Е.Р. Гафаров, Москва, ВЦ РАН УДК 519.854.2

Гибридный алгоритм решения задачи минимизации суммарного запаздывания для одного прибора

Для NP-трудной в обычном смысле задачи теории расписаний минимизация суммарного запаздывания для одного прибора построен *Гибридный* алгоритм, использующий идею известного метаэвристического алгоритма "Муравьиные колонии"и комбинаторные свойства Правил исключения 1-4. Приводится сравнительный анализ эффективности *Гибридного* алгоритма и алгоритма "Муравьиные колонии".

Введение

На практике зачастую встречаются задачи, для которых сложно построить эффективный алгоритм без привлечения математического аппарата. Задача, рассматриваемая в данной работе является NP-трудной, то есть для любого алгоритма существует такая вариация входных параметров, что алгоритм будет работать "неприемлемо долго"(в предположении $P \neq NP$).

В связи с этим особую практическую ценность имеют алгоритмы, позволяющие за приемлемое время получать "хорошее" решение.

В работе рассматривается классическая NP-трудная в обычном смысле задача теории расписаний минимизация суммарного запаздывания для одного прибора [1]. Необходимо обслужить n требований на одном приборе. Прерывания при обслуживании и обслуживание более одного требования в любой момент времени запрещены. Для требования $j \in N = \{1, 2, \ldots, n\}$ заданы продолжительность обслуживания $p_j > 0$ и директивный срок его окончания d_j, где N – множество требований, которые необходимо обслужить. Задан момент времени t_0, с которого прибор готов начать обслуживание требований, поступающих одновременно. Без ограничения общности будем полагать, что $t_0 = 0$. Расписание обслуживания требований π строится с момента времени t_0 и однозначно задаётся перестановкой элементов множества N.

Требуется построить расписание π^* обслуживания требований множества N, при котором достигается минимум функции $F(\pi) = \sum_{j=1}^{n} \max\{0, c_j(\pi) - d_j\}$, где $c_j(\pi)$ – момент завершения обслуживания требования j при расписании π. Пусть $\pi = (j_1, j_2, \ldots, j_n)$, тогда $c_{j_1}(\pi) = t_0 + p_{j_1}$ и $c_{j_k}(\pi) = c_{j_{k-1}}(\pi) + p_{j_k}$ для $k = 2, 3, \ldots, n$. Величина $T_j(\pi) = \max\{0, c_j(\pi) - d_j\}$ называется *запаздыванием* требования j при расписании π, а $F(\pi)$ - *суммарным запаздыванием* требований при расписании π.

Можно дать несколько практических интерпретаций рассматриваемой задачи. К примеру, работа аэропорта. Имеется одна взлетно-посадочная полоса. По техническим причинам вовремя не были "посажены" несколько рейсов. То есть некоторые самолеты будут запаздывать (у каждого из них назначено точное время прилета). Задача диспетчера – составить последовательность "посадки" самолетов, чтобы минимизировать суммарное запаздывание.

Можно дать и другую интерпретацию задачи. Представим производственную линию. Поступают заказы, которые имеют конкретный объем (а следовательно, продолжительность изготовления) и директивные сроки исполнения. Может случиться, что заказов было набрано слишком много и некоторые из них обязательно не будут исполнены в срок, что грозит выплатой неустойки. Задача - сократить суммарное запаздывание исполнения заказов, упорядочив их по времени.

Исследуемая задача является NP-трудной в обычном смысле [2]. Лаулер [3] предложил псевдополиномиальный алгоритм решения общего случая проблемы трудоемкости $O(n^4 \sum p_j)$. Шварц и др. построили [4, 5] алгоритмы решения проблемы, которые были протестированы для примеров $n < 600$ (тестовые примеры Поттса и Ван Вассенхова [6]). Исследование приближенных алгоритмов решения проблемы было проведено в работе [7], где построены примеры, на которых известные приближенные алгоритмы находят решение с относительной погрешностью порядка размерности примера n.

Под гибридным (комбинированным) алгоритмом дискретной оптимизации будем понимать алгоритм, в котором на различных этапах вычислительного процесса имеется возможность выбора на программном уровне одного из нескольких алгоритмов для решения исходной задачи или ее подзадач [10].

На основе известного алгоритма "Муравьиные колонии" – *Ant Colony Optimization*(ACO)[11] и Правил Исключения 1-4[4, 8, 9] нами построен новый *Гибридный алгоритм*. В данной работе приводится сравнительный анализ его эффективности и эффективности алгоритма ACO. В частности, исследовались: среднее количество итераций, необходимых для нахождения оптимального решения, относительная погрешность, процент точно найденных решений.

Экспериментальные исследования проводились на множестве примеров из 3-х классов (тестовые примеры Поттса и Ван Вассенхова[6], примеры случая B-1[8], канонические DL-примеры[2]).

В первом параграфе приводятся Правила исключения 1-4 и точный алгоритм решения для общего случая задачи.

Алгоритм ACO и *Гибридный* алгоритм описаны, соответственно, во втором и третьем параграфах.

Результаты экспериментальных исследований приводятся в параграфах 4-6.

1. Комбинаторные методы решения задачи $1||\sum T_j$

Определения. Расписание $\pi = (j_1, j_2, \ldots, j_n)$ будем называть *EDD-расписанием* (early due date), если $d_{j_k} \leq d_{j_{k+1}}$ (для $d_{j_k} = d_{j_{k+1}}$ выполняется $p_{j_k} \leq p_{j_{k+1}}$), $k = 1, 2, \ldots, n-1$.

Через $x = \langle \{p_j, d_j\}_{j \in N}, t_0 \rangle$ обозначим пример проблемы, для которого требуется построить оптимальное расписание. Подпримером примера x будем называть пару $\{N', t'\}$, где $N' \subseteq N$, $N' \neq \emptyset$, и $t' \geq t_0$.

Без ограничения общности будем предполагать, что требования множества N пронумерованы в порядке неубывания директивных сроков

$$d_1 \leq d_2 \leq \ldots \leq d_n,$$

если $d_k = d_{k+1}$ то $p_k \leq p_{k+1}$.

Через $j^*(N')$ обозначим требование с наибольшей продолжительностью обслуживания среди требований множества $N' \subseteq N$, если таких требований несколько, то выбирается требование с наибольшим директивным сроком, т.е. $j^*(N') = \arg\max\limits_{j \in N'}\{d_j : p_j = \max\limits_{i \in N'} p_i\}$. Для сокращения записи вместо $j^*(N')$ будем записывать j^*, если очевидно о каком множестве идет речь.

Рассмотрим пример (подпример) обслуживания требований множества $N' \subseteq N$, $N' = \{1, 2, \ldots, n'\}$, с момента времени $t' \geq t_0$. Множество $L(N', t')$ есть множество всех индексов $k \in \{1, \ldots, n'\}$, $k \geq j^*(N')$, таких что:

(a) $t' + \sum_{j=1}^{k} p_j < d_{k+1}$ (**правило исключения 1**[5, 8]) и

(b) $d_j + p_j \leq t' + \sum_{j=1}^{k} p_j$, для всех $j = \overline{j^*(N')+1, k}$ (**правила исключения 2,3**[5, 8]),

где $d_{n'+1} := +\infty$.

Теорема 1. [8]Для любого примера (подпримера) $\langle \{p_j, d_j\}_{j \in N}, t_0 \rangle$ множество $L(N, t_0)$ не пусто.□

Лемма 1. [3, 6, 8] Для любого примера $\langle \{p_j, d_j\}_{j \in N}, t_0 \rangle$ существует оптимальное расписание π^* обслуживания требований множества N, при котором $(j \to j^*)_{\pi^*}$ для всех требований $j \in \{1, 2, \ldots, k\} \backslash \{j^*\}$ и $(j^* \to j)_{\pi^*}$ для всех требований $j \in \{k+1, \ldots, n\}$ для некоторого $k \in L(N, t_0)$.□
Через $(i \to j)_\pi$ обозначают, что требование i обслуживается раньше требования j при расписании π.

Пусть $N = (j_1, \ldots, j_n)$, $d_{j_1} \leq \ldots \leq d_{j_n}$. Модифицированное EDD-расписание, при котором требование j^* обслуживается k-м по порядку, обозначим $\pi^k = (j_1, \ldots, j_{m-1}, j_{m+1}, \ldots, j_k, j^*, j_{k+1}, \ldots, j_n)$, $j^* = j_m, m < k$.

Лемма 2. Правило исключения 4 [4, 9]. Если $F(\pi^k) > F(\pi^{k+1})$ или $F(\pi^k) \geq F(\pi^i)$ для некоторого $j^* \leq i < k$, то позиция k исключается из списка "подходящих" позиций $L(N', t')$ для примера $\langle \{p_j, d_j\}_{j \in N'}, t' \rangle$, если $|L(N', t')| > 1$.□

Первые правила доминирования для данной задачи были предложены в работах [1, 13, 14]. Эти правила сокращают множество возможных продолжений каждой последовательности $(j_1, \ldots, j_k)$ в схеме последовательного анализа вариантов, т.е. множество требований j, которые могут обрабатываться непосредственно за j_k. Предложенная схема предполагала последовательное конструирование расписания, когда на шаге i выбиралось требование для постановки в позицию i.

Другого рода правила доминирования были предложены в работе [3]. Правила исключения 1-4 являются усовершенствованными правилами отсеивания вариантов, описанными в [3].

Опишем алгоритм нахождения оптимального расписания на основе Правил исключения 1 – 4.

Процедура ProcL (N, t)

0. Дан пример $\{N, t\}$ с множеством требований $N = \{j_1, j_2, \ldots, j_n\}$ и моментом начала обслуживания t, $d_{j_1} \leq d_{j_2} \leq \ldots \leq d_{j_n}$;

1. IF $N = \emptyset$ **THEN** π^*:= пустое расписание, **GOTO 6.**;

2. Найдем требование $j^*(N, t)$ из множества N;

3. Найдем множество $L(N, t)$ для требования j^*;

4. FOR ALL $k \in L(N, t)$ **DO:**

$\pi_k := (\mathbf{ProcL}(N', t'), j^*, \mathbf{ProcL}(N'', t''))$, где
$N' := \{j_1, \ldots, j_k\} \setminus \{j^*\}$, $t' := t$,
$N'' := \{j_{k+1}, \ldots, j_n\}$, $t'' := t + \sum_{i:=1}^{k} p_{j_i}$;

5. π^*:=$\arg \min\limits_{k \in L(N,t)} \{F(\pi_k, t)\}$;

6. RETURN π^*.

алгоритм A

π^*:=**ProcL**(N, t_0).

2. Алгоритм ACO для задачи $1||\sum T_j$

Для решения многих комбинаторных задач эффективно используются *метаэвристические* методы: **генетические алгоритмы**, **локальный поиск с запретами (Tabu search)**, **муравьиные колонии (Ant Colony Optimization)** и др.

В данной работе рассматривается метод Ant Colony Optimization (ACO) [11]. Этот итерационный метод основан на идее последовательного приближения к оптимальному решению.

Каждая итерация – "запуск искусственного муравья", – который "пытается" по некоторому правилу выбрать наилучший маршрут к "пище" (к оптимуму функции), используя метки своих предшественников.

Каждый муравей выполняет цепочку шагов. На каждом шаге $i = 1, 2, \ldots, n$ выбирается требование j для постановки на место i расписания, используя "вероятности перехода".

В алгоритме используются параметры:

η_{ij} – эвристическая информация о том, насколько хорошим кажется постановка требования j на место i в расписании. Этот параметр вычисляется эвристически по одному из вариантов:

1. По правилу EDD: $\eta_{ij} = \frac{1}{d_j},\ i = 1, \ldots, n$;
2. По правилу MDD (modified due date). В алгоритме MDD последовательно на позиции $i = 1, \ldots, n$ выбирается еще неупорядоченное требование j с наименьшим значением $max\{S + p_j, d_j\}$, где S – сумма продолжительностей предшествующих упорядоченных требований. Эвристическая информация рассчитывается следующим образом: $\eta_{ij} = \frac{1}{max\{S+p_j, d_j\}}$;
3. По правилу L-MDD (Look-ahead MDD): $\eta_{ij} = \frac{1}{Tard_j}$, где $Tard_j$ – суммарное запаздывание при модифицированном расписании MDD, при котором на позиции i обслуживается требование j, а все остальные требования упорядочены в соответствии с правилом MDD;
4. По правилу SPT: $\eta_{ij} = \frac{1}{p_j},\ i = 1, \ldots, n$;

τ_{ij} – "след" (в природе: след феромона). После каждой итерации этот параметр корректируется. Параметр показывает насколько "хорошим" для требования j оказалась позиция i. То есть это накопленная статистическая информация о качестве выбора для позиции i требования j, в то время как η_{ij} характеризует предполагаемую выгоду такой постановки при недостатке накопленной информации.

Под эвристической процедурой понимается процедура, не имеющая формального обоснования, а опирающаяся лишь на анализ специфики структуры задачи и на связанные с ней содержательные соображения [15].

Перед первой итерацией $\tau_{ij} = 1/(mT_{EDD})$, где T_{EDD} – суммарное запаздывание при EDD-расписании. m–количество итераций (муравьев). Параметры η_{ij} рассчитываются один раз перед первой итерацией.

На каждом шаге вычисляется **Матрица вероятностей перехода**:

$$\rho_{ij} = \begin{cases} \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{h\in\Omega}[\tau_{ih}]^\alpha [\eta_{ih}]^\beta}, & j \in \Omega, \\ 0, & j \notin \Omega, \end{cases}$$

где Ω – множество неупорядоченных требований.

Правило, по которому на позицию i выбирается требование j определяется следующим образом:

$$\begin{cases} j = argmax_{h\in\Omega}[\tau_{ih}]^\alpha [\eta_{ih}]^\beta, & q < q_0, \\ j \text{ определяется случайным образом,} \\ \text{согласно распределению вероятностей } \rho_{ij}, & q \geq q_0, \end{cases}$$

где $0 \leq q_0 \leq 1$ – параметр алгоритма, а значение q вычисляется случайным образом на каждом шаге.

После того, как требование j было поставлено на позицию i, пересчитывается "локальный след":

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

где $\tau_0 = 1/(mT_{EDD})$. T_{EDD} – суммарное запаздывание при EDD-расписании. $\rho \in [0,1]$ – коэффициент распада феромона (параметр алгоритма).

После каждой итерации "глобальный след" τ_{ij} корректируется по правилу:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho/T^*,$$

если в "лучшем" найденном расписании на позиции i обслуживается требование j. Иначе

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

Значение T^* – суммарное запаздывание для "лучшего" найденного расписания.

Алгоритм ACO дает хорошие результаты при интеграции в него локального поиска, к примеру, попарной перестановки требований. Локальный поиск запускается после каждой итерации.

В работе [11] приводиться экспериментальная оценка эффективности этого алгоритма. Эксперименты проводились для тестовых примеров [6] при $n = 50, 100$. Для размерности задачи $n = 50$ из 125 примеров неточно был решен только 1. Для размерности $n = 100$ неточно решено 0 примеров из 125. Относительная погрешность не превосходила 0.08%.

В своих экспериментах мы использовали те же значения управляющих параметров и эвристики, как и в работе [11].

Нетрудно убедиться, что трудоемкость алгоритма без локального поиска $O(mn^2)$. Для каждой позиции i (всего n позиций) выбиралось требование j за $O(n)$ операций.

Трудоемкость локального поиска $O(n^3)$. Тогда теоретическая трудоемкость алгоритма ACO с локальным поиском составляет не меньше $O(mn^3)$ операций.

3. Гибридный алгоритм

На основе алгоритма ACO и Правил исключения 1-4 нами построен новый *Гибридный алгоритм*. Идея алгоритма заключается в том, что на каждой итерации запускается модифицированный алгоритм A, в котором очередное требование j^* "случайным" образом ставится на некоторую позицию $k \in L(N, t)$.

"Случайный" выбор происходит согласно методу "Муравьиные колонии", т.е. подзадача выбора подходящей позиции решается алгоритмом ACO.

Модифицированная процедура ProcL (N, t)

0. Дан пример $\{N, t\}$ с множеством требований $N = \{j_1, j_2, \ldots, j_n\}$ и моментом начала обслуживания t, $d_{j_1} \leq d_{j_2} \leq \ldots \leq d_{j_n}$;

1. **IF** $N = \emptyset$ **THEN** π^*:= пустое расписание, **GOTO 7.**;

2. Найдем требование $j^*(N, t)$ из множества N;

3. Найдем множество $L(N, t)$ для требования j^*;

4. Рассчитаем матрицу вероятностей перехода для каждой позиции $i \in L(N,t)$.

$$\rho_{ij*} = \frac{\tau_{ij*} 1/F(\pi^i, t)}{\sum_{h \in L(N,t)} \tau_{hj*} 1/F(\pi^h, t)},$$

где $\pi^i = (j_1, \ldots, j_{m-1}, j_{m+1}, \ldots, j_i, j^*, j_{i+1}, \ldots, j_n)$, $j^* = j_m, m < i$, $F(\pi^i, t)$ – суммарное запаздывание при расписании π^i, построенном с момента времени t.

5. Выберем позицию $k \in L(N,t)$ произвольным образом согласно распределению вероятностей ρ_{ij*}.

6. Пересчитаем "локальный след":

$$\tau_{kj*} = (1-\rho)\tau_{kj*} + \rho\tau_0,$$

где $\tau_0 = 1/(mT_{EDD})$. T_{EDD} – суммарное запаздывание при EDD-расписании.

7. RETURN

$\pi^* :=$(**ProcL**(N', t'), j^*,**ProcL**(N'', t'')), где
$N' := \{j_1, \ldots, j_k\} \setminus \{j^*\}$, $t' := t$,
$N'' := \{j_{k+1}, \ldots, j_n\}$, $t'' := t + \sum_{i:=1}^{k} p_{j_i}$.

После каждой итерации "глобальный след" τ_{ij} корректируется по правилу:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho/T^*,$$

если в "лучшем" найденном расписании на позиции i обслуживается требование j. Иначе

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

где $\rho \in [0,1]$ – параметр алгоритма. Значение T^* – суммарное запаздывание для "лучшего" найденного расписания.

После каждой итерации запускается процедура локального поиска – попарная перестановка требований.

Идея выбора подходящей позиции случайным образом (тем самым в методе ветвей и границ выбирается только одна ветвь из многих) не нова.

Шкурба и др.[13] предложили схему вероятностного моделирования для задач календарного планирования, согласно которой просматривается ограниченное подмножество "подходящих" вариантов, выбираемых неким случайным образом. Если в предлагаемом *Гибридном* алгоритме не учитывать параметр τ_{ij}, тем самым не накапливая статистику о приоритетности той или иной подходящей позиции, то мы получим алгоритм вероятностного моделирования.

Лазарев [8] предложил модифицированный алгоритм А для рассматриваемой задачи, в котором выбор подходящей позиции происходит случайным образом без накопления статистики.

Нетрудно убедиться, что трудоемкость *Гибридного* алгоритма без локального поиска $O(mn^2)$. Процедура ProcL запускается n раз. В каждой процедуре выполняется порядка $O(n)$ действий.

Трудоемкость локального поиска $O(n^3)$. Тогда теоретическая трудоемкость *Гибридного* алгоритма с локальным поиском составляет не меньше $O(mn^3)$ операций. То есть трудоемкость исследуемых алгоритмов идентична.

4. Эффективность алгоритмов для тестовых примеров [6]

Вычислительные эксперименты, представленные в этом параграфе, проводились для примеров размерности $n = 4, \ldots, 70, 100$, которые генерировались с помощью схемы Поттса и Ван Вассенхова [6].

Примеры генерировались следующим образом. Значения $p_j \in Z$ распределены по равномерному закону на промежутке $[1, 100]$. Значения d_j генерировались по равномерному закону на промежутке

$$[\sum_{j:=1}^{n} p_j(1 - TF - RDD/2), \sum_{j:=1}^{n} p_j(1 - TF + RDD/2)].$$

Параметры TF И RDD принимают значения из множества $\{0.2, 0.4, 0.6, 0.8, 1\}$ [4, 5, 11]. Для каждой комбинации (TF, RDD) генерировалось по 100 примеров (всего 2500 примеров для каждого n).

Примеры, для которых $F(\pi_{EDD}) = 0$, не рассматривались, т.к. в этом случае *Гибридный* алгоритм и ACO гарантированно находят точные решения. π_{EDD}–расписание построенное по правилу EDD.

В алгоритмах использовались следующие параметры: $\alpha = 1; \beta = 2; \rho = 0{,}1$. Локальный поиск – попарная перестановка. Применяемая эвристика – MDD [11].

Для каждого примера запускался точный алгоритм A. Мы получали оптимальное значение целевой функции F_{opt}.

В алгоритме ACO муравьи генерировались до тех пор, пока не находилось расписание π, при котором $F(\pi) = F_{opt}$. Это "необходимое" количество муравьев фиксировалось. Число муравьев нами было ограничено $m \leq 100$. Поэтому перед первой итерации параметры τ_{ij} рассчитывались по формуле $\tau_{ij} = 1/(T_{EDD}),\ i = 1, \ldots, n, j = 1, \ldots, n$, $\tau_0 = 1/(T_{EDD})$.

Алгоритм запускался до 10 раз (пока не найдено оптимальное расписание). Таким образом мы пытались избежать "случайности" работы алгоритма.

Лучшее найденное значение целевой функции F_{ACO} фиксировалось. Вычислялась относительная погрешность $\frac{F_{ACO} - F_{opt}}{F_{opt}}$.

Такая же схема использовалась для *Гибридного* алгоритма. Таким образом мы могли сравнить относительную погрешность двух алгоритмов (ACO и *Гибридного*), количество муравьев, необходимое каждому алгоритму для поиска оптимального расписания.

Результаты экспериментов представлены в табл.1.

Таблица 1.

Результаты экспериментов на примерах Поттса и Ван Вассенхова.

n	Не опт. ACO	Не опт. Гиб.	Отн.п. ACO	Отн.п. Гиб.	Мур. ACO	Мур. Гиб.
19	2	0	0.22	0	1.6164	1.4004
20	1	0	0.58	0	1.6064	1.4204
22	1	0	0.16	0	1.626	1.4844
28	2	0	0.09	0	1.9704	1.6688
34	1	0	0.15	0	2.2212	1.9568
36	0	1	0	0.04	2.2332	2.154
37	1	1	0.38	0.01	2.4796	2.102

n	Не опт. АСО	Не опт. Гиб.	Отн.п. АСО	Отн.п. Гиб.	Мур. АСО	Мур. Гиб.
40	1	0	0.04	0	2.6036	2.2424
42	1	1	0.05	0.01	2.7888	2.4092
43	1	1	0.07	0.06	2.7316	2.3656
44	3	0	0.04	0	2.8464	2.3784
45	2	0	0.68	0	2.9736	2.4728
46	1	0	0.03	0	3.1624	2.4088
47	2	0	0.01	0	3.248	2.5152
48	9	0	0.56	0	3.4516	2.5196
49	3	1	0.15	0.08	3.4252	2.7
50	9	1	0.35	0.29	3.716	2.6336
51	8	0	0.22	0	3.8412	2.7768
52	4	1	0.04	0.07	3.5816	2.86
53	4	2	0.03	0.42	3.8948	2.9668
54	9	3	0.1	0.29	4.0324	2.9924
55	8	2	0.11	0.06	4.1048	3.0496
56	9	1	0.83	0.01	4.2916	3.0064
57	7	0	0.23	0	4.1568	3.158
58	14	0	0.17	0	4.71	3.3724
59	14	4	0.24	0.1	4.81	3.3372
60	11	1	0.22	0.01	4.7268	3.4224
61	18	2	1.26	0.02	5.3032	3.5216
62	10	2	0.26	0.01	5.0964	3.5032
63	17	7	0.16	0.08	5.3016	3.5728
64	15	6	0.57	0.46	5.2388	3.6504
65	18	7	0.1	0.14	5.548	3.6604
66	17	11	0.15	0.14	5.4288	3.8552
67	17	7	0.83	0.1	6.1068	4.1016
68	25	4	0.2	0.08	6.3864	3.7252
69	18	6	0.12	0.1	6.1912	4.0796
70	33	4	0.23	0.05	6.974	3.8672
100	36	0	0.31	0	27.35	4.66

Для размерности задачи $n = 100$ рассмотрено 617 примеров. Для остальных размерностей по 2500 примеров. В таблице выводятся только те строки, соответствующие размерности задачи n, для которых количество неточно решенных примеров больше 0.

В первой колонке представлено значение n – размерность задачи. Во второй и третьей колонках, соответственно, количество примеров, для которых алгоритм ACO или *Гибридный* не нашли точные решения. В четвертой и пятой колонках представлена максимальная относительная погрешность алгоритмов (с точностью до сотых процента). В последних двух колонках – среднее количество муравьев, необходимое для поиска оптимального решения.

Как видно из таблицы, примерно для 99% примеров оба алгоритма находят точные решения.

Количество примеров, неточно решенных *Гибридным* алгоритмом, значительно меньше, чем в алгоритме ACO, и не превосходит 0.44% от общего числа примеров. Относительная погрешность не превосходит 0.46%. Количество муравьев, необходимое *Гибридному* алгоритму также меньше. Заметно, что для $n = 100$ *Гибридному* алгоритму в среднем необходимо 4-5 муравьев для нахождения оптимального расписания, в то время как алгоритму ACO требуется в среднем более 27 муравьев.

Относительная погрешность алгоритма ACO превосходит 1.26% для $n = 61$. Количество "неточно" решенных примеров для $n = 70$ больше 1%.

Следует ожидать, что с ростом n будет наблюдаться значительное преимущество *Гибридного* алгоритма.

5. Эффективность алгоритмов для случая B-1 [8]

В этом параграфе представлены экспериментальные исследования алгоритмов для примеров случая B-1. Параметры требований в этом случае задаются следующим образом:

$$\begin{cases} p_1 \geq p_2 \geq \ldots \geq p_n, \\ d_1 \leq d_2 \leq \ldots \leq d_n, \\ d_n - d_1 \leq p_n. \end{cases} \tag{1}$$

Экспериментальные исследования алгоритма A показали, что для этого случая дерево поиска имеет наибольшее количество ветвлений.

В [16] представлено доказательство NP-трудности в обычном смысле случая B-1.

Эксперименты, аналогичные экспериментам из предыдущего параграфа, проводились для примеров размерности $n = 4, \ldots, 100$. Для каждого значения n рассматривалось по 1000 примеров случая B-1.

Значения p_j генерировались по равномерному закону на промежутке $[1, 500]$. Директивные сроки d_j распределены равномерно на интервале $[A, A + p_n]$, где $A \in [0, \sum p_j - p_n]$.

Схема проведения экспериментов и параметры алгоритмов описаны в предыдущем параграфе. В качестве точного алгоритма использовался алгоритм B-1 модифицированный, трудоёмкость которого не больше $O(n \sum p_j)$ для целочисленных примеров.

Были получены результаты, представленные в табл.2.

Таблица 2.
Результаты экспериментов на примерах B-1.

n	Не опт. АСО	Не опт. Гиб.	Отн.п. АСО	Отн.п. Гиб.	Мур. АСО	Мур. Гиб.
7	1	0	0.67	0	1.38	1.044
26	0	3	0	0.01	1.381	1.871
27	0	1	0	0.01	1.429	1.707
31	0	4	0	0.01	1.354	1.929

В таблице выводятся только те строки, соответствующие размерности задачи n, для которых количество неточно решенных примеров больше 0 и относительная погрешность больше или равна 0.01%.

Алгоритм АСО находит точное решение практически для всех примеров. Единственный неточно решенный пример встретился для $n = 7$. *Гибридный* алгоритм находит точное решение для 99% примеров. Причем относительная погрешность *Гибридного* алгоритма не превосходит 0.01%.

В среднем обоим алгоритмам требуется не более двух муравьев, чтобы найти точное решение.

По результатам экспериментов эффективность *Гибридного* алгоритма, использующего идеи алгоритма A, ниже эффективности алгоритма АСО на примерах B-1. Можно объяснить это тем, что примеры B-1 "наиболее сложные" для алгоритма A.

Заметим так же, что при такой генерации примеров, трудоемкость точного алгоритма B-1 модифицированный меньше $O(n^3)$, так как генерацией "не захватывается" NP-трудный подслучай.

6. Эффективность алгоритмов для канонических DL-примеров [2]

NP-трудность канонических DL-примеров доказана сведением NP-полной проблемы Четно-Нечетного Разбиения(ЧНР) к проблеме $1||\sum T_j$ [2].

Постановка задачи ЧНР.

Задано упорядоченное множество из $2n$ положительных целых чисел $B = \{b_1, b_2, \ldots, b_{2n}\}$, $b_i > b_{i+1}$, $1 \le i \le 2n-1$. Требуется определить, существует ли разбиение множества B на два подмножества B_1 и B_2, такое, что $\sum_{b_i \in B_1} b_i = \sum_{b_i \in B_2} b_i$ и для каждого $i = 1, 2, \ldots, n$ подмножество B_1 (следовательно, и B_2) содержит в точности один элемент из пары $\{b_{2i-1}, b_{2i}\}$.

Приведем полиномиальную схему сведения ЧНР к проблеме $\mathbf{1} \;||\; \sum \boldsymbol{T_j}$ [2]. Определим $\delta = \sum_{i=1}^{n}(b_{2i-1} - b_{2i})$.

Пусть $a_{2i-1} = b_{2i-1} + (9n^2 + 3n - i + 1)\frac{1}{2}\delta + 5n(b_1 - b_{2n})$ и
$a_{2i} = b_{2i} + (9n^2 + 3n - i + 1)\frac{1}{2}\delta + 5n(b_1 - b_{2n})$, $i = 1, \ldots, n$.

Построим *канонический DL-пример* [2] проблемы $\mathbf{1} \;||\; \sum \boldsymbol{T_j}$ для множества из $3n+1$ требований $N = \{V_1, V_2 \ldots, V_{2n}, W_1, W_2, \ldots, W_{n+1}\}$. Пусть $b = (4n+1)\frac{1}{2}\delta$. Зададим параметры требований следующим образом:

$$p_{V_i} = a_i, \qquad i = 1, 2, \ldots, 2n,$$

$$p_{W_i} = b, \qquad i = 1, 2, \ldots, n+1,$$

$$d_{V_i} = \begin{cases} (j-1)b + \frac{1}{2}\delta + (a_2 + a_4 + \ldots + a_{2i}), & \text{если } i = 2j-1, \\ d_{V_{2j-1}} + 2(n-j+1)(a_{2j-1} - a_{2j}), & \text{если } i = 2j, \\ & j = 1, 2, \ldots, n, \end{cases}$$

$$d_{W_i} = \begin{cases} ib + (a_2 + a_4 + \ldots + a_{2i}) & \text{если } i = 1, 2, \ldots, n, \\ d_{W_n} + \frac{1}{2}\delta + b & \text{если } i = n+1. \end{cases}$$

Обозначим $\delta_i = b_{2i-1} - b_{2i},\ i = 1, \ldots, n$.

Мы генерировали примеры ЧНР для $n = 4, \ldots, 40$. Параметры δ_i распределены по равномерному закону на промежутке $[1, 50]$. Пример ЧНР задается следующим образом $b_{2n} := 1,\ b_{2n-1} := b_{2n} + \delta_n,\ b_{2i} := b_{2i+1} + 1,\ b_{2i-1} := b_{2i} + \delta_i,\ i := 1, \ldots, n-1$.

За полиномиальное время мы преобразовывали пример ЧНР к каноническому DL-примеру (количество требований $3n + 1 = 13, 16, \ldots, 121$). Для канонического DL-примера запускался точный псевдополиномиальный алгоритм B-1 канонический, трудоемкостью $O(n\delta)$, алгоритмы ACO и *Гибридный*. Параметры алгоритмов прежние.

Для каждого значения n генерировалось по 50 примеров. Каждый алгоритм (*Гибридный* и ACO) запускался 1 раз.

Были получены результаты, представленные в табл.3.

Таблица 3.
Результаты экспериментов на примерах DL.

$3n+1$	Не опт. ACO	Не опт. Гиб.	Отн.п. ACO	Отн.п. Гиб.	Мур. ACO	Мур. Гиб.
16	1	0	0	0	4.92	3.4
19	2	3	0	0	10.3	11.64
22	2	2	0	0	7.66	8.22
25	4	4	0	0	16.28	16.44
28	6	4	0	0	19.2	15.24
31	0	3	0	0	8.16	15.66
34	1	1	0	0	8.8	9.44
37	1	0	0	0	7.12	7.58

В таблице выводятся только те строки, соответствующие размерности задачи $3n + 1$, для которых количество неточно решенных примеров больше 0.

Эффективность алгоритмов примерно идентична. Только для $3n + 1 = 25, 28$ количество неточно решенных примеров достигает 10%. При этом погрешность алгоритмов меньше 0.01%. Среднее необходимое количество итераций для нахождения точного решения не превышает 20. Стоит отметить, что эти метаэвристические алгоритмы находят точное решение для канонических DL-примеров размерности $3n + 1 = 121$. Для решения таких примеров необходимо перебрать порядка 2^n канонических

DL-расписаний[2]. В *Гибридном* алгоритме для каждой пары требований $V_{2i-1}, V_{2i},\ i := n, \ldots, 1$ рассматривается только 2 порядка обслуживания: требование V_{2i-1} обслуживается на позиции $2i-1$, а требование V_{2i} обслуживается на позиции $3n+1-(i-1)$ и наоборот. Причем вероятности постановки в эти позиции примерно равны $1/2$ на каждой итерации. То есть позиции "равновероятны". Следовательно, шанс найти точное решение стремиться к величине $(1/2)^n$.

Как было замечено, на некоторых примерах, для которых не найдено оптимальное расписание, количество запусков локального поиска превышает n^2. То есть на локальный поиск затрачено гораздо больше времени, чем на основную процедуру поиска решения.

В работе [17] приводится доказательство, что если $P \neq NP$, то никакой алгоритм локального поиска для NP-трудной задачи с полиномиальной сложностью каждой итерации не может быть точным.

Мы повторили эксперимент на тех же примерах, при тех же условиях, отключив в алгоритмах "локальный поиск". Были получены результаты, представленные в табл.4.

Таблица 4.
Результаты экспериментов на примерах DL без локального поиска.

$3n+1$	Не опт. АСО	Не опт. Гиб.	Отн.п. АСО	Отн.п. Гиб.	Мур. АСО	Мур. Гиб.
13	50	26	13.46	0.01	100	58.2
16	50	35	0.77	0.03	100	73.82
19	50	43	3.29	2.78	100	88.06
22	50	46	2.86	2.05	100	93.52
25	50	49	2.03	1.58	100	98.02
28	50	50	2.97	2.49	100	100
31	50	49	3.48	2.02	100	98.48
34	50	49	2.85	1.67	100	98.4
37	50	49	1.71	1.41	100	98.02

В таблице не выводятся строки, соответствующие размерности задачи $3n+1$, для которых оба алгоритма неточно решают все примеры.

Анализируя обе таблицы можно сделать вывод, что эффективность алгоритмов на данном классе примеров достигается в основном за счет локального поиска.

При $3n+1 \geq 40$ оба алгоритма без локального поиска неточно решают все примеры.

Заключение

По результатам экспериментов, *Гибридный* алгоритм существенно эффективнее алгоритма ACO на тестовых примерах Поттса и Ван Вассенхова. Для 99.5% примеров *Гибридным* алгоритмом найдено точное решение. Относительная погрешность не превосходит 0.5 %. Среднее необходимое количество муравьев не превосходит 5.

На "трудных" примерах случая B-1 *Гибридный* алгоритм уступает в точности алгоритму ACO, но находит точное решение более чем для 99% примеров. Относительная погрешность не превосходит 0.01 %.

Для NP-трудных канонических DL-примеров "хорошая эффективность" алгоритмов достигается за счет локального поиска.

Список литературы

1. **Танаев В.С., Шкурба В.В.**, Введение в теорию расписаний. М: Наука, 1975.

2. **Du J.** and **Leung J. Y.-T.**, *Minimizing total tardiness on one processor is NP-hard* // Math. Oper. Res.. 1990. № 15. P. 483–495.

3. **Lawler E.L.**, *A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness* // Ann. Discrete Math.. 1977. № 1. P. 331–342.

4. **Szwarc W.,Della Croce F.** and **Grosso A.**, *Solution of the single machine total tardiness problem* // Journal of Scheduling. 1999. № 2. P. 55–71.

5. **Szwarc W., Grosso A**. and **Della Croce F.**, *Algorithmic paradoxes of the single machine total tardiness problem* // Journal of Scheduling. 2001. № 4. P. 93-104.

6. **Potts C.N.** and **Van Wassenhove L.N.**, *A decomposition algorithm for the single machine total tardiness problem* // Oper. Res. Lett.. 1982. № 1. P. 177–182.

7. **Della Croce F., Grosso A., Paschos V.**, *Lower bounds on the approximation ratios of leading heuristics for the single-machine total tardiness problem* // Journal of Scheduling. 2004. № 7. P. 85–91

8. **Lazarev A., Kvaratskhelia A., Tchernykh A.**, *Solution algorithms for the total tardiness scheduling problem on a single machine* // Workshop Proceedings of the ENC'04 International Conference. 2004. P. 474–480.

9. **Chang S., Lu Q., Tang G., Yu W.**, *On decomposition of total tardiness problem* // Oper. Res. Lett.. 1995. № 17. P. 221–229.

10. **Корбут А.А., Сигал И.Х., Финкельштейн Ю.Ю.**, *Гибридные методы в дискретной оптимизации.* // Известия АН СССР. Техническая кибернетика. 1988. № 1. P. 65-78.

11. **Bauer A., Bullnheimer B., Hartl R.F., Strauss C.**, *Minimizing Total Tardiness on a Single Machine Using Ant Colony Optimization.* // Proceedings of the 1999 Congress on Evolutionary Computation (CEC99), 6-9 July Washington D.C., USA. 1999. P. 1445–1450.

12. **Merkle D., Middendorf M.**, *An Ant Algorithm with a New Pheromone Evaluation Rule for Total Tardiness Problem.* // EvoWorkShops 2000, LNCS 1803, Springer-Verlag. 2000. P. 287–296.

13. **Шкурба В.В., Подчасова Т.П., Пшичук А.Н., Тур Л.П.**, Задачи календарного планирования и методы их решения. Киев: Наукова думка, 1966. 154 с.

14. **Emmons H.**, *One machine sequencing to minimize certain functions of job tardiness* // Oper. Res., 17. 1969. P. 701–715.

15. **Корбут А.А., Финкельштейн Ю.Ю.**,*Приближенные методы дискретного программирования* // Известия АН СССР. Технич. кибернетика, 1. 1983. P. 165–176.

16. **Лазарев А.А., Гафаров Е. Р.**, *Доказательство NP-трудности частного случая задачи минимизация суммарного запаздывания.* // Известия РАН:Теория и системы управления. 2006. №3. (в печати).

17. **Пападимитриу Х., Стайглиц К.**, Комбинаторная оптимизация. Алгоритмы и сложность. М: Мир, 1985. 510с.

Таблица 1.
Результаты экспериментов на примерах Поттса и Ван Вассенхова.

n	Не опт. АСО	Не опт. Гиб.	Отн.п. АСО	Отн.п. Гиб.	Мур. АСО	Мур. Гиб.
19	2	0	0.22	0	1.6164	1.4004
20	1	0	0.58	0	1.6064	1.4204
22	1	0	0.16	0	1.626	1.4844
28	2	0	0.09	0	1.9704	1.6688
34	1	0	0.15	0	2.2212	1.9568
36	0	1	0	0.04	2.2332	2.154
37	1	1	0.38	0.01	2.4796	2.102

n	Не опт. АСО	Не опт. Гиб.	Отн.п. АСО	Отн.п. Гиб.	Мур. АСО	Мур. Гиб.
40	1	0	0.04	0	2.6036	2.2424
42	1	1	0.05	0.01	2.7888	2.4092
43	1	1	0.07	0.06	2.7316	2.3656
44	3	0	0.04	0	2.8464	2.3784
45	2	0	0.68	0	2.9736	2.4728
46	1	0	0.03	0	3.1624	2.4088
47	2	0	0.01	0	3.248	2.5152
48	9	0	0.56	0	3.4516	2.5196
49	3	1	0.15	0.08	3.4252	2.7
50	9	1	0.35	0.29	3.716	2.6336
51	8	0	0.22	0	3.8412	2.7768
52	4	1	0.04	0.07	3.5816	2.86
53	4	2	0.03	0.42	3.8948	2.9668
54	9	3	0.1	0.29	4.0324	2.9924
55	8	2	0.11	0.06	4.1048	3.0496
56	9	1	0.83	0.01	4.2916	3.0064
57	7	0	0.23	0	4.1568	3.158
58	14	0	0.17	0	4.71	3.3724
59	14	4	0.24	0.1	4.81	3.3372
60	11	1	0.22	0.01	4.7268	3.4224
61	18	2	1.26	0.02	5.3032	3.5216
62	10	2	0.26	0.01	5.0964	3.5032
63	17	7	0.16	0.08	5.3016	3.5728
64	15	6	0.57	0.46	5.2388	3.6504
65	18	7	0.1	0.14	5.548	3.6604
66	17	11	0.15	0.14	5.4288	3.8552
67	17	7	0.83	0.1	6.1068	4.1016
68	25	4	0.2	0.08	6.3864	3.7252
69	18	6	0.12	0.1	6.1912	4.0796
70	33	4	0.23	0.05	6.974	3.8672
100	36	0	0.31	0	27.35	4.66

Таблица 2.
Результаты экспериментов на примерах B-1.

n	Не опт. ACO	Не опт. Гиб.	Отн.п. ACO	Отн.п. Гиб.	Мур. ACO	Мур. Гиб.
7	1	0	0.67	0	1.38	1.044
26	0	3	0	0.01	1.381	1.871
27	0	1	0	0.01	1.429	1.707
31	0	4	0	0.01	1.354	1.929

Таблица 3.
Результаты экспериментов на примерах DL.

$3n+1$	Не опт. ACO	Не опт. Гиб.	Отн.п. ACO	Отн.п. Гиб.	Мур. ACO	Мур. Гиб.
16	1	0	0	0	4.92	3.4
19	2	3	0	0	10.3	11.64
22	2	2	0	0	7.66	8.22
25	4	4	0	0	16.28	16.44
28	6	4	0	0	19.2	15.24
31	0	3	0	0	8.16	15.66
34	1	1	0	0	8.8	9.44
37	1	0	0	0	7.12	7.58

Таблица 4.
Результаты экспериментов на примерах DL без локального поиска.

$3n+1$	Не опт. ACO	Не опт. Гиб.	Отн.п. ACO	Отн.п. Гиб.	Мур. ACO	Мур. Гиб.
13	50	26	13.46	0.01	100	58.2
16	50	35	0.77	0.03	100	73.82
19	50	43	3.29	2.78	100	88.06
22	50	46	2.86	2.05	100	93.52
25	50	49	2.03	1.58	100	98.02
28	50	50	2.97	2.49	100	100
31	50	49	3.48	2.02	100	98.48
34	50	49	2.85	1.67	100	98.4
37	50	49	1.71	1.41	100	98.02