ИЗВЕСТИЯ РАН. ТЕОРИЯ И СИСТЕМЫ УПРАВЛЕНИЯ, 2008, № 5, с. 69–74

СИСТЕМНЫЙ АНАЛИЗ И ИССЛЕДОВАНИЕ ОПЕРАЦИЙ

УДК 519.86

АЛГОРИТМЫ РЕШЕНИЯ МИНИМАКСНОЙ ЗАДАЧИ СОСТАВЛЕНИЯ РАСПИСАНИЯ

© 2008 г. Д. В. Красовский, М. Г. Фуругян

Москва, ВЦ РАН

Поступила в редакцию 01.02.08 г., после доработки 22.04.08 г.

Для задачи составления оптимального по быстродействию расписания без прерываний и переключений в многопроцессорной системе разработан ряд точных и приближенных алгоритмов. Определяется эффективность алгоритмов и проводится их сравнительный анализ.

Введение. Одна из основных задач, возникающих при разработке программного обеспечения многопроцессорных вычислительных систем, в частности систем реального времени, заключается, во-первых, в создании математической модели, адекватной реальной вычислительной системе, и, во-вторых, в расчете с помощью этой модели режима функционирования системы, в результате которого получается расписание, показывающее, когда и какой программе должны быть выделены те или иные ресурсы ЭВМ.

Большинство задач планирования вычислений в многопроцессорных системах являются NP-трудными, и все известные точные алгоритмы их решения имеют переборный характер, а точных полиномиальных (эффективных) алгоритмов в настоящее время не известно. Число шагов переборного метода, как правило, растет экспоненциально в зависимости от размеров задачи. Поэтому при решении переборных задач больших размеров возникает необходимость в разработке приближенных методов. Для некоторых задач планирования вычислений в многопроцессорных системах удается построить эффективные (существенно менее трудоемкие, чем полный перебор вариантов) методы решения. Например, это удается сделать для задачи составления расписания с прерываниями без ограничений на частичный порядок выполнения заданий. Однако число таких задач невелико. Настоящая работа посвящена исследованию точных и приближенных алгоритмов составления оптимального по быстродействию расписания без прерываний в многопроцессорной вычислительной системе. Для решения этой задачи разработаны новые точные псевдополиномиальные алгоритмы (с поиском в ширину и в глубину) при фиксированном числе процессоров и приближенные алгоритмы (эвристический и вероятностный). Для случая, когда процессоры идентичны, предложен алгоритм, основанный на методе агрегирования. Данные алгоритмы позволяют также находить решения с гарантированной точностью. В статье приводятся результаты вычислительных экспериментов, на основании которых проводится сравнительный анализ этих алгоритмов и одного из известных.

Подобные задачи широко освещены в литературе. Отметим, например, такие методы, применяемые при их решении, как случайный и исчерпывающий поиск [1, 2], математическое программирование [3], метод ветвей и границ [4], муравьиные алгоритмы [5], поиск с запретами [6], вероятностные алгоритмы [7], генетические алгоритмы [8], имитация отжига [9, 10], различные эвристические алгоритмы [11, 12] и др.

1. Постановка задачи. Рассматривается вычислительная система $P = \{P_1, \ldots, P_m\}$, состоящая из m процессоров P_j, $j = 1, 2, \ldots, m$. Имеется множество заданий или работ, $T = \{T_1, \ldots, T_n\}$. Для каждого задания T_i известна длительность τ_{ij} его выполнения на процессоре P_j. Предполагается, что в каждый момент времени каждый процессор может выполнять не более одной работы, а каждая работа исполняется не более чем одним процессором. При этом не допускаются прерывания и переключения с одного процессора на другой. Требуется так распределить задания по процессорам, чтобы общее время выполнения всего множества работ было минимальным. Иными словами, необходимо построить оптимальное по быстродействию расписание, т.е. имеющее минимальную длину

$$\omega(S) = \max_{j = 1, \ldots, m} f_j(S),$$

$$f_j(S) = \sum_{i \in T(j)} \tau_{ij},$$

где $T(j)$– множество работ, назначенных на j-й процессор.

2. Приближенные алгоритмы. 2.1. Эвристический алгоритм "Процессор с ранним окончанием первым" (ПРОП).

* Работа выполнена при частичной финансовой поддержке гранта Президента РФ по поддержке ведущих научных школ НШ – 693.2008.1.

Работа этого алгоритма состоит в следующем. На k-м шаге задание T_k назначается на тот процессор, суммарное время выполнения работ на котором с учетом данной работы минимальное. Иными словами, минимизируется по j выражение $(B^j_{k-1} + \tau_{kj})$, где B^j_{k-1} – суммарное время выполнения заданий, назначенных на j-й процессор на первых $k-1$ шагах. После чего T_k назначается на процессор j. Вычислительная сложность алгоритма составляет $O(nm)$.

2.2. Вероятностный алгоритм (ВА). Предлагаемый алгоритм основан на методе решения задачи упаковки, изложенном в [7]. Запишем поставленную задачу в виде задачи булевого линейного программирования

$$\sum_{j=1}^{m} x_{ji} = 1, \quad i = 1, \ldots, n, \tag{2.1}$$

$$\sum_{j=1}^{n} x_{ji}\tau_{ij} \le B, \quad j = 1, \ldots, m, \tag{2.2}$$

$$x_{ji} \in \{0, 1\}, \tag{2.3}$$

$$Z = B \longrightarrow \min. \tag{2.4}$$

При этом работа T_i выполняется на процессоре P_j тогда и только тогда, когда $x_{ji} = 1$. В силу условий (2.1) и (2.3) каждая работа будет исполнена ровно на одном процессоре (расписание без прерываний). Условие (2.2) – ограничение на длину расписания, а в силу условия (2.4) будет найдено расписание с минимальной длиной.

Сначала решим релаксационную задачу линейного программирования, в которой $x_{ji} \in [0, 1]$. Это может быть сделано, например, симплексным методом или одним из полиномиальных алгоритмов. Пусть в результате получена матрица с элементами x^*_{ji}. Затем найдем такую целочисленную матрицу $\|\bar{x}_{ji}\|$, аппроксимирующую x^*_{ji}, что $P(\bar{x}_{ji} = 1) = x^*_{ji}$ и выполняется условие (2.1). Это можно сделать следующим образом. Заполним элементы матрицы $\|\bar{x}_{ji}\|$ по строкам: для каждого j будем брать последовательно элементы $\bar{x}_{ji}$ с i от 1 до n, полагая $\bar{x}_{ji}$ равным 1 с вероятностью x^*_{ji}. Если некоторый элемент $\bar{x}_{ji}$, полученный таким образом, окажется равным 1, то остальные элементы строки j полагаем равными 0. Матрица $\|\bar{x}_{ji}\|$ определяет искомое расписание.

3. Псевдополиномиальные алгоритмы. Дадим следующую интерпретацию рассматриваемой задачи. Необходимо решить оптимизационную задачу вида $\left\|\sum_{i=1}^{n} \vec{p}_i\right\| \longrightarrow \min$, где $\vec{p}_i = (0, \ldots, \tau_{ij}, \ldots, 0)$ – m-мерный вектор, j-я компонента которого равна τ_{ij}, $\|\vec{a}\| = \max\limits_k a_k$ для вектора $\vec{a} = (a_1, a_2, \ldots, a_m)$.

3.1. Псевдополиномиальный алгоритм с поиском в ширину (ПАПШ). Работу алгоритма, решающего поставленную задачу, опишем следующим образом. Вычислим верхнюю оценку длины расписания, величину B, которую будем называть директивным сроком. Значение B можно получить, например, с помощью алгоритма ПРОП. Затем будем рассматривать m-мерный куб Ω со стороной B. Необходимо выбрать векторы $\vec{p}_i$ так, чтобы каждая компонента вектора $\sum_{i=1}^{n} \vec{p}_i$ не превосходила B. Под выбором вектора $\vec{p}_i$ будем понимать выбор процессора, на который назначена работа T_i, что однозначно задает вектор. Выбор осуществляем путем построения множества точек m-мерного пространства (дерева решений), для каждой из которых на l-м уровне проверяем, принадлежит ли точка кубу Ω. Если это условие не выполнено, то точка исключается из дальнейшего рассмотрения. Формализуем алгоритм по шагам.

Шаг 1. Строим первые m точек m-мерной решетки

$$(\tau_{11}, 0, \ldots, 0), (0, \tau_{12}, \ldots, 0), \ldots, (0, 0, \ldots, \tau_{1m}).$$

Полагаем $l = 1$, а все указанные точки активными.

Шаг 2. Исключаем из списка активных точек те, которые не принадлежат Ω. Если $l = n$, то переходим на шаг 4. В противном случае переходим на шаг 3.

Шаг 3. Полагаем $l = l + 1$. Каждую активную точку $(a_1, a_2, \ldots, a_m)$, полученную на шаге $l-1$, исключаем из списка активных точек и строим новые активные точки $(a_1 + \tau_{l1}, a_2, \ldots, a_m)$, $(a_1, a_2 + \tau_{l2}, \ldots, a_m)$, $\ldots$, $(a_1, a_2, \ldots, a_m + \tau_{lm})$. Переходим на шаг 2.

Шаг 4. Множество решений найдено. Из числа активных точек находим точку, соответствующую оптимальному решению. Завершаем работу алгоритма.

Таким образом, первый этап работы алгоритма (шаги 1–3) позволяет получить все расписания с длиной, не превосходящей B. Отметим, что на шаге 2 множество активных точек не может быть пусто, так как известно, что существует хотя бы одно решение, найденное с помощью алгоритма ПРОП. На шаге 4 алгоритма проводится поиск оптимального расписания.

Из описания алгоритма видно, что на первом шаге выполняется $O(m)$ действий с точками m-мерного пространства (всего $O(m^2)$ операций), затем на этапах 2 и 3 алгоритма для каждой из активных точек m-мерного пространства выполняется по $O(m)$ операций. Поскольку в худшем случае число вершин ветвления совпадает с числом узлов целочисленной решетки в Ω (т.е. с числом $(B+1)^m$ в силу це-

лочисленности задачи), то вычислительная сложность шагов 1–3 алгоритма составляет $O(m^2(B+1)^m)$. На шаге 4 необходимо из множества оконечных точек допустимых решений найти такую точку $(a_1, a_2, \ldots, a_m)$, для которой $\max\limits_k a_k$ минимально. Так как для каждой оконечной точки необходимо выполнить $(m-1)$ сравнений, а число таких точек не может превышать числа всех точек рассматриваемой целочисленной решетки, то окончательная вычислительная сложность алгоритма составляет $O(m^2B^m)$. Таким образом, при фиксированном числе процессоров m описанный алгоритм является псевдополиномиальным.

3.2. Псевдополиномиальный алгоритм с поиском в глубину (ПАПГ). Этот алгоритм отличается от рассмотренного в предыдущем разделе тем, что вместо построения полного дерева решений, укладывающегося в m-мерный куб со стороной B, находится одно решение, удовлетворяющее директивному сроку B (или определяется, что решения не существует). Эта процедура выполняется для значений B, полученных с помощью метода деления отрезка пополам. Начальный отрезок возможных значений для B – это отрезок $[\underline{B}, \overline{B}]$. Здесь

$$\underline{B} = \max\left\{\max_i(\min_j \tau_{ij});\ \frac{\sum\limits_i \min\limits_j \tau_{ij}}{m}\right\} \qquad (3.1)$$

– нижняя оценка для B (величина $\max\limits_i(\min\limits_j \tau_{ij})$ – длительность наиболее ресурсоемкой работы, считая, что она распределена на наиболее быстрый для ее выполнения процессор, а величина $\dfrac{\sum\limits_i \min\limits_j \tau_{ij}}{m}$ равна времени завершения исполнения всех работ в случае, когда они равномерно распределены на наиболее быстрые для них процессоры). Величина $\overline{B}$ – верхняя оценка для B, полученная с помощью алгоритма ПРОП. В худшем случае каждая итерация алгоритма может иметь такую же сложность, как и алгоритм с поиском в ширину, однако в среднем каждая итерация этого алгоритма работает существенно быстрее (это подтверждается экспериментальными данными). Представленный алгоритм обладает также тем преимуществом, что с его помощью можно находить как точные, так и приближенные решения с заданной точностью. Для этого следует остановить работу алгоритма при разности между верхней и нижней оценками, не превосходящей требуемой точности.

3.3. Комбинированный псевдополиномиальный алгоритм (КПА), основанный на декомпозиции. Для решения задачи составления расписаний большой размерности предлагается следующий алгоритм. С помощью алгоритма ПРОП строится расписание. На следующем шаге проводится декомпозиция задачи на подзадачи в соответствии с полученным расписанием. Для этого проводится разбиение множества процессоров на M групп (M – задаваемый параметр) таким образом, что в первую группу попадает $\left[\dfrac{m}{2M}\right]$ наиболее загруженных процессоров (суммарное время выполнения работ на которых наибольшее) и столько же наименее загруженных (здесь $[a]$ – целая часть числа a). Аналогично строятся остальные группы из оставшихся процессоров. Затем производится декомпозиция множества работ. Оно также разбивается на M групп, и в каждую группу попадают те работы, которые были назначены алгоритмом ПРОП на процессоры из соответствующей группы. Каждая из полученных таким образом подзадач решается с помощью одного из описанных выше псевдополиномиальных алгоритмов, после чего решения подзадач агрегируются в решение исходной задачи.

Если в исходной задаче требуется получить расписание с точностью не хуже заданной, то на первом шаге для построения первоначального решения вместо быстрого эвристического алгоритма можно использовать псевдополиномиальный алгоритм с поиском в глубину. При этом следует находить не точное решение, а остановиться на шаге $(i+1)$, таком, что $|B_i - B_{i+1}| < \varepsilon$. Здесь ε – заданная точность, B_i и B_{i+1} – директивные сроки i-й и $(i+1)$-й итераций алгоритма, причем на одной из этих итераций решение было найдено, а на другой было определено, что решения с таким директивным сроком не существует.

3.4. Сравнительный анализ псевдополиномиальных алгоритмов. В задачах, требующих нахождения точного решения, наиболее эффективным представляется (и это подтверждается проведенными авторами экспериментами) использование ПАПГ. Практическое применение ПАПШ осложняется тем фактом, что при построении дерева решений необходимо хранить и обрабатывать в памяти большое количество данных, что существенно замедляет работу алгоритма. В случаях, когда допускается нахождение решений с некоторой погрешностью относительно оптимального, КПА позволяет найти допустимое решение наиболее быстро (погрешность составляла 6–12%).

4. Агрегирующий алгоритм. Метод агрегирования для поставленной задачи заключается в том, что исходное множество заданий разбивается на несколько подмножеств, для каждого из которых при-

меняется один из точных алгоритмов построения расписания. Из полученных таким образом расписаний строится расписание выполнения исходного множества заданий. Авторами были разработаны и исследованы несколько агрегирующих алгоритмов для случая, когда все процессоры идентичны. В этом случае для каждого задания T_i задается его длительность t_i, которая не зависит от процессора. Лучшие результаты были получены для многоуровневого агрегирующего алгоритма (МАА), который может быть описан следующим образом. Множество всех заданий, предварительно упорядоченных по не возрастанию длительностей, разбивается на s_1 подмножеств. Если n кратно s_1, то в каждое подмножество попадает $[n/s_1]$ соседних заданий. В противном случае в некоторые подмножества может попасть на одно задание больше, при этом разброс длительностей в пределах одного подмножества должен быть минимальным. Для каждого из полученных подмножеств с помощью ПАПГ строится оптимальное m-процессорное расписание. Задания из одного подмножества, назначенные на один и тот же процессор, объединяются в одно задание, длительность которого равна сумме длительностей соответствующих заданий. В результате получаем множество из ms_1 агрегированных заданий, которое свою очередь может быть аналогично разбито на s_2 подмножеств. После выполнения нескольких процедур агрегирования будет получено решение исходной задачи. При реализации данного метода полагалось $s_i = [s_{i-1}/2]$. Отметим, что подобный подход для решения задачи коммивояжера рассмотрен в [13].

5. Алгоритмы с гарантированной точностью. Описанные выше алгоритмы ПАПГ, ВА и ПРОП позволяют находить решения с гарантированной точностью. Это сформулировано в приведенных ниже утверждениях, доказательства которых содержатся в [14]. Используются следующие обозначения: B – длина расписания, найденного соответствующим алгоритмом; $\underline{B}$ – нижняя оценка длины оптимального расписания, вычисленная по формуле (3.1); $\overline{B}$ – верхняя оценка длины оптимального расписания (вычисленная, например, алгоритмом ПРОП); B^* – длина оптимального расписания.

Утверждение 1. С помощью ПАПГ можно найти решение с погрешностью $\varepsilon = \dfrac{B - B^*}{B^*}$ за время $O\left(\log\left(\left(\dfrac{\overline{B} - \underline{B}}{\varepsilon \underline{B}}\right) + 1\right) m^2 (\overline{B} - \varepsilon \underline{B})^m\right)$, если $\varepsilon \underline{B} > 1$, и за время $O(\log(\overline{B} - \underline{B}) m^2 \overline{B}^m)$, если $\varepsilon \underline{B} \le 1$.

Утверждение 2. За время $O\left(\log\log\left(\dfrac{\overline{B}}{\underline{B}}\right) m^2 \overline{B}^m\right)$ с помощью ПАПГ можно найти расписание длиной B, такое, что $\dfrac{1}{2} B \le B^* \le B$.

Утверждение 3. С вероятностью $P_n(k) \ge \left(1 - \left(1 - \dfrac{1}{m}\right)^{mk}\right)^n$ за время $O(nmk)$ (не считая времени решения релаксационной задачи) с помощью ВА можно с вероятностью, большей $1 - \dfrac{1}{n}$, найти решение с погрешностью $\varepsilon = \dfrac{B - B^*}{B^*} \le D\left(B^*, \dfrac{1}{n}\right)$, где

$$D\left(B^*, \frac{1}{n}\right) \le (e-1)\sqrt{\frac{\ln n}{B^*}} \text{ при } B^* > \ln n,$$

$$D\left(B^*, \frac{1}{n}\right) \le \frac{e \ln n}{B^* \ln\left(\dfrac{e \ln n}{B^*}\right)} \text{ при } B^* \le \ln n.$$

Утверждение 4. Погрешность ε расписания, получаемого алгоритмом ПРОП, составляет $\varepsilon = \dfrac{B - \underline{B}}{\underline{B}} \le m - 1$.

Утверждение 5. Для случая, когда процессоры идентичны, погрешность расписания, получаемого алгоритмом ПРОП, составляет $\varepsilon = \dfrac{B - B^*}{B^*} \le 1$.

6. Результаты экспериментов. Продемонстрируем результаты работы описанных выше алгоритмов. Расчеты проводились на персональном компьютере на базе процессора AMD Athlon XP 2800+. Для реализации алгоритмов было разработано программное обеспечение с использованием языков программирования Java и C++. Для нахождения решения релаксационной задачи линейного программирования применялась свободно распространяемая библиотека lp_solve 5.1 (http://geocities.com/lpsolve/), где B – длина полученного расписания; $\Delta = \dfrac{B - \underline{B}}{\underline{B}} \times 100\%$ – относительная погрешность (в процентах); t – время работы алгоритма (в секундах); m – количество процессоров; n – количество работ; s_1 – число подмножеств заданий на первом уровне агрегирования при работе МАА. Для каждого набора (m, n) проводилось 50 экспериментов со значениями длительностей работ, полученных с помощью программного генератора случайных чисел, позволяющего выдавать псевдослучайные числа с равномерным распределением на

Таблица 1. Сравнительные характеристики алгоритмов ПРОП, ВА, ПАПГ и МИО

Условия задачи	ПРОП	ВА	ПАПГ	МИО
$m = 2$, $n = 20$	$\Delta = 18.2$, $t = 0.001$	$\Delta = 6.1$, $t = 0.1$	$\Delta = 0$, $t = 0.02$	$\Delta = 12.6$, $t = 1.4$
$m = 2$, $n = 50$	$\Delta = 14.2$, $t = 0.002$	$\Delta = 3.1$, $t = 0.2$	$\Delta = 2.9$, $t = 320$	$\Delta = 16.4$, $t = 2.5$
$m = 8$, $n = 30$	$\Delta = 47.6$, $t = 0.001$	$\Delta = 46.2$, $t = 0.1$	$\Delta = 0$, $t = 120$	$\Delta = 46.3$, $t = 1.8$
$m = 2$, $n = 1000$	$\Delta = 13.6$, $t = 0.016$	$\Delta = 0.2$, $t = 1.0$	–	$\Delta = 12.3$, $t = 8.7$
$m = 4$, $n = 5000$	$\Delta = 25.1$, $t = 0.042$	$\Delta = 0.1$, $t = 56$	–	$\Delta = 22.9$, $t = 1.8$

Таблица 2. Сравнительные характеристики алгоритмов МАА, ПРОП и МИО

Условия задачи	МАА	ПРОП	МИО
$m = 2$, $n = 20$, $s_1 = 2$	$\Delta = 0.05$, $t < 1$	$\Delta = 0.5$, $t < 0.01$	$\Delta = 3.7$, $t = 1.5$
$m = 2$, $n = 100$, $s_1 = 8$	$\Delta = 0.0$, $t = 1$	$\Delta = 2$, $t < 0.01$	$\Delta = 0.1$, $t = 2.1$
$m = 4$, $n = 20$, $s_1 = 2$	$\Delta = 0.5$, $t = 2.1$	$\Delta = 15$, $t < 0.01$	$\Delta = 14.9$, $t = 1.6$
$m = 4$, $n = 100$, $s_1 1 = 16$	$\Delta = 0.5$, $t = 12$	$\Delta = 1$, $t = 0.01$	$\Delta = 1.4$, $t = 2.4$
$m = 2$, $n = 1000$, $s_1 = 16$	$\Delta = 0.5$, $t = 12$	$\Delta = 3.5$, $t = 0.02$	$\Delta = 0.8$, $t = 9.1$
$m = 4$, $n = 1000$, $s_1 = 100$	$\Delta = 2.4$, $t = 317$	$\Delta = 3.5$, $t = 0.02$	$\Delta = 3.2$, $t = 12.4$

отрезке [1, 1000]. Рассчитанные значения погрешности и времени работы алгоритма затем усреднялись путем отбрасывания пяти самых лучших и самых худших значений и вычисления среднего арифметического оставшихся значений.

Для случая различных процессоров в табл. 1 приведены результаты работы алгоритмов ПРОП, ВА И ПАПГ, а также для сравнения – одного из лучших ранее известных алгоритмов, основанного на методе имитации отжига (МИО), для пяти вариантов данных. Как видно из результатов расчета, из приближенных алгоритмов наиболее предпочтительным является ВА. ПАПГ позволяет находить точные решения, однако для задач больших размерностей время его работы существенно превосходит время работы остальных алгоритмов. Для случая идентичных процессоров в табл. 2 приведены результаты работы алгоритмов МАА, ПРОП и МИО для шести вариантов данных. Согласно результатам расчета, наименьшую погрешность имеет МАА, однако для задач больших размерностей время его работы существенно превосходит время работы остальных алгоритмов. Время работы алгоритма ПРОП существенно меньше времени работы остальных алгоритмов. Что касается погрешности, то ПРОП ненамного уступает МИО, а в ряде случаев превосходит его.

Заключение. Для решения минимаксной задачи составления расписания без прерываний разработаны новые точные и приближенные алгоритмы. Точные алгоритмы отличаются от известных ранее алгоритмов тем, что имеют псевдополиномиальную оценку сложности (при фиксированном числе процессоров). Предложены псевдополиномиальные алгоритмы с поиском в ширину и глубину, а также комбинированный приближенный псевдополиномиальный алгоритм, основанный на декомпозиции. Кроме того, предложены приближенные эвристический и вероятностный алгоритмы. Как показали численные эксперименты, среди точных алгоритмов наиболее эффективным оказался псевдополиномиальный алгоритм с поиском в глубину. Для решения указанной задачи впервые разработаны алгоритмы с помощью метода агрегирования, среди которых наиболее точным оказался многоуровневый агрегирующий алгоритм. Для случая идентичных процессоров этот алгоритм имеет также наименьшую погрешность по сравнению с другими предложенными в статье приближенными алгоритмами. Кроме того, он показал большую точность, чем один из лучших известных ранее алгоритмов, базирующийся на методе имитации отжига. Для случая различных процессоров наиболее точным из приближенных алгоритмов является вероятностный алгоритм, а наиболее быстрым – эвристический алгоритм. Получены оценки погрешностей приближенных алгоритмов.

СПИСОК ЛИТЕРАТУРЫ

1. *Гончаров Е.Н., Кочетов Ю.А.* Вероятностный поиск с запретами для дискретных задач безусловной оптимизации // Дискретный анализ и исследования операций. Сер. 2. 2002. Т. 9. № 2. С. 13–30.
2. *Кочетов Ю., Младенович Н., Хансен П.* Локальный поиск с чередующимися окрестностями // Дискретный анализ и исследования операций. Сер. 2. 2003. Т. 10. № 1. С. 11–44.
3. *Кочетов Ю.А., Столяр А.А.* Использование чередующихся окрестностей для приближенного решения задачи календарного планирования с ограни-

ченными ресурсами // Дискретный анализ и исследования операций. Сер. 2. 2003. Т. 10. № 2. С. 29–56.

4. *Алексеев О.Г.* Комплексное применение методов дискретной оптимизации. М.: Наука, 1986.
5. *Штовба С. Д.* Муравьиные алгоритмы // ExponentaPro. Математика в приложениях. 2003. № 4(4). С. 70–75.
6. *Glover F., Laguna M.* Chapter 3: Tabu search/Ed. R. Colin Reeves. Modern Heuristics Techniques for Combinatorial Problems. Oxford: Blackwell Scientific Publications, 1993. P. 70–150.
7. *Raghavan R.* Probabilistic Construction of Deterministic Algorithms: Approximating Packing Integer Programs // J. Computer and System Sciences. 1988. V. 37. P. 130–143.
8. *Костенко В.А., Смелянский Р.Л., Трекин А.Г.* Синтез структур вычислительных систем реального времени с использованием генетических алгоритмов // Программирование. 2000. № 5. С. 63–72.
9. *Laarhoven P., Aarts E., Lenstra J.* Job Shop Scheduling by Simulated Annealing // Operations Research. 1992. V. 40(1). P. 113–125.
10. *Shen C., Pao Y., Yip P.* Scheduling multiple job problems with guided evolutionary simulated annealing approach // Proc. First IEEE Conf. on Evolutionary Computations. Orlando, 1994. P. 702–706.
11. *Головкин Б.А.* Расчет характеристик и планирование параллельных вычислительных процессов. М.: Радио и связь, 1983.
12. *Brucker P.* Scheduling Algorithms. Heidelberg: Springer, 2001.
13. *Сигал И.Х., Иванова А.П.* Введение в прикладное дискретное программирование. М.: Физматлит, 2002.
14. *Красовский Д.В.* Алгоритмы решения задачи составления оптимального расписания без прерываний. Дис … канд. физ.-матем. наук. М.: МФТИ, 2007.