис. 1

поэтому вне зависимости от того, на какой позиции в последовательности переходов стоит вершина A, переход из неё в вершину B, в случае, когда он допустим, всегда будет иметь одинаковую стоимость. В случае задачи ММСС это не так. Действительно, переход по ребру $A \to B$ может оказывать различный вклад в итоговое значение ЦФ в зависимости от последовательности обхода графа. На рис. 1–2 слева изображён обход одного и того же графа в разном порядке, а справа — использование ресурса в различных позициях расписания (каждая вершина графа имеет вес 1). Сплошные линии задают частичный порядок, а пунктирные обозначают переходы между несвязанными вершинами. На каждом ребре изображено значение, которое добавляется к ЦФ при переходе по нему. Можно заметить, что переход по ребру из вершины 2 в вершину 4 оказывает различный вклад в значение ЦФ (0 и 1) в зависимости от того, была ли ранее посещена вершина 3. На диаграммах в правой части рисунков в прямоугольниках указаны номера заданий таких, что ресурс, занятый ими, в данной позиции расписания ещё не освобождён.

Из вышесказанного следует, что в рассматриваемой задаче стоимость построенного решения определяют не отдельные вершины или переходы, а их взаимодействие, т. е. их взаимный порядок. Таким образом, появляется необходимость в структуре, которая хранит информацию вида «вершина B следует после вершины A». В этой формулировке «следует после» означает именно отношение порядка, непосредственного следования не требуется.

Как отмечено выше, расписание НР представляет собой перестановку π заданий из G. Пусть π_i — позиция вершины i в расписании НР. Рассмотрим матрицу $R \in \mathbb{R}^{n \times n}$ [15], в которой элемент на месте $(i, j) \in \{1, 2, \ldots, n\}^2$ равен

$$r_{ij} = \begin{cases} 1, & \text{если } \pi_i < \pi_j, \\ 0, & \text{если } \pi_i > \pi_j. \end{cases}$$

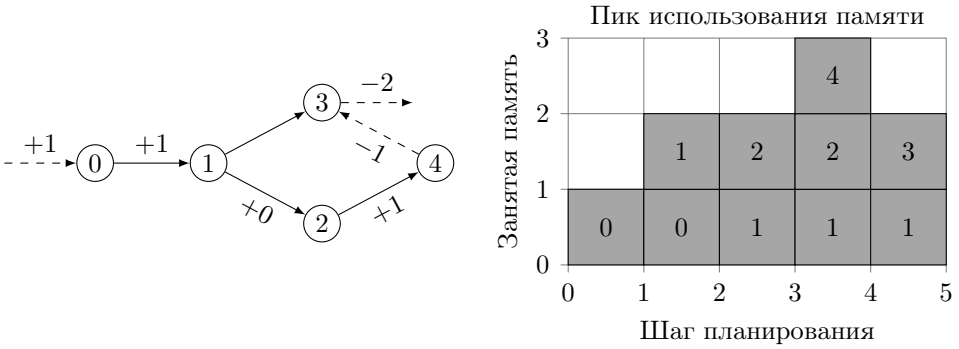


Рис. 2

Можно показать, что существует биективное отображение из множества перестановок π в множество матриц R, если абстрагироваться от допустимости конкретной перестановки (в противном случае можно составить матрицу, которая не будет удовлетворять ни одной допустимой перестановке). Пусть π изначально пуста, и задана матрица R. Будем рассматривать вершины в порядке увеличения меток и добавлять их в π. Вершина с меткой 1 первой включается в перестановку π. Для каждой следующей вершины p_i последовательно проходим по вершинам p_k из π в порядке возрастания меток; если $r_{ik} = 1$, то вставляем вершину p_i в перестановку π перед вершиной p_k, иначе переходим к следующей вершине в перестановке. Если достигнут конец перестановки π, то добавляем вершину p_i в конец перестановки.

Структура матрицы T феромонного следа в предложенном МА основывается на матрице R. Каждая ячейка матрицы τ_{ij} содержит вещественное число — информацию о том, насколько выгодным оказалось расположение заданий, при котором $\pi_i < \pi_j$, на предыдущих итерациях. Таким образом, пройдясь по строкам матрицы, для каждого задания можно определить его наиболее выгодный порядок относительно других заданий. Предложенный МА основывается на данной идее.

Более широко используемая в муравьиных алгоритмах [14] схема, в которой феромон отражает выгодность непосредственных переходов между вершинами (т. е. наличия пар вида $\pi_i\pi_j$ в составе решения), также исследовалась авторами, но продемонстрировала отсутствие сходимости. На рис. 3 показана динамика целевой функции в зависимости от номера итерации алгоритма. Слева график для алгоритма с традиционной схемой работы с феромонным следом, справа — для алгоритма со схемой, описанной в данной статье.

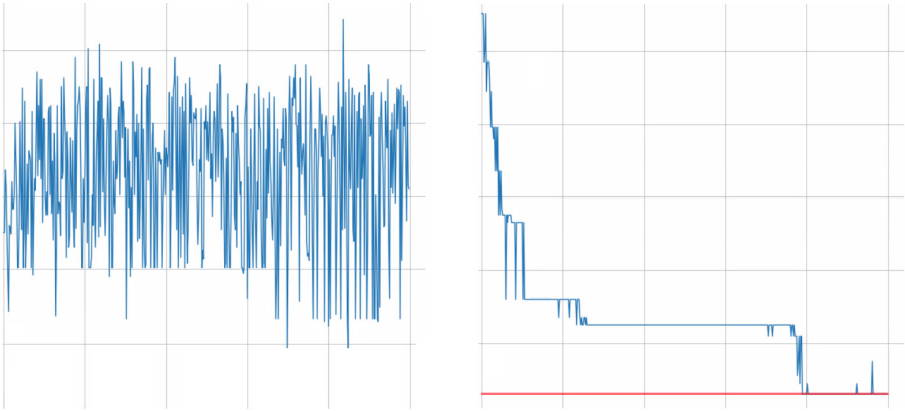


Рис. 3. Сходимость муравьиного алгоритма для различных схем работы с феромонным следом

2.2. Жадная эвристика. Как известно, МА строит решение задачи не только на основе исторической информации, содержащейся в матрице феромонного следа. При построении решения МА опирается также на некоторую жадную эвристику, которая содержит локальную информацию о выгодности текущего перехода. Предложенный МА не стал исключением. В качестве жадной эвристики в нём используется следующая стратегия. Пусть на вход алгоритму подаётся ориентированный ациклический граф $G = (V, E)$, $|V| = n$, $|E| = m$. Пусть также для каждого задания p_i значение i, $1 \leqslant i \leqslant n$, которое задаёт индекс задания, будет меткой вершины, соответствующей данному заданию. Произведём некоторую топологическую сортировку графа G и получим граф $G_s = (V_s, E_s)$, $|V_s| = n$, $|E_s| = m$. Очевидно, что для любой вершины p_j^s графа G_s если $(p_i^s, p_j^s) \in E_s$, то $i < j$, т. е. для любой вершины p_i^s её метка i меньше метки j любого её потомка p_j^s. Таким образом, если рассматривать вершины в порядке возрастания значений меток, то очередь рассмотрения вершины p_k^s наступает лишь после рассмотрения всех её предков.

Будем рассматривать вершины в порядке увеличения значений меток, назначенных вершинам графа при топологической сортировке. Пусть на шаге k, $1 \leqslant k \leqslant n$, длина уже построенного расписания HP^k равна $k-1$ и позиция последнего предка задания p_k^s равна l_k, т. е. $l_k = \max\limits_{p_i^s :\, (p_i^s, p_k^s) \in E_s} \pi_i$, где π_i — позиция вершины p_i^s в расписании HP^k. Ввиду строения графа G_s все предки вершины p_k^s уже содержатся в частичном расписании HP^k. Рассмотрим расписания $\{\mathrm{HP}_d^k\}_{d=l_k}^k$, каждое из которых получается путём вставки вершины p_k^s в HP^k на позицию d (вершины, для которых

$\pi_i \geqslant d$, сдвинутся на 1 позицию вправо). Введём вещественные значения $\eta_{kd} = \frac{1}{f_{\mathrm{HP}_d^k}}$ — локальную информацию о выгодности вставки вершины p_k^s на позицию d в расписание HP^k. Здесь $f_{\mathrm{HP}_d^k}$ — значение целевой функции для частично построенного расписания, вычисленное по формуле (1). Из множества $\{\mathrm{HP}_d^k\}_{d=l_k}^k$ выбирается такое расписание, для которого значение η_{kd} максимально, т. е. $\mathrm{HP}^{k+1} = \mathrm{HP}_b^k$, $b = \underset{l_k \leqslant d \leqslant k}{\operatorname{argmax}} \eta_{kd}$. Другими словами, на каждом шаге новая вершина вставляется в уже построенное расписание таким образом, чтобы локально минимизировать целевую функцию, а выбор вершин для вставки выполняется в порядке, определённом топологической сортировкой.

2.3. Муравьиный алгоритм. В МА используются следующие параметры:

1) критерий останова;

2) N — численность муравьёв, каждый из которых строит своё решение на каждой итерации;

3) ρ — коэффициент распада феромона, $\rho \in [0, 1]$;

4) q_0 — пороговое значение для вероятности перехода, $q_0 \in [0, 1]$;

5) α, β — влияние на вероятность перехода следа и жадной эвристики соответственно;

6) K — число выбираемых наилучших решений.

Пусть также $T = (\tau_{ij}) \in \mathbb{R}^{n \times n}$ — матрица феромонного следа и B — множество наилучших найденных решений, $|B| = K$; матрица T и множество B корректируются на каждой итерации алгоритма; η_{kd} — значение жадной эвристики, описанной в п. 2.2.

На вход алгоритму подаётся ориентированный ациклический граф $G = (V, E)$, $|V| = n$, $|E| = m$. Положим, что к нему уже применена топологическая сортировка, в противном случае применим её. Тогда МА может быть представлен в следующем виде.

Шаг 1. Сгенерировать начальное решение π_0, используя только жадную эвристику. Пусть f^0 — значение целевой функции для такого расписания.

Шаг 2. Положить $\tau_0 := \frac{1}{f^0}$, $T := \tau_0$, $B := \{\pi_0\}$, $f_{\mathrm{best}} := f^0$.

Шаг 3. Сгенерировать $2K$ случайных допустимых расписаний (перестановок), оставить среди них $K - 1$ лучших (с меньшим значением целевой функции) и пополнить ими множество B.

Шаг 4. Обновить $T := (1 - \rho)T$.

Шаг 5. Выполнить процедуру глобального обновления матрицы феромононного следа T.

Шаг 6. Для каждого муравья $a = 1, \ldots, N$ построить допустимое расписание в соответствии с операцией генерации решения и добавить его в множество B, при этом сохраняя его размер $|B| = K$ путём удаления наихудшего решения (решения с наибольшим значением целевой функции).

Шаг 7. Положить $f_{\text{best}} = \min(f_{\text{best}}, \min\limits_{\pi \in B} f_\pi)$.

Шаг 8. Если критерий останова не выполнен, то перейти на шаг 4.

Шаг 9. Вернуть f_{best}.

Процедура генерации решения муравьём заключается в следующем. Как и при использовании жадной эвристики, расписание строится последовательной вставкой заданий на некоторые позиции, только выбор позиции происходит иным образом. Пусть на шаге k имеется множество $\{\text{HP}_d^k\}_{d=l_k}^k$ и вычислены значения $\{\eta_{kd}\}_{d=l_k}^k$. Для расчёта исторической выгодности c_d, $l_k \leqslant d \leqslant k$, т. е. того, насколько хорошим кажется расписание HP_d^k с точки зрения матрицы T, используется произведение

$$c_d = \prod_{p_i \in \text{HP}_d^k} \begin{cases} \tau_{ik}, & \text{если } \pi_i < d, \\ \tau_{ki}, & \text{если } \pi_i > d. \end{cases} \tag{2}$$

Фактически c_d — информация о том, насколько выгодным кажется определённый порядок вершины p_k относительно уже входящих в расписание HP_d^k вершин, исходя из истории, хранящейся в матрице T.

Выбор позиции d для вставки вершины p_k основывается на вероятностной схеме. Рассмотрим распределение вероятностей

$$P_{kd} = \frac{[c_d]^\alpha [\eta_{kd}]^\beta}{\sum\limits_{h \in [l_k, k]} [c_h]^\alpha [\eta_{kh}]^\beta}, \quad d \in [l_k, k]. \tag{3}$$

Пусть $q \in [0, 1]$ — число, случайно выбранное в соответствии с равномерным распределением на указанном отрезке. Если $q \leqslant q_0$, то положим $d = \operatorname{argmax}\limits_{h \in [l_k, k]} [c_h]^\alpha [\eta_{kh}]^\beta$. Иначе d выбирается в соответствии с распределением вероятностей (3).

После выбора d и обновления π ($\text{HP}^{k+1} := \text{HP}_d^k$) происходит локальное обновление следа: все значения τ_{ij}, которые появились в произведении (2), изменяются по формуле

$$\tau_{ij} = (1 - \rho)\tau_{ij} - \rho\tau_0.$$

Процедура глобального обновления матрицы T выполняет для каждого $\pi \in B$ следующие действия.

1: **for** $i = 1, 2, \ldots, n-1$ **do**
2: **for** $j = i+1, i+2, \ldots, n$ **do**
3: **if** $\pi_i < \pi_j$ **then** $\tau_{ij} := \tau_{ij} + \frac{1}{f_\pi}$
4: **else** $\tau_{ji} := \tau_{ji} + \frac{1}{f_\pi}$

В качестве критерия останова можно рассматривать заданное общее число итераций алгоритма или же заданное число итераций без улучшения целевой функции на наилучшем найденном решении.

3. Экспериментальное исследование свойств алгоритма

3.1. Описание инструментального компьютера и наборов данных. В рамках настоящей работы эксперименты проводились на компьютере со следующими характеристиками:

- ЦП: Intel(R) Xeon(R) CPU E5-2650 v4 @ 2,20 ГГц;
- объём дискового хранилища 450 ГБ;
- объём ОЗУ 62 ГБ;
- ОС Ubuntu 16.04.5 LTS.

Алгоритм на языке C++ реализован последовательно и использует ресурсы одного процессорного ядра.

Выполнены две серии экспериментов на синтетических наборах входных данных. В первой серии экспериментов набор данных состоит из графов, для которых по построению известно оптимальное или субоптимальное (несущественно большее чем оптимальное) значение целевой функции. Генерация таких графов основана на результатах работы [16], где предложен полиномиальный алгоритм построения расписания с минимизацией пикового использования ресурса для случая, когда граф заданий является последовательно-параллельным графом (ППГ).

С учётом того, что в [16] рассматривалась задача, в которой количество используемого ресурса сопоставлено не вершине графа, а ребру, схема генерации графа для экспериментов выглядит следующим образом.

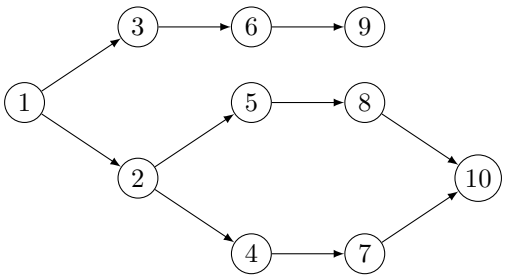


Рис. 4. Последовательно-параллельный граф

Вначале генерируется ППГ (рис. 4) для исходной задачи, т. е. с ресурсом, сопоставленным вершинам. Параметрами генерации являются число вершин и плотность графа (отношение числа рёбер к числу вершин). Затем осуществляется переход от этого ППГ к графу с ресурсом, сопоставленным рёбрам. При этом для каждой вершины исходного графа выполняется следующее. Если у вершины один потомок, то ресурс переносится из вершины на выходящее из неё ребро. Если у вершины более одного потомка, то используется преобразование, показанное (для случая двух потомков) на рис. 5 и в общем случае выводящее получаемый граф из класса ППГ.

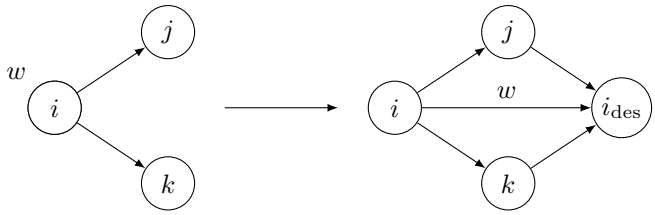


Рис. 5. Переход от ППГ с ресурсом в вершинах к ППГ с ресурсом на рёбрах

Из полученного графа выделяется ППГ (также с ресурсом на рёбрах), и для него строится оптимальное расписание при помощи алгоритма из [16]. Затем в граф возвращаются ранее отброшенные рёбра, при этом расписание корректируется для учёта соответствующих им зависимостей (на этом шаге расписание может стать субоптимальным). Затем происходит возврат к исходной постановке задачи с «возвращением» ресурса в вершины. Построенное расписание корректно и для исходной постановки, поскольку учитывает все зависимости.

Чтобы вывести формируемые графы заданий из класса последовательно-параллельных, для которых задача решается полиномиальным алгоритмом, в исходный ППГ добавляются дополнительные рёбра, которые не нарушают корректности построенного (суб-) оптимального расписания и не меняют значения целевой функции на этом расписании: если в расписании имеются задания a, b, c (именно в таком порядке, необязательно подряд), причём c — последний в расписании непосредственный потомок a, а ребра $a \to b$ в графе нет, то это ребро можно добавить в граф, не изменив значения целевой функции на расписании. Неизменность целевой функции обусловлена тем, что ресурс, занятый заданием a, по-прежнему высвобождается после выполнения c. После добавления рёбер проверяется, что полученный граф не является ППГ.

В наборе данных для первой серии экспериментов число вершин варьируется от 60 до 250, средняя плотность графов равна 1,7, а веса вершин выбраны на отрезке $[1, 100]$ согласно равномерному распределению.

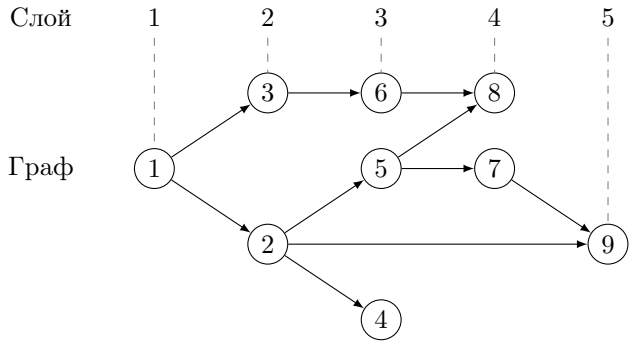


Рис. 6. Слоистый граф

Набор данных для второй серии экспериментов состоит из «слоистых» графов (рис. 6), структура которых соответствует приложениям по многостадийной обработке данных. В таких графах большинство рёбер связывают между собой вершины соседних слоёв. Для каждого из графов задаются число вершин и их веса, средняя ширина слоя, плотность связей между слоями, а также максимальное число слоёв, минуя которые можно провести ребро (на рис. 6 это ребро $2 \to 9$). Среднее отношение числа рёбер к числу вершин при генерации задано равным 3. Веса вершин выбраны на отрезке $[10, 50]$ согласно равномерному распределению. Схема генерации слоистых графов приведена в статье [17], там же дана ссылка на репозиторий с исходным кодом генератора.

Характеристики второго набора данных приведены в табл. 1. Из таблицы видно, что графы масштабированы в глубину, т. е. число вершин растёт за счёт увеличения числа слоёв (глубины графа). Для каждого числа вершин сгенерировано по 20 графов с приблизительно равными характеристиками. Результаты в таблице усреднены по графам с одинаковым числом вершин.

3.2. Настройки муравьиного алгоритма. Во всех экспериментах муравьиному алгоритму выставлялись заранее подобранные значения параметров: $N = 10$, $\rho = 0{,}2$, $q_0 = 0{,}8$, $\alpha = 0{,}6$, $\beta = 0{,}4$, $K = 6$. Рассматривались следующие критерии останова:

1) выполнение заданного числа итераций (500) без улучшения целевой функции на наилучшем найденном решении;

2) выполнение фиксированного числа итераций (1000).

В зависимости от выбранного критерия останова обозначим варианты алгоритма соответственно МАН и МА1000.

Фиксированное число итераций выбрано так, чтобы качество решений, находимых МАН и алгоритмом с фиксированным числом итераций,

Таблица 1

Параметры слоистых графов

Число вершин	Число рёбер	Глубина	Минимальное число вершин в слое	Среднее число вершин в слое	Максимальное число вершин в слое
50	150	5	3	10	15
55	161	6	3	10	15
60	181	6	5	11	16
65	196	9	3	7	11
70	211	10	3	7	11
75	226	10	3	8	11
80	240	11	2	7	11
85	257	10	3	9	13
100	290	15	3	7	10
120	358	15	3	8	12
130	389	19	3	7	11
140	421	20	3	7	11
150	451	21	3	7	11
160	478	23	2	7	11
170	510	24	2	7	11
180	542	26	3	7	11
190	569	24	3	8	12
200	600	25	3	8	12

было сопоставимым. На рис. 7 это видно по диаграмме, практически симметричной относительно отметки 0. Одной из целей экспериментов было выяснить, позволяет ли алгоритм МАН на рассматриваемом наборе слоистых графов заданий получить выигрыш по времени работы относительно МА1000 за счёт автоматического определения необходимости останова, когда оптимальное решение более не удаётся улучшить (при этом МА1000 может продолжать работу).

3.3. Результаты экспериментов для графов заданий с известным субоптимальным значением целевой функции. В этой серии экспериментов выполнялся прогон МАН и алгоритма, использующего только жадную эвристику (см. п. 2.2). Каждый из этих алгоритмов строил решение со значением целевой функции, не худшим (а в ряде случаев лучшим), чем заранее известное субоптимальное значение. При этом различие в значениях целевой функции на решении между муравьиным и жадным алгоритмами не превышало 1% в пользу муравьиного алгоритма.

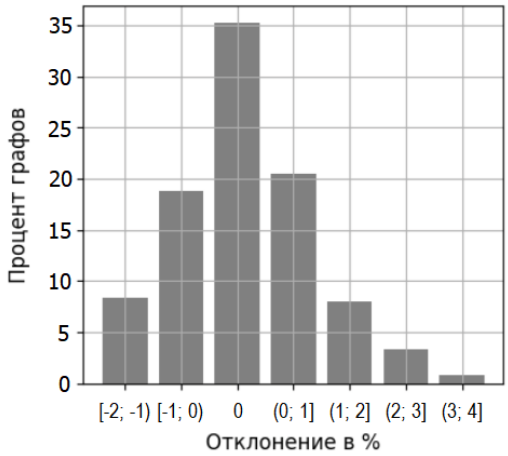


Рис. 7. Относительное отклонение целевой функции на решениях, получаемых MAH и MA1000

3.4. Результаты экспериментов для слоистых графов заданий. На рис. 7 приведена сводная информация о попарном сравнении значений целевой функции на решениях, получаемых МА с различными критериями останова, запускаемых на одних и тех же наборах данных. Например, по диаграмме видно, что целевая функция на решениях от MAH превышает целевую функцию на решениях от MA1000 на 1–2% (интервал по горизонтальной оси) примерно в 8% случаев (высота столбца диаграммы). В такой же доле случаев на решениях от MAH целевая функция меньше на 1–2%, чем целевая функция на решениях от MA1000. Основная часть (более 95%) отображённых на диаграмме случаев приходятся на её практически симметричную часть в интервале от -2 до 2% по горизонтальной оси, что говорит о том, что по качеству решений на рассматриваемом наборе входных данных MAH и MA1000 выступают на равных.

На рис. 8 показано время работы MAH и MA1000. На горизонтальной оси расположены номера графов, отсортированных в лексикографическом порядке по числу вершин, затем по числу рёбер. Большой разброс по вертикальной оси у MAH вызван непостоянным числом итераций при запуске, так как алгоритм работает до тех пор, пока решения не перестанут улучшаться. Видно, что время работы MAH не превосходит, а зачастую много меньше времени работы МА с жёстко установленным числом итераций 1000.

Поскольку муравьиный алгоритм рандомизированный, необходимо исследовать стабильность его результата, т. е. разброс значений целевой функции при прогонах на одних и тех же входных данных. На рис. 9

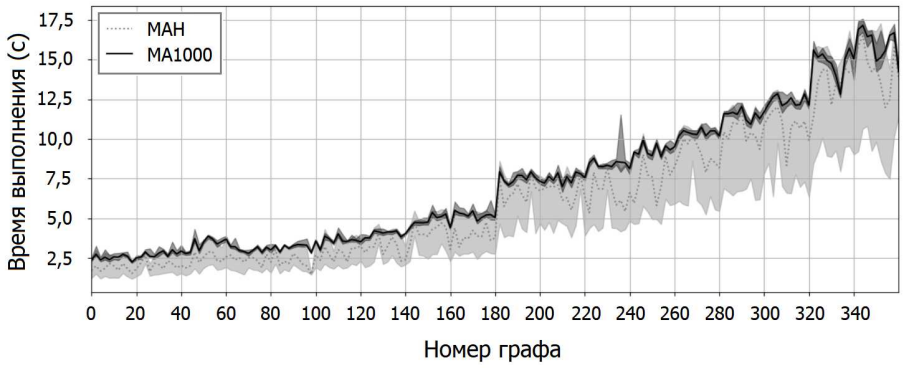


Рис. 8. Время выполнения МА с различными критериями останова

показан такой разброс для MAH и MA1000. Можно видеть, что разброс значений обоих алгоритмов не сильно различается и в среднем составляет около 5,5%.

Из графиков на рис. 7–9 следует, что MAH оказывается более гибким. Этот алгоритм способен автоматически определить, когда необходимо остановить процесс построения решения, за счёт чего выигрывает по времени работы у МА с жёстко заданным большим числом итераций (в данном случае — 1000), выдавая при этом решение сопоставимого качества.

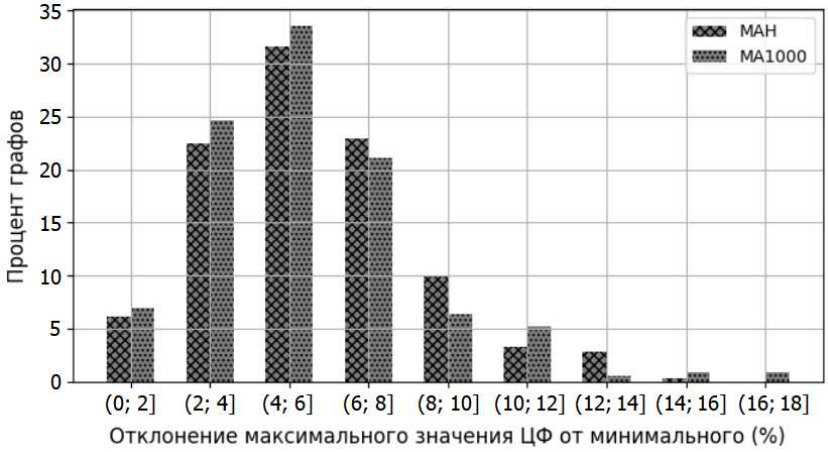


Рис. 9. Относительное отклонение максимального значения ЦФ от минимального, полученных на одном и том же графе заданий при различных запусках

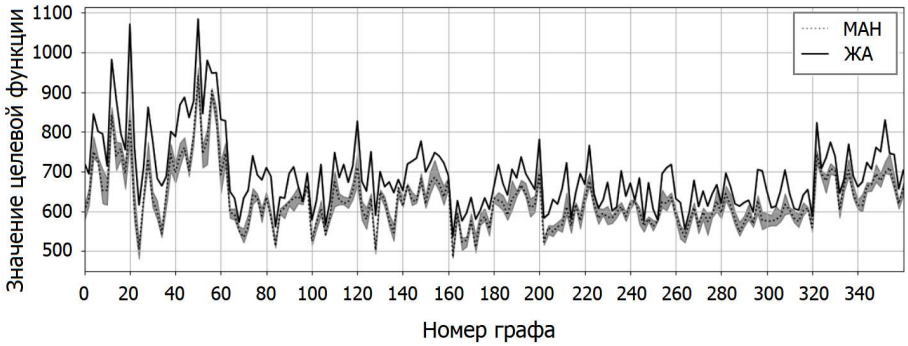


Рис. 10. Графики целевой функции на решениях, получаемых жадным алгоритмом и МАН

Рассмотрим теперь МАН на слоистых графах в сравнении с жадным алгоритмом (ЖА), использующим эвристику из п. 2.2. На рис. 10–11 показаны графики значения целевой функции на решении и времени работы алгоритмов. На всех графах жадный алгоритм отработал меньше секунды. Можно заметить, что с увеличением размера графа растёт не только время работы алгоритма МАН, но и разброс времени его работы, что связано с увеличением числа возможных перестановок и повышением вероятности продления работы МАН в связи с очередным небольшим улучшением решения.

На рис. 12 представлено относительное отклонение целевой функции на решениях, получаемых жадным алгоритмом, от целевой функции на решениях, найденных МАН. Видно, что в более чем 45% случаев МАН удалось найти решение, на 10–30% лучшее по значению целевой функции, чем решение от жадного алгоритма, что говорит о способности МА, использующего ту же жадную эвристику, существенно улучшать решение за счёт использования механизма феромонного следа.

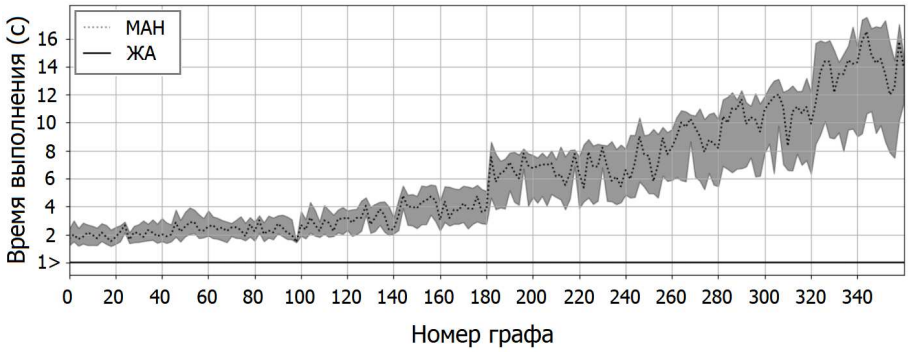


Рис. 11. Время работы жадного алгоритма и МАН

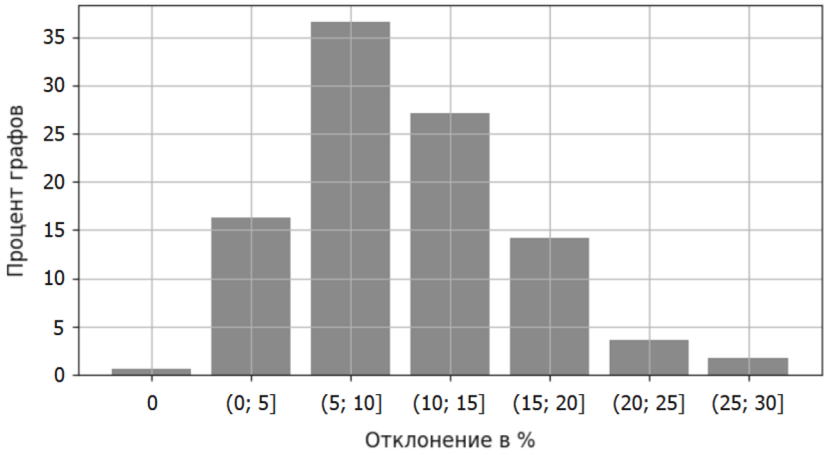


Рис. 12. Относительное отклонение целевой функции на решениях, получаемых жадным алгоритмом и МАН

Заключение

Предложен муравьиный алгоритм для решения задачи о построении расписания с минимизацией пикового использования ресурса, описаны отличия поставленной задачи от стандартных задач, решаемых при помощи МА. По итогам исследования алгоритма на рассмотренной выборке слоистых графов заданий, сделаны следующие выводы.

1. МА способен значительно улучшать качество решений, найденных жадной эвристикой, что вкупе с приемлемым временем работы делает использование МА оправданным.

2. МА, останавливающийся после выполнения заданного числа итераций (500) без улучшения целевой функции на наилучшем найденном решении, позволяет на графах рассмотренной размерности получить заметный выигрыш по времени выполнения по сравнению с МА, останавливающимся после выполнения фиксированного числа итераций (1000). При этом указанные МА выдают решения сопоставимого качества.

3. Применяемая в предложенном алгоритме схема работы с феромонным следом, учитывающая не только непосредственное следование заданий в расписании, вносит существенный вклад в сходимость МА.

В качестве направлений дальнейшего развития МА рассматриваются выполнение параллельной реализации (в рамках одной итерации могут одновременно строиться расписания для разных муравьёв), а также увеличение доли случайности в работе алгоритма при стабилизации значения целевой функции на лучшем решении с целью выхода из локального оптимума.

Финансирование работы

Исследование выполнено за счёт бюджетов организаций, обозначенных авторами на первой странице статьи. Дополнительных грантов на проведение или руководство этим исследованием получено не было.

Конфликт интересов

Авторы заявляют, что у них нет конфликта интересов.

Литература

1. **Гэри М., Джонсон Д.** Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982. 416 с.
2. **Abdel-Wahab H. M.** Scheduling with applications to register allocation and deadlock problems: PhD Thes. Waterloo, ON: Univ. Waterloo, 1976. 440 p.
3. **Sethi R.** Complete register allocation problems // SIAM J. Comput. 1975. V. 4, No. 3. P. 226–248.
4. **Bauer A., Straus C., Hartle R. F.** Minimizing total tardiness on a single machine using ant colony optimization // Cent. Eur. J. Oper. Res. Econ. 2000. V. 8, No. 2. P. 125–141.
5. **Гафаров Е. Р.** Гибридный алгоритм решения задачи минимизации суммарного запаздывания для одного прибора // Информ. технологии. 2007. № 1. С. 30–37.
6. **Jha S.** Balanced ant colony algorithm for scheduling DAG to grid heterogeneous system // Int. J. Sci. Eng. Res. 2011. V. 2, No. 6. 10 p.
7. **Myszkowski P., Skowronski M., Olech L., Oslizlo K.** Hybrid ant colony optimization in solving multi-skill resource-constrained project scheduling problem // Soft Comput. 2015. No. 19. P. 3599–3619.
8. **Костенко В. А., Плакунов А. В.** Муравьиные алгоритмы для планирования вычислений в центрах обработки данных // Вестн. Моск. ун-та. Сер. 15. Вычисл. математика и кибернетика. 2017. № 1. С. 44–50.
9. **Gagne C., Price W. L., Grave M.** Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times // J. Oper. Res. Soc. 2002. V. 53. P. 895–906.
10. Gurobi optimizer. Gurobi optimization, 2023. URL: `www.gurobi.com/solutions/gurobi-optimizer` (accessed Mar. 22, 2024).
11. MOSEK solver. MOSEK ApS, 2023. URL: `www.mosek.com/products/mosek` (accessed Mar. 22, 2024).
12. SCIP: Solving Constraint Integer Programs. Berlin: Zuse Inst. Berlin, 2023. URL: `www.scipopt.org` (accessed Mar. 22, 2024).
13. COIN-OR: Computations Infrastructure for Operations Research. Projects by Category. Towson: COIN-OR Found., 2023. URL: `www.coin-or.org/projects` (accessed Mar. 22, 2024).
14. **Xuan H. H., Linh-Trung N., Dong D. D., Huynh T.** Solving the Traveling Salesman Problem with ant colony optimization: A revisit and new efficient algorithms // J. Electron. Commun. 2012. V. 2. P. 121–129.

15. Tong C. J., Lau H. C., Lim A. Ant colony optimization for the Ship Berthing Problem // Advances in computing science — ASIAN'99. Proc. 5th Asian Computing Science Conf. (Phuket, Thailand, Dec. 10–12, 1999). Heidelberg: Springer, 1999. P. 359–370. (Lect. Notes Comput. Sci.; V. 1742). DOI: 10.1007/3-540-46674-6_30.

16. Kayaaslan E., Lambert T., Marchal L., Uçar B. Scheduling series-parallel task graphs to minimize peak memory // Theor. Comput. Sci. 2018. V. 707. P. 1–23.

17. Canon L.-C., El Sayah M., Héam P.-C. A comparison of random task graph generation methods for scheduling problems // Euro-Par 2019: Parallel processing. Proc. 25th Int. Conf. Parallel and Distributed Computing (Göttingen, Germany, Aug. 26–30, 2019). Cham: Springer, 2019. P. 61–73. (Lect. Notes Comput. Sci.; V. 11725). DOI: 10.1007/978-3-030-29400-7_5.

Балашов Василий Викторович
Абрамов Алексей Владимирович
Чупахин Андрей Андреевич
Туркин Андрей Владимирович
Гао Цзесин
Сунь Шумин
Чжоу Ли
Сунь Цзе

Статья поступила
19 декабря 2022 г.
После доработки —
25 октября 2023 г.
Принята к публикации
22 декабря 2023 г.

DISKRETNYI ANALIZ I ISSLEDOVANIE OPERATSII
/DISCRETE ANALYSIS AND OPERATIONS RESEARCH/
April–June 2024. Vol. 31, No. 2. P. 5–27

UDC 519.854.2 DOI: 10.33048/daio.2024.31.760

ANT COLONY ALGORITHM FOR SINGLE PROCESSOR SCHEDULING WITH MINIMIZATION OF PEAK RESOURCE USAGE

V. V. Balashov[1, a], *A. V. Abramov*[1, b], *A. A. Chupakhin*[1, c], *A. V. Turkin*[2, d], *J. Gao*[2, e], *C. Sun*[3, f], *L. Zhou*[3, g], and *J. Sun*[3, h]

[1] Lomonosov Moscow State University,
1 Leninskie Gory, 119991 Moscow, Russia
[2] Huawei Moscow Research Center,
7-9 Smolenskaya Square, 121099 Moscow, Russia
[3] Huawei Hong Kong Research Center,
8/F, 2 Science Park West Avenue, 999077 Hong Kong, PRC
E-mail: [a] hbd@cs.msu.ru, [b] avabramov@lvk.cs.msu.ru,
[c] andrewchup@lvk.cs.msu.ru, [d] andrei.turkin@huawei.com,
[e] gaojiexing@huawei.com, [f] sunchumin@huawei.com,
[g] zhouli107@huawei.com, [h] j.sun@huawei.com

Abstract. We consider the problem of constructing a single processor task schedule with minimization of peak resource usage. An example of the resource is the main memory of the target computer. Task set to be scheduled is represented as a directed acyclic graph every node of which is marked with the amount of resource used by the corresponding task. The resource allocated to a task is released on completion of the last (according to the schedule) immediate successor of this task in the graph. Correctness constraint on the schedule is the partial order specified by the task graph. Task duration values are not considered. The formal statement of the problem is provided. To solve the problem, we propose an ant colony algorithm modified so that the pheromone matrix reflects the desirability of pairwise order in the schedule for every pair of tasks, not only for pairs of adjacent tasks. During the schedule construction, for every task the algorithm chooses its position in the schedule, in contrast to existing ant colony scheduling algorithms that construct schedule in increasing order of positions (left-to-right) choosing a task for every next position. Experimental evaluation of

English transl.: *Journal of Applied and Industrial Mathematics* **18** (2), 192–205 (2024), DOI: 10.1134/S1990478924020029.

the algorithm was conducted on two sets of task graphs. The first set contains graphs generated in such a way that the estimation for the optimum value of the goal function is known a priori. Graphs from the second set are "layered," and their structure corresponds to the structure of multistage data processing applications. In both sets, the graphs are generated randomly with respect to specified generation parameters and constraints on the graph structure. The experiments indicate high precision and stability of the proposed ant colony algorithm. Tab. 1, illustr. 12, bibliogr. 17.

Keywords: combinatorial optimization, single-processor schedule, resource minimization, ant colony algorithm.

References

1. **M. R. Garey** and **D. S. Johnson,** *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, 1979; Mir, Moscow, 1982 [Russian]).
2. **H. M. Abdel-Wahab,** Scheduling with applications to register allocation and deadlock problems, *PhD Thesis* (Univ. Waterloo, Waterloo, ON, 1976).
3. **R. Sethi,** Complete register allocation problems, *SIAM J. Comput.* **4** (3), 226–248 (1975).
4. **A. Bauer, C. Straus,** and **R. F. Hartle,** Minimizing total tardiness on a single machine using ant colony optimization, *Cent. Eur. J. Oper. Res. Econ.* **8** (2), 125–141 (2000).
5. **E. R. Gafarov,** A hybrid algorithm for solution to the total delay minimization problem with one device, *Inf. Tekhnol.,* No. 1, 30–37 (2007) [Russian].
6. **S. Jha,** Balanced ant colony algorithm for scheduling DAG to grid heterogeneous system, *Int. J. Sci. Eng. Res.* **2** (6) (2011).
7. **P. Myszkowski, M. Skowronski, L. Olech,** and **K. Oslizlo,** Hybrid ant colony optimization in solving multi-skill resource-constrained project scheduling problem, *Soft Comput.,* No. 19, 3599–3619 (2015).
8. **V. A. Kostenko** and **A. V. Plakunov,** Ant algorithms for scheduling computations in data centers, *Vestn. Mosk. Univ., Ser. 15,* No. 1, 44–50 (2017) [Russian] [*Mosc. Univ. Comput. Math. Cybern.* **41** (1), 44–50 (2017)].
9. **C. Gagne, W. L. Price,** and **M. Grave,** Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times, *J. Oper. Res. Soc.* **53**, 895–906 (2002).
10. Gurobi Optimizer (Gurobi Optimization, 2023), URL: `www.gurobi.com/solutions/gurobi-optimizer` (accessed Mar. 22, 2024).
11. MOSEK Solver (MOSEK ApS, 2023), URL: `www.mosek.com/products/mosek` (accessed Mar. 22, 2024).
12. SCIP: Solving Constraint Integer Programs (Zuse Inst. Berlin, Berlin, 2023), URL: `www.scipopt.org` (accessed Mar. 22, 2024).
13. COIN-OR: Computations Infrastructure for Operations Research. Projects by Category (COIN-OR Found., Towson, 2023), URL: `www.coin-or.org/projects` (accessed Mar. 22, 2024).

14. H. H. Xuan, N. Linh-Trung, D. D. Dong, and **T. Huynh,** Solving the Traveling Salesman Problem with ant colony optimization: A revisit and new efficient algorithms, *J. Electron. Commun.* **2**, 121–129 (2012).

15. C. J. Tong, H. C. Lau, and **A. Lim,** Ant colony optimization for the Ship Berthing Problem, in *Advances in Computing Science — ASIAN'99* (Proc. 5th Asian Computing Science Conf., Phuket, Thailand, Dec. 10–12, 1999) (Springer, Heidelberg, 1999), pp. 359–370 (Lect. Notes Comput. Sci., Vol. 1742), DOI: 10.1007/3-540-46674-6_30.

16. E. Kayaaslan, T. Lambert, L. Marchal, and **B. Uçar,** Scheduling series-parallel task graphs to minimize peak memory, *Theor. Comput. Sci.* **707**, 1–23 (2018).

17. L.-C. Canon, Sayah M. El, and **P.-C. Héam,** A comparison of random task graph generation methods for scheduling problems, in *Euro-Par 2019: Parallel Processing* (Proc. 25th Int. Conf. Parallel and Distributed Computing, Göttingen, Germany, Aug. 26–30, 2019) (Springer, Cham, 2019), pp. 61–73 (Lect. Notes Comput. Sci., Vol. 11725), DOI: 10.1007/978-3-030-29400-7_5.

Vasily V. Balashov
Aleksey V. Abramov
Andrey A. Chupakhin
Andrey V. Turkin
Jiexing Gao
Chumin Sun
Li Zhou
Jie Sun

Received December 19, 2022
Revised October 25, 2023
Accepted December 22, 2023