УДК 519.854.2

Гибридный алгоритм решения задачи минимизации суммарного запаздывания для одного прибора

©2006 г. Е.Р. Гафаров

Москва, ВЦ РАН

Поступила в редакцию . .

Аннотация

Для NP-трудной в обычном смысле задачи теории расписаний минимизация суммарного запаздывания для одного прибора $1 \mid\mid \sum T_j$ построен *Гибридный алгоритм*, использующий идею известного метаэвристического алгоритма "Муравьиные колонии"[9] и комбинаторные свойства Правил исключения 1-4 [3, 7, 8]. Приводится сравнительный анализ эффективности Гибридного алгоритма и алгоритма "Муравьиные колонии".

Введение

В работе рассматривается классическая NP-трудная в обычном смысле задача теории расписаний минимизация суммарного запаздывания для одного прибора $1 \mid\mid \sum T_j$. 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.

Исследуемая задача является NP-трудной в обычном смысле [1]. Лаулер [2] предложил псевдополиномиальный алгоритм решения общего случая проблемы трудоемкости $O(n^4 \sum p_j)$. Шварц и др. построили [3, 4] алгоритмы решения проблемы, которые были протестированы для примеров $n < 600$ (тестовые примеры Поттса и Ван Вассенхова [5]). Исследование приближенных алгоритмов решения проблемы было проведено в работе [6], где построены примеры, на которых известные приближенные алгоритмы находят решение с относительной погрешностью порядка размерности примера n.

На основе известного алгоритма "Муравьиные колонии" – *Ant Colony Optimization*(ACO)[9] и Правил Исключения 1-4[3, 7, 8] 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[5], примеры случая **B-1**[7], канонические DL-примеры[1]).

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**[4, 7]) 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**[4, 7]),

где $d_{n'+1} := +\infty$.

Теорема 1. [7]Для любого примера (подпримера) $\langle \{p_j, d_j\}_{j \in N}, t_0 \rangle$ множество $L(N, t_0)$ не пусто.□

Лемма 1. [2, 5, 7] 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 [3, 8]. 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 $|L(N', t')| > 1$.□

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 А

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) [9]. 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 подстановки 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: $\eta_{ij} = \frac{1}{d_j}$;
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: $\eta_{ij} = \frac{1}{p_j}$,

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 подстановки при недостатке накопленной информации.

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 [9] приводиться экспериментальная оценка эффективности этого алгоритма. Эксперименты проводились для тестовых примеров [5] 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 [9].

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 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.*

На каждой итерации запускается модифицированный алгоритм A, в котором очередное требование j^* "случайным" образом ставиться на некоторую позицию $k \in L(N, t)$.

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 $i \in L(N, t)$.

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

где $\pi^i = (j_1, \ldots, j_{m-1}, j_{m+1}, \ldots, j_i, j^*, j_{i+1}, \ldots, j_n),\ j^* = j_m, m < i.$

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.

Не трудно убедиться, что трудоемкость 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 трудоемкость Гибридного алгоритма с локальным поиском составляет не меньше $O(mn^3)$ операций. То есть теоретическая трудоемкость исследуемых алгоритмов идентична.

4. Эффективность алгоритмов для тестовых примеров [5]

Вычислительные эксперименты, представленные в этом параграфе, проводились для примеров размерности $n = 4, \ldots, 70, 100$, которые генерировались с помощью схемы Поттса и Ван Васенхова [5].

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 $\{0.2, 0.4, 0.6, 0.8, 1\}$ [3, 4, 9]. 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 [9].

Для каждого примера запускался точный алгоритм A. Мы получали точное решение 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. Количество муравьев нами было ограничено $m \leq 100$.

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

Лучше найденное решение 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 следующей таблице.

n	Кол. прим.	не опт. АСО	не опт. Гиб.	rel. АСО	rel. Гиб.	Мур. АСО	Мур. Гиб.
4	2500	0	0	0	0	1.0404	1.004
5	2500	0	0	0	0	1.0616	1.0216
6	2500	0	0	0	0	1.0908	1.036
7	2500	0	0	0	0	1.1384	1.0612
8	2500	0	0	0	0	1.1968	1.0588
9	2500	0	0	0	0	1.1456	1.1044
10	2500	0	0	0	0	1.2576	1.1228
11	2500	0	0	0	0	1.2364	1.126
12	2500	0	0	0	0	1.2672	1.1484
13	2500	0	0	0	0	1.2968	1.2296
14	2500	0	0	0	0	1.3704	1.2464
15	2500	0	0	0	0	1.3576	1.2744
16	2500	0	0	0	0	1.4324	1.3928
17	2500	0	0	0	0	1.4376	1.3196
18	2500	0	0	0	0	1.4656	1.3216
19	2500	2	0	0.22	0	1.6164	1.4004
20	2500	1	0	0.58	0	1.6064	1.4204
21	2500	0	0	0	0	1.5892	1.4232
22	2500	1	0	0.16	0	1.626	1.4844
23	2500	0	0	0	0	1.6572	1.5184
24	2500	0	0	0	0	1.6992	1.5568
25	2500	0	0	0	0	1.8128	1.5504
26	2500	0	0	0	0	1.7736	1.584
27	2500	0	0	0	0	1.88	1.6828
28	2500	2	0	0.09	0	1.9704	1.6688
29	2500	0	0	0	0	1.9172	1.7872
30	2500	0	0	0	0	1.9744	1.7268
31	2500	0	0	0	0	1.9656	1.8656
32	2500	0	0	0	0	2.1688	1.8788
33	2500	0	0	0	0	2.214	1.844
34	2500	1	0	0.15	0	2.2212	1.9568
35	2500	0	0	0	0	2.3272	2.1152
36	2500	0	1	0	0.04	2.2332	2.154
37	2500	1	1	0.38	0.01	2.4796	2.102
38	2500	0	0	0	0	2.2696	2.1172
39	2500	0	0	0	0	2.576	2.1044

n	Кол. прим.	не опт. АСО	не опт. Гиб.	rel. АСО	rel. Гиб.	Мур. АСО	Мур. Гиб.
40	2500	1	0	0.04	0	2.6036	2.2424
41	2500	0	0	0	0	2.552	2.2704
42	2500	1	1	0.05	0.01	2.7888	2.4092
43	2500	1	1	0.07	0.06	2.7316	2.3656
44	2500	3	0	0.04	0	2.8464	2.3784
45	2500	2	0	0.68	0	2.9736	2.4728
46	2500	1	0	0.03	0	3.1624	2.4088
47	2500	2	0	0.01	0	3.248	2.5152
48	2500	9	0	0.56	0	3.4516	2.5196
49	2500	3	1	0.15	0.08	3.4252	2.7
50	2500	9	1	0.35	0.29	3.716	2.6336
51	2500	8	0	0.22	0	3.8412	2.7768
52	2500	4	1	0.04	0.07	3.5816	2.86
53	2500	4	2	0.03	0.42	3.8948	2.9668
54	2500	9	3	0.1	0.29	4.0324	2.9924
55	2500	8	2	0.11	0.06	4.1048	3.0496
56	2500	9	1	0.83	0.01	4.2916	3.0064
57	2500	7	0	0.23	0	4.1568	3.158
58	2500	14	0	0.17	0	4.71	3.3724
59	2500	14	4	0.24	0.1	4.81	3.3372
60	2500	11	1	0.22	0.01	4.7268	3.4224
61	2500	18	2	1.26	0.02	5.3032	3.5216
62	2500	10	2	0.26	0.01	5.0964	3.5032
63	2500	17	7	0.16	0.08	5.3016	3.5728
64	2500	15	6	0.57	0.46	5.2388	3.6504
65	2500	18	7	0.1	0.14	5.548	3.6604
66	2500	17	11	0.15	0.14	5.4288	3.8552
67	2500	17	7	0.83	0.1	6.1068	4.1016
68	2500	25	4	0.2	0.08	6.3864	3.7252
69	2500	18	6	0.12	0.1	6.1912	4.0796
70	2500	33	4	0.23	0.05	6.974	3.8672
100	617	36	0	0.31	0	27.35	4.66

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

В первой колонке представлено значение n – размерность задачи. Во второй колонке – количество рассмотренных примеров. В третьей и четвертой колонках, соответственно, количество примеров, для которых алгоритм ACO или Гибридный не нашли точные решения. В пятой и шестых 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.

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 [7]

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 [12] 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 следующие результаты:

n	Кол. прим.	не опт. АСО	не опт. Гиб.	rel. АСО	rel. Гиб.	Мур. АСО	Мур. Гиб.
4	1000	0	0	0	0	1.038	1.005
5	1000	0	0	0	0	1.071	1.034
6	1000	0	0	0	0	1.159	1.073
7	1000	1	0	0.67	0	1.38	1.044
8	1000	0	0	0	0	1.311	1.058
9	1000	0	0	0	0	1.401	1.137
10	1000	0	0	0	0	1.418	1.088
11	1000	0	0	0	0	1.584	1.091
12	1000	0	0	0	0	1.478	1.247
13	1000	0	0	0	0	1.495	1.201
14	1000	0	0	0	0	1.441	1.207
15	1000	0	0	0	0	1.621	1.241
16	1000	0	0	0	0	1.501	1.279
17	1000	0	0	0	0	1.45	1.271
18	1000	0	0	0	0	1.526	1.805
19	1000	0	0	0	0	1.511	1.36
20	1000	0	0	0	0	1.448	1.252
21	1000	0	0	0	0	1.457	1.457
22	1000	0	0	0	0	1.448	1.373
23	1000	0	1	0	0	1.481	1.761
24	1000	0	0	0	0	1.446	1.644
25	1000	0	2	0	0	1.337	1.696
26	1000	0	3	0	0.01	1.381	1.871
27	1000	0	1	0	0.01	1.429	1.707
28	1000	0	1	0	0	1.532	1.8
29	1000	0	3	0	0	1.423	1.815
30	1000	0	2	0	0	1.311	2.027
31	1000	0	4	0	0.01	1.354	1.929
32	1000	0	2	0	0	1.343	1.98
33	1000	0	3	0	0	1.379	2.005
34	1000	0	2	0	0	1.166	1.764
35	1000	0	7	0	0	1.287	2.435
36	1000	0	4	0	0	1.288	1.894
37	1000	0	5	0	0	1.237	2.102
38	1000	0	5	0	0	1.266	2.027
39	1000	0	5	0	0	1.216	2.115

n	Кол. прим.	не опт. АСО	не опт. Гиб.	rel. АСО	rel. Гиб.	Мур. АСО	Мур. Гиб.
40	1000	0	4	0	0	1.187	2.043
41	1000	0	3	0	0	1.241	1.868
42	1000	0	6	0	0	1.212	2.162
43	1000	0	4	0	0	1.287	2.042
44	1000	0	4	0	0	1.335	1.861
45	1000	0	2	0	0	1.304	2.149
46	1000	0	3	0	0	1.239	1.895
47	1000	0	4	0	0	1.224	1.847
48	1000	0	5	0	0	1.251	2.298
49	1000	0	3	0	0	1.264	2.179
50	1000	0	1	0	0	1.168	1.712
51	1000	0	0	0	0	1.251	1.332
52	1000	0	5	0	0	1.22	1.82
53	1000	0	6	0	0	1.213	1.995
54	1000	0	2	0	0	1.189	1.59
55	1000	0	1	0	0	1.139	1.639
56	1000	0	5	0	0	1.107	2.075
57	1000	0	5	0	0	1.18	2.049
58	1000	0	4	0	0	1.208	2.175
59	1000	0	0	0	0	1.218	1.424
60	1000	0	5	0	0	1.114	2.076
61	1000	0	4	0	0	1.15	1.773
62	1000	0	6	0	0	1.123	2.154
63	1000	0	4	0	0	1.114	1.909
64	1000	0	0	0	0	1.137	1.207
65	1000	0	4	0	0	1.112	1.854
66	1000	0	4	0	0	1.237	1.798
67	1000	0	1	0	0	1.132	1.57
68	1000	0	1	0	0	1.098	1.412
69	1000	0	4	0	0	1.12	1.912
70	1000	0	6	0	0	1.076	1.904
71	1000	0	5	0	0	1.105	1.907
72	1000	0	4	0	0	1.123	1.765
73	1000	0	3	0	0	1.084	1.589
74	1000	0	2	0	0	1.11	1.527
75	1000	0	4	0	0	1.122	1.656

n	Кол. прим.	не опт. ACO	не опт. Гиб.	rel. ACO	rel. Гиб.	Мур. ACO	Мур. Гиб.
76	1000	0	4	0	0	1.122	1.688
77	1000	0	0	0	0	1.177	1.382
78	1000	0	2	0	0	1.088	1.532
79	1000	0	6	0	0	1.122	2.114
80	1000	0	6	0	0	1.104	1.97
81	1000	0	3	0	0	1.103	1.553
82	1000	0	2	0	0	1.103	1.602
83	1000	0	2	0	0	1.18	1.653
84	1000	0	2	0	0	1.08	1.603
85	1000	0	3	0	0	1.111	1.555
86	1000	0	1	0	0	1.149	1.534
87	1000	0	0	0	0	1.11	1.415
88	1000	0	2	0	0	1.123	1.401
89	1000	0	1	0	0	1.087	1.484
90	1000	0	3	0	0	1.086	1.596
91	1000	0	4	0	0	1.083	1.76
92	1000	0	4	0	0	1.094	1.936
93	1000	0	2	0	0	1.097	1.519
94	1000	0	2	0	0	1.096	1.463
95	1000	0	2	0	0	1.093	1.459
96	1000	0	5	0	0	1.095	1.963
97	1000	0	0	0	0	1.079	1.134
98	1000	0	0	0	0	1.125	1.238
99	1000	0	1	0	0	1.073	1.275
100	1000	0	1	0	0	1.068	1.525

Таб.2. Результаты экспериментов на примерах B-1.

Необходимо заметить, что мы вычисляли относительную погрешность с точностью до сотых процента. Поэтому в таблице результатов можно наблюдать ситуацию, когда количество неточно решенных примеров больше 0, а относительная погрешность равна 0 (например для $n = 23$).

Алгоритм ACO 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 2-х муравьев чтобы найти точное решение.

Как видно из таб. 2 эффективность Гибридного алгоритма, использующего идеи алгоритма А, ниже эффективности алгоритма ACO на примерах B-1. Можно объяснить это тем, что примеры B-1 "наиболее сложные" для алгоритма А.

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 [1]

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

Постановка задачи ЧНР.

Задано упорядоченное множество из $2n$ положительных целых чисел $B = \{b_1, b_2, \ldots, b_{2n}\}$, $b_i > b_{i+1}$, $1 \leq i \leq 2n-1$. 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 $\mathbf{1} \;||\; \sum \boldsymbol{T_j}$ [1]. 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* [1] 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:

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

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

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

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

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 следующие результаты:

$3n+1$	Кол. прим.	не опт. АСО	не опт. Гиб.	rel. АСО	rel. Гиб.	Мур. АСО	Мур. Гиб.
13	50	0	0	0	0	1.96	1.94
16	50	1	0	0	0	4.92	3.4
19	50	2	3	0	0	10.3	11.64
22	50	2	2	0	0	7.66	8.22
25	50	4	4	0	0	16.28	16.44
28	50	6	4	0	0	19.2	15.24
31	50	0	3	0	0	8.16	15.66
34	50	1	1	0	0	8.8	9.44
37	50	1	0	0	0	7.12	7.58
40	50	0	0	0	0	5.88	4.74
43	50	0	0	0	0	4.14	5.76
46	50	0	0	0	0	3.32	4.48
49	50	0	0	0	0	4.52	4.76
52	50	0	0	0	0	3.28	4.48
55	50	0	0	0	0	3.36	4.26
58	50	0	0	0	0	3.82	4.58
61	50	0	0	0	0	3.04	4.36
64	50	0	0	0	0	3.48	3.46
67	50	0	0	0	0	3.22	3.48
70	50	0	0	0	0	2.26	3.1
73	50	0	0	0	0	2.46	3.36
76	50	0	0	0	0	2.96	3.22
79	50	0	0	0	0	2.22	2.52
82	50	0	0	0	0	2.94	3.24
85	50	0	0	0	0	3.34	3.72
88	50	0	0	0	0	2.8	3.56
91	50	0	0	0	0	2.64	2.6
94	50	0	0	0	0	2.7	2.8
97	50	0	0	0	0	2.7	2.9
100	50	0	0	0	0	2.46	2.68
103	50	0	0	0	0	2.48	2.52
106	50	0	0	0	0	3.08	2.46
109	50	0	0	0	0	2.44	2.2
112	50	0	0	0	0	2.18	2.22
115	50	0	0	0	0	2.08	2.12
118	50	0	0	0	0	2.02	1.96
121	50	0	0	0	0	2.18	2.56

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[1]. 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 $(1/2)^n$.

Мы повторили эксперимент на тех же примерах, при тех же условиях, отключив в алгоритмах "локальный поиск". Результаты приводятся в следующей таблице:

$3n+1$	Кол. прим.	не опт. АСО	не опт. Гиб.	rel. АСО	rel. Гиб.	Мур. АСО	Мур. Гиб.
13	50	50	26	13.46	0.01	100	58.2
16	50	50	35	0.77	0.03	100	73.82
19	50	50	43	3.29	2.78	100	88.06
22	50	50	46	2.86	2.05	100	93.52
25	50	50	49	2.03	1.58	100	98.02
28	50	50	50	2.97	2.49	100	100
31	50	50	49	3.48	2.02	100	98.48
34	50	50	49	2.85	1.67	100	98.4
37	50	50	49	1.71	1.41	100	98.02
40	50	50	50	0.96	1.79	100	100
43	50	50	50	2.3	2.06	100	100
46	50	50	50	1.16	2.24	100	100
49	50	50	50	2.18	2.36	100	100
52	50	50	50	1.61	1.75	100	100
55	50	50	50	1.42	1.87	100	100
58	50	50	50	1.08	1.4	100	100
61	50	50	50	1.22	1.51	100	100
64	50	50	50	1.37	1.6	100	100
67	50	50	50	2.41	1.66	100	100
70	50	50	50	1.82	1.71	100	100
73	50	50	50	1.52	1.57	100	100
76	50	50	50	1.71	1.61	100	100
79	50	50	50	2.47	1.49	100	100
82	50	50	50	2.44	1.65	100	100
85	50	50	50	2.02	2.69	100	100
88	50	50	50	1.93	2.03	100	100
91	50	50	50	2.79	1.79	100	100
94	50	50	50	2.28	1.46	100	100
97	50	50	50	1.95	1.37	100	100
100	50	50	50	2.21	1.75	100	100
103	50	50	50	1.48	1.74	100	100
106	50	50	50	2.05	1.56	100	100
109	50	50	50	1.78	1.4	100	100
112	50	50	50	1.97	1.76	100	100
115	50	50	50	1.76	1.95	100	100
118	50	50	50	1.79	1.98	100	100
121	50	50	50	2.27	1.38	100	100

Анализируя обе таблицы можно сделать вывод, что эффективность алгоритмов на данном классе примеров достигается в основном за счет локального поиска. Причем количество запусков локального поиска на каждой итерации может быть экспоненциально.

При $3n + 1 = 40$ оба алгоритма неточно решают все примеры.

Заключение

По результатам экспериментов, Гибридный алгоритм работает существенно эффективнее алгоритма ACO для тестовых примеров 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. J. Du and J. Y.-T. Leung, *Minimizing total tardiness on one processor is NP-hard* // Math. Oper. Res.. 1990. № 15. 483–495.

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

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

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

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

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

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

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

9. A. Bauer, B. Bullnheimer, R.F.Hartl, C. Strauss. *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. 1445–1450.

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

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

12. 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).