МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование кафедры)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Системное программирование и компьютерные технологии
(направленность (профиль)/специализация)

БАКАЛАВРСКАЯ РАБОТА

на тему Метод ветвей и границ для конвейерной задачи теории расписаний

Студент	Р.Р. Кашкин	
	(И.О. Фамилия)	(личная подпись)
Руководитель	Н.А. Сосина	
	(И.О. Фамилия)	(личная подпись)
Консультанты	Н.В. Андрюхина	
	(И.О. Фамилия)	(личная подпись)

Допустить к защите

Заведующий кафедрой к.т.н., доцент, А.В. Очеповский _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)

«_____»_____20_____г.

Тольятти 2019

АННОТАЦИЯ

Представленная бакалаврская работа посвящена актуальной проблеме применения метода ветвей и границ для решения конвейерной задачи теории расписаний.

Целью бакалаврской работы является анализ и применение метода ветвей и границ для решения конвейерной задачи теории расписаний.

Объектом исследования является конвейерной задачи теории расписаний.

Предмет исследования - метод ветвей и границ для конвейерной задачи теории расписаний.

Первая глава посвящена проблеме применения метода ветвей и границ для решения задач дискретной оптимизации. Описаны принципы метода ветвей и границ.

Во второй главе рассматривается формализация конвейерной задачи теории расписаний. Выполнена постановка задачи и описана математическая модель конвейерной задачи теории расписаний.

В третьей главе исследован алгоритм ветвей и границ для решения конвейерной задачи теории расписаний. Выполнены реализация и проверка данного алгоритма для решения конвейерной задачи Джонсона.

Структура бакалаврской работы: страниц 46, рисунков 6, таблиц 16, источников 24.

ABSTRACT

This bachelor’s thesis is devoted to the actual problem of utilizing the branch and bound method for solving the flow shop scheduling problem.

The goal of this bachelor’s thesis was to analyze and utilize applying the branch and bound method for solving the flow shop scheduling problem.

The object of the study was flow shop scheduling problem.

The first part of the bachelor’s thesis is devoted to the problem of utilizing the branch and bound method for solving discrete optimization problems. The principles of the branch and bound method are described.

The second part of the bachelor’s thesis considers the formalization of the flow shop scheduling problem. The statement of the problem was performed and a mathematical model of a flow shop scheduling problem was described.

In the third part of the bachelor’s thesis branch and bound algorithms for solving the flow shop scheduling problem were implemented.

The bachelor's work consists of an explanatory note on 46 pages including 6 figures, 16 tables, the list of 24 references.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время для управления социально-экономическими системами и производственными объектами применяются прикладные задачи, относящиеся к категории задач теории расписаний.

Как правило, на основе методов теории расписаний решаются задачи производственного календарного и сетевого планирования, а также управления проектами.

Среди вышеперечисленных задач особо следует выделить задачи так называемых конвейерных расписаний, в которых порядок выполнения операций для каждого задания одинаков.

Для решения задач оптимизации конвейерных расписаний теории расписаний используются различные методы, среди которых наибольший научно-практический интерес представляет метод ветвей и границ.

Таким образом, **актуальность** бакалаврской работы обусловлена необходимостью решения задач конвейерных расписаний с помощью метода ветвей и границ.

Объектом исследования бакалаврской работы является конвейерная задача теории расписаний.

Предмет исследования бакалаврской работы – метод ветвей и границ для конвейерной задачи теории расписаний.

Целью бакалаврской работы является анализ и применение метода ветвей и границ для решения конвейерной задачи теории расписаний.

Для достижения поставленной цели необходимо решить следующие задачи:

− проанализировать научную и учебно-методическую литературу по теме бакалаврской работы;

− описать метод ветвей и границ решения конвейерной задачи теории расписаний;

− формализовать конвейерную задачу теории расписаний;

− произвести анализ и представить пример применения алгоритма метода ветвей и границ для решения конвейерной задачи теории расписаний.

Методы исследования: теория расписаний, метод ветвей и границ.

Практическая значимость бакалаврской работы заключается в практическом применении алгоритмов метода ветвей и границ для решения конвейерной задачи теории расписаний.

Первая глава посвящена проблеме применения метода ветвей и границ для решения задач дискретной оптимизации. Описаны принципы метода ветвей и границ.

Во второй главе рассматривается формализация конвейерной задачи теории расписаний. Выполнена постановка задачи и описана математическая модель конвейерной задачи теории расписаний.

В третьей главе исследованы алгоритмы ветвей и границ для решения конвейерной задачи теории расписаний. Разработана и протестирована программа реализации алгоритма решения конвейерной задачи.

В заключении подводятся итоги исследования, формируются окончательные выводы по исследуемой проблемы.

В приложении представлен фрагмент программного кода.

Бакалаврская работа состоит из 46 страниц и включает 6 рисунков, 16 таблиц, 24 источника.

Глава 1 ПРИМЕНЕНИЕ МЕТОДА ВЕТВЕЙ И ГРАНИЦ ДЛЯ РЕШЕНИЯ ЗАДАЧ ДИСКРЕТНОЙ ОПТИМИЗАЦИИ

1.1 Задачи дискретной оптимизации

Дискретная оптимизация - это раздел математического программирования в прикладной математике и информатике.

В отличие от непрерывной оптимизации, некоторые или все переменные, используемые в дискретной оптимизации, ограничены множеством дискретных значений, например, целыми числами [9].

В дискретной оптимизации выделяются две группы задач:

− комбинаторная оптимизация, к которой относятся задачи на графах, матроидах и других дискретных структурах;

− целочисленное программирование.

Однако эти группы тесно переплетены, поскольку многие проблемы комбинаторной оптимизации могут быть смоделированы как целочисленные программы (например, кратчайший путь), и, наоборот, целочисленные программы часто могут иметь комбинаторную интерпретацию.

Для решения задач дискретной оптимизации используются методы дискретного программирования, раздела математики, предметом исследования которого является изучение и решение экстремальных задач на конечных множествах [15].

Пусть задано множество $M = \{a_1,\ldots,a_n\}$ и пусть f - числовая функция, определенная на элементах множества M.

Задача состоит в нахождении элемента $a_i \in M$, при котором достигается абсолютный минимум или максимум функции f.

Упрощенное описание данной задачи имеет вид:

$$f(x) \underset{x \in M}{\rightarrow} \text{extr (min или max)}.$$

Дискретное программирование имеет дело только с нетривиальными задачами в этом классе задач, которые удовлетворяют следующим дополнительным условиям:

1) число $n=|M|$ должно быть достаточно большим, чтобы проблема не была решена путем непосредственного перебора значений $f(a_i)$ вручную или с помощью электронного компьютера. Типичной задачей дискретного программирования является задача коммивояжера;

2) задача не должна быть регулярной.

Напомним, что регулярной задача является, если:

а) для каждого a_i,M существует непустая область $S(a_i,M)$ и $|S(a_i,M)|<<|M|$;

б) локальный экстремум функции f (т.е. $f(a_i)$ = extr $f(x)$, $x\in S(a_i,M)$) определяется с помощью простого алгоритма;

в) локальный экстремум функции f совпадает, по меньшей мере, с одним глобальным экстремумом.

Таким образом, дискретное программирование имеет дело с проблемами, включающими несколько локальных экстремумов.

В типичных случаях количество локальных экстремумов очень велико.

К такому классу задач относятся задачи целочисленного линейного программирования с булевыми переменными, в которых целевая функция и ограничения зависят от переменных $x_1,\ldots, x_k$.

К задачам дискретной оптимизации относится конвейерная задача теории расписаний, которая является объектом исследования настоящей бакалаврской работы.

Для решения задач дискретной оптимизации используются методы отсечения, динамического программирования, ветвей и границ.

Метод ветвей и границ является предметом исследования настоящей бакалаврской работы.

Рассмотрим особенности данного метода.

1.2 Метод ветвей и границ

Метод ветвей и границ (ВиГ) - это парадигма разработки алгоритмов для задач дискретной и комбинаторной оптимизации, а также для математической оптимизации.

Под методами ветвей и границ подразумевают алгоритмы перебора, представляющие собой фундаментальный общий метод нахождения оптимальных решений дискретных оптимизационных задач.

Алгоритм ветвления и границ состоит из систематического перечисления возможных решений посредством поиска в пространстве состояний: считается, что множество возможных решений образует корневое дерево с полным набором в корне.

Алгоритм исследует ветви этого дерева, которые представляют подмножества набора решений. Перед перечислением возможных решений ветви, ветвь проверяется по верхним и нижним оценочным границам на оптимальном решении и отбрасывается, если она не может дать лучшего решения, чем лучшее, найденное до сих пор алгоритмом.

Метод ВиГ был предложен А. Лэнд и А. Дойг в 1960г. [23].

Цель метода ВиГ состоит в том, чтобы найти значение x, которое максимизирует или минимизирует значение целевой функции $f(x)$, среди некоторого набора S допустимых или возможных решений.

Множество S называется пространством поиска или допустимой областью. В оставшейся части этого раздела предполагается, что минимизация функции $f(x)$ желательна; это предположение происходит без ограничения общности, поскольку можно найти максимальное значение $f(x)$, найдя минимум $g(x) = -f(x)$.

Метод ВиГ основан на двух принципах:

1) принцип ветвления. Пространство поиска рекурсивно разбивается на меньшие пространства (подзадачи), а затем минимизирует $f(x)$ на этих маленьких пространствах.

На рисунках 1.1 и 1.2 изображены графическое представление принципа ветвления и пример ветвления пространства поиска по методу ВиГ [13].

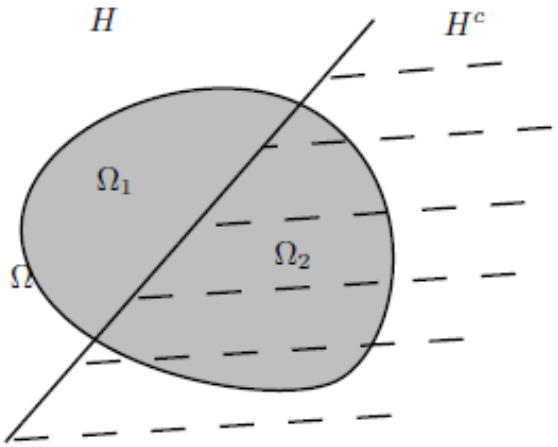

Рисунок 1.1 – Графическое представление принципа ветвления (H, H^c – полупространства поиска, $\Omega_1 = \Omega \cap H$ и $\Omega_2 = \Omega \cap H^c = \Omega \setminus H$ – ветви)

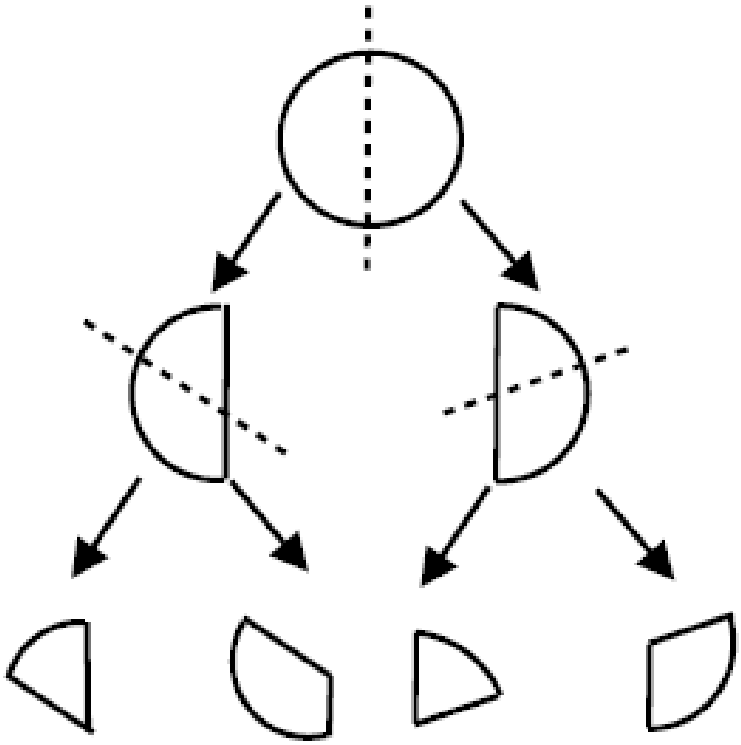

Рисунок 1.2 – Диаграмма процесса последовательного разбиения допустимой области поиска на более мелкие части по методу ВиГ

2) принцип границ. Производится вычисление нижних и/или верхних границ для значения целевой функции подзадачи.

Преобразование этих принципов в конкретный алгоритм для конкретной задачи оптимизации требует некоторой структуры данных, представляющей наборы возможных решений. Такое представление называется экземпляром проблемы.

На рисунке 1.3 представлен пример дерева поиска по методу ВиГ [14].

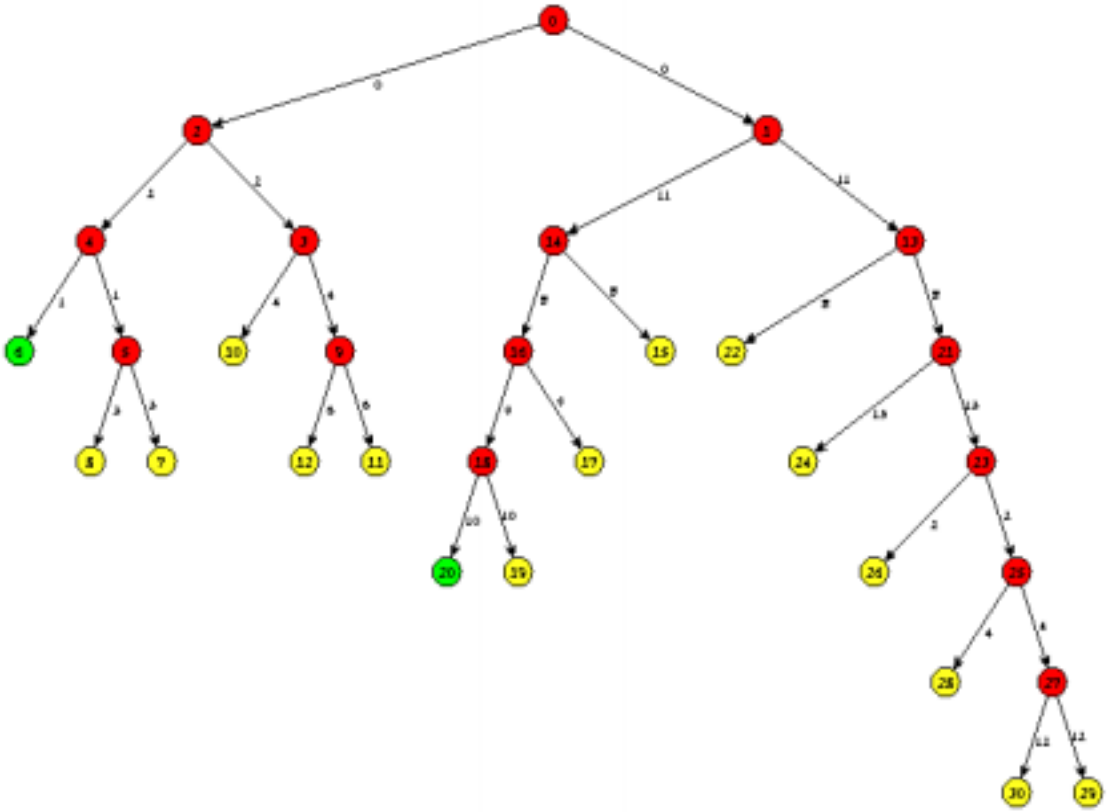

Рисунок 1.3 – Пример дерева поиска по методу ВиГ

Обозначим множество возможных решений экземпляра I через S_I.

Представление экземпляра должно сопровождаться тремя операциями:

1) создается два или более экземпляров, каждый из которых представляет подмножество S_I. Как правило, подмножества не пересекаются, чтобы предотвратить выработку алгоритмом одного и того же возможного решения дважды, но это не требуется. Однако оптимальное решение среди S_I должно содержаться по крайней мере в одном из подмножеств;

2) вычисляется нижняя граница для значений любого возможного решения в пространстве, представленном I, то есть границы $(I) \leq f(x)$ для всех x в S_I;

3) определяется, представляет ли I решение с одним кандидатом. При желании, если это не так, операция может решить вернуть какое-либо возможное решение из числа S_I.

Используя эти операции, алгоритм ВиГ выполняет рекурсивный поиск сверху вниз по дереву экземпляров, сформированному операцией ветвления.

При посещении экземпляра I он проверяет, больше ли граница (I), чем нижняя граница для какого-либо другого экземпляра, который он уже посетил; если это так, I может быть безопасно удален из поиска, и рекурсия прекращается. Этот этап сокращения обычно реализуется путем поддержания глобальной переменной, которая записывает минимальную нижнюю границу, наблюдаемую среди всех рассмотренных на данный момент экземпляров.

Рассмотрим универсальный алгоритм ВиГ [12].

Данный алгоритм состоит из следующих шагов:

Шаг 1. Используя эвристику, находим решение x_h задачи оптимизации. Сохраняем значение $B = f(x_h)$. (Если эвристика недоступна, установим $B = \infty$) B будет обозначать лучшее решение, найденное на данный момент, и будет использоваться в качестве верхней границы для возможных решений.

Шаг 2. Инициализируем очередь для хранения частичного решения, не указав ни одной из переменных задачи.

Шаг 3. Выполняем цикл, пока очередь не станет пустой:

1)выбираем узел N из очереди;

2)если N представляет единственное возможное решение x и $f(x) < B$, то x пока является лучшим решением. Сохраняем его и устанавливаем $B \leftarrow f(x)$;

3) в противном случае выполняем ветвление N, чтобы произвести новые узлы N_i. Для каждого из них:

− если граница (N_i)> B, ничего не меняем; поскольку нижняя граница этого узла больше верхней границы задачи, она никогда не приведет к оптимальному решению и может быть отброшена;

− иначе сохраняем N_i в очереди.

Можно использовать несколько различных структур данных очереди. Так, реализация на основе метода FIFO («первым зашел-первым вышел») обеспечивает поиск в ширину. Метод LIFO («последним зашел-первым вышел») выдает алгоритм глубины в первую очередь. Алгоритм с наилучшим первым ответвлением и привязкой может быть получен с использованием очереди приоритетов, которая сортирует узлы по их нижней границе.

Примерами алгоритмов поиска «лучший первый» в рассматриваемом контексте являются алгоритм Дейкстры и его вариации.

Чтобы получить из универсального алгоритма реальный алгоритм, требуется ограничивающая функция g, которая вычисляет нижние границы f для узлов дерева поиска, а также правило ветвления для конкретной задачи.

Таким образом, общий алгоритм, представленный здесь, является функцией более высокого порядка.

Рассмотрим примеры алгоритма целочисленного программирования на основе метода ВиГ [18].

1. Создание границ .

При исследовании подзадачи:

$$\max c^T x,\ Ax=b,\ x \geq 0,\ x_i \in \mathrm{Z} \text{ для } i \in I.$$

можно с помощью симплекс-метода решить следующую задачу:

$$\max c^T x,\ Ax=b,\ x \geq 0,\ x \in \mathrm{R}^n.$$

Если оптимальное значение x^* удовлетворяет целочисленным ограничениям, то мы находим оптимальное значения для всей подзадачи.

Таким образом, получим:

$$f_U^k = f_L^k = c^T x^*,$$

где:

f_U^k, f_L^k - верхняя и и нижняя границы, соответственно.

Эта ветвь не нуждается в дальнейшем рассмотрении.

Если оптимальное значение не удовлетворяет целочисленным ограничениям, то получим хотя бы одну верхнюю границу:

$$f_U^k = c^T x^*.$$

Если нет возможных решений, то ветвь не нуждается в дальнейшем рассмотрении.

2. Ветвление.

Если узел не считается сформированным, то это может быть ввиду того, что оптимальное значение не удовлетворяет целочисленным ограничениям.

Выбираем основную переменную, для которой $g = x_i^*$ не является целочисленным.

Разделяем задачу с помощью двух новых дополнительных ограничений:

$$x_i \geq [g] \text{ и } x_i \leq [g],$$

1) $\max c^T x$, $Ax=b$, $x_i \geq [g] + 1$, $x_i \geq 0$, $x \in Z$ для $i \in I$;

2) $\max c^T x$, $Ax=b$, $x_i \leq [g]$, $x_i \geq 0$, $x \in Z$ для $i \in I$.

Рассмотрим решение популярной задачи дискретной оптимизации об одномерном рюкзаке (0-1 knapsack) с помощью вышеописанного алгоритма.

Задача формально описывается следующим образом:

$$\max \{c^T x | \, a^T x \leq b, \, x \in \{0,1\}^n\}.$$

Предположим, что переменные были упорядочены следующим образом:

$$c_1/a_1 \geq c_2/a_2 \geq \ldots \geq c_n/a_n.$$

Пусть s будет максимальным индексом k таким, что:

$$\sum_{j=1}^{k} a_j \leq b$$

Согласно теореме Данцига (жадный алгоритм Данцига) оптимальное решение для непрерывной релаксации задачи о рюкзаке имеет вид:

$$w_j = 1, \, j = 1, \ldots, s$$

$$w_j = 0, j = s+2, \ldots, N$$

$$w_{s+1} = (b - \sum_{j=1}^{s} a_j) / a_{s+1}.$$

Если c_j, j =1, …, N - положительные целые числа, то верхняя граница оптимального значения определяется как:

$$UB = \sum_{j=1}^{s} c_j + [(b - \sum_{j=1}^{s} a_j) c_{s+1} / a_{s+1}].$$

На рисунке 1.4 представлен алгоритм Данцига для решения релаксационной задачи о рюкзаке.

1: Для каждого предмета $j = 1, 2, \ldots, n$, вычислим $\lambda_j = \dfrac{p_j}{w_j}$;
2: Перенумеруем требования согласно правилу $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$;
3: $F := 0$;
4: $C_0 := C$;
5: $x_i := 0,\ i = 1, 2, \ldots, n$;
6: **for** $j := 1$ to n **do**
7: **if** $C_0 \geq w_j$ **then**
8: $x_j := 1,\ C_0 := C_0 - w_j,\ F := F + p_j$;
9: **else**
10: Прерываем цикл;
11: **end if**
12: **end for**
13: $x_j := \dfrac{C - \sum_{i:=1}^{j-1} x_i w_i}{w_j}$;
14: $F := F + p_j x_j$;

Рисунок 1.4 - Алгоритм Данцига для решения задачи о рюкзаке

Необходимо отметить, что все известные алгоритм ВиГ для задачи о рюкзаке имеют экспоненциальную трудоемкость.

К преимуществам метода ВиГ можно отнести обобщенность, адаптивность и применение для решения широкого круга задач дискретной оптимизации.

Выводы к главе 1

1) Для решения задач дискретной оптимизации используются методы дискретного программирования, раздела математики, предметом исследования которого является изучение и решение экстремальных задач на конечных множествах.

2) Метод ВиГ основан на двух принципах: принципе ветвления и принципе границ.

3) Преимуществами метода ВиГ являются обобщенность, адаптивность и применение для решения широкого круга задач дискретной оптимизации, в том числе, для решения конвейерной задачи теории расписания.

Глава 2 ФОРМАЛИЗАЦИЯ КОНВЕЙЕРНОЙ ЗАДАЧИ ТЕОРИИ РАСПИСАНИЙ

2.1 Постановка конвейерной задачи теории расписаний

Расписание - это процесс организации, контроля и оптимизации работы и рабочих нагрузок в производственном процессе. Компании используют расписание для распределения ресурсов машин и оборудования, планирования человеческих ресурсов, планирования производственных процессов и закупки материалов.

Это важный инструмент для производства и проектирования, где он может оказать существенное влияние на производительность процесса. При производстве цель составления расписания состоит в том, чтобы минимизировать время и затраты на производство, сообщая производственному предприятию, когда производить, с каким персоналом и на каком оборудовании. Производственное расписание направлено на максимизацию эффективности работы и снижение затрат.

Расписание является предметом исследования теории расписаний.

Теория расписаний – это раздел исследования операций, в котором строятся и анализируются математические модели календарного планирования (упорядочивания во времени) различных целенаправленных действий с учетом целевой функции и различных ограничений [5].

Проблемы, с которыми сталкивается теория расписаний, обычно формулируются как задачи оптимизации для процесса обработки конечного множества заданий (работ) в системе с ограниченными ресурсами.

Конечные множество работ - это то, что отличает модели теории расписаний от аналогичных моделей в теории очередей, где рассматриваются в основном бесконечные потоки действий. Во всем остальном отправные точки двух теорий близки. В теории расписаний указывается время поступления каждого задания в систему. В рамках системы задание должно проходить несколько этапов обработки, в зависимости от условий задачи.

Следует отметить, что, будучи практически ориентированной, теория расписаний не стремится унифицировать свою терминологию; наряду с термином «работа» также используются такие термины, как «задача», «операция» и т. д. По той же причине график обработки формально определяется несколькими способами.

В общем случае расписание можно рассматривать как однозначное отображение, которое каждой работе в каждый момент времени присваивает определенный набор ресурсов.

Обобщенно задача оптимизации в теории расписаний может быть формализована следующим образом [15]:

$$F(s) = \max_{\alpha \in N} \phi_\alpha(t_\alpha(s)),$$

где:

$t_\alpha(s)$ - время завершения работы α в соответствии с расписанием s;

N – множество работ;

ϕ_α - неубывающая функция, называемая функцией затрат.

Среди задач теории расписаний следует выделить группу задач, именуемых задачами цеха (Shop Scheduling).

Задача цеха - это проблема оптимизации в компьютерных науках и исследованиях операций, когда рабочие места получают доступ к ресурсам в определенное время.

Наиболее стандартная версия задачи цеха имеет следующий вид.

Дано n работ J_1, J_2, ..., J_n. Каждая работа содержит набор операций O_{1j}, O_{2j}, ..., O_{nj}, которые необходимо обрабатывать в определенном порядке (так называемые ограничения приоритетов). Каждая операция выполняется на определенной машине, и только одна операция в работе может быть выполнена в данный момент времени. Всего дано m машин $\{M_1,M_2,\ldots,M_n\}$. Время выполнения операции O_{ij} равно p_{ij}, причем она может выполняться на машине $\mu_{ij} \in \{M_1, M_2,\ldots, M_m\}$.

Необходимо минимизировать время выполнения работы.

Одним из частных случаев задачи цеха является конвейерная задача теории расписаний (Flow-shop) [2].

В постановке данной задачи каждая работа состоит из одних и тех же операций, $n_j = m$, $\forall j \in N$, а также задана машина, на которой обслуживаются операции:

$$\mu_{ij} = M_i,\ i = 1,\ldots, m,\ j = 1,\ldots, n.$$

Тогда расписание для каждой машины задается вектором – порядком обслуживания операций, относящихся к разным работам. В особенности желательно поддержание непрерывного потока задач обработки с минимумом времени простоя и минимумом времени ожидания.

Требуется определить для каждой машины порядок выполнения работ, минимизирующий общее время выполнения всех работ на всех машинах.

Эта задача, являющаяся одной из наиболее известных в теории расписаний, впервые была сформулирована Р. Беллманом [19].

2.2 Математическое моделирование конвейерной задачи

Рассмотрим математическую модель, формализующую постановку конвейерной задачи для процесса металлообработки [6].

Введем следующие обозначения:

i – номер станка; j – номер детали;

$\boldsymbol{T}$ - действительная матрица размерности m x n, элемент которой $t_{ij} \geq 0$ задает время выполнения детали j на станке i, где $i = 1,2,\ldots, m;\ j = 1,2,\ldots, n$.

Необходимо найти матрицу $\boldsymbol{X}$ размерности m x n, элемент которой x_{ij} задает момент начала выполнения детали j на станке i, где $i = 1,2,\ldots, m;\ j = 1,2,\ldots, n$ для следующих ограничений:

1) обработка детали на станке может начаться не раньше, чем эта деталь завершит обработку на предыдущем станке:

$$x_{ij} \geq x_{i-1\,j} + t_{i-1\,j}, i = \overline{2,m};\ j = \overline{1,n}; \quad (2.1)$$

2) на станке одновременно не может обрабатываться более одной детали:

$$x_{ij} \geq x_{ik} + t_{ik},\ \textit{или}\ \ x_{ik} \geq x_{ij} + t_{ij}, \quad i = \overline{1,m};\ j=\overline{1,n};\ k=\overline{1,n};, \quad (2.2)$$

3) порядок обработки деталей один и тот же для всех станков:

$$\textit{если}\ x_{ij} \geq x_{ik},\ \textit{то}\ x_{sj} \geq x_{sk},\ s = \overline{1,m};\ j = \overline{1,n};, \quad (2.3)$$

4) естественные условия на переменные:

$$x_{ij} \geq 0,\ i = \overline{1,m};\ j = \overline{1,n}; \quad (2.4)$$

Критерием оптимальности является минимизация времени завершения обработки деталей:

$$F(X) = \min_{j=1,n}(x_{mj} + t_{mj}) \quad (2.5)$$

Выражения (2.2), (2.3) и (2.5) являются формализацией постановки конвейерной задачи.

Следует отметить, что критерий (2.5) является нелинейным.

Для применения алгоритмов ВиГ для решения рассматриваемой задачи необходимо свести данную задачу к задаче дискретной оптимизации.

Введем следующие ограничения для базовой модели конвейерной задачи [20]:

$$\sum_{k=1}^{M_j} Y_{ijk} = 1,\ \text{для всех}\ i\ \text{и}\ j; \quad (2.6)$$

$$C_{ij} - C_{i,\,\text{j}-1} \geq \sum_{k=1}^{M_j} Y_{ijk}\, p_{ijk} + t_{ij},\ \ \text{для всех}\ i\ \text{и}\ j; \quad (2.7)$$

$$Q(2-Y_{ijk}-Y_{rjk}+X_{irj})+C_{ij}-C_{rj}\geq p_{ijk},\ \text{для всех } i,j,r \text{ и } k; \quad (2.8)$$

$$Q(3-Y_{ijk}-Y_{rjk}+X_{irj})+C_{rj}-C_{ij}\geq p_{ijk},\ \text{для } i<r$$

$$Y_{ijk}=\{0,1\} \text{ для всех } i,j \text{ и } k;$$

$$X_{irj}=\{0,1\} \text{ для всех } i,r,j \text{ и } k, \text{ причем } i<r;$$

$$C_{ij}\geq 0 \text{ для всех } i,j; \quad (2.9)$$

$$C_{i0}=r_i \text{ для всех } i,$$

где:

n – количество работ;

m – количество этапов обработки;

$i = 1, 2,\ldots, n$;

$j = s_{i1}, s_{i2},\ldots, m$, причем $s_{ij} = j$;

k=1,2,…, M_j – количество параллельно работающих станков на j –ом этапе;

p_{ijk} – время выполнения i–ой работы на k-ом станке;

t_{ij} – время перехода между этапами j-1 и j;

C_{ij} – время окончания i–ой работы на j-ом этапе;

$C_{i0}=r_i$ – время выпуска i–ой работы;

Q – очень большое число;

X_{irj} равно 1, если i –я работа предшествует r-ой работе на j-ом этапе на k-ой машине, и равно 0 в противном случае;

Y_{ijk} равно 1, если i –я работа на j-ом этапе выполняется на k-ой машине, и равно 0 в противном случае.

Формулировка предполагает выполнение работы на отдельных станках на каждом этапе обработки.

В данной модели конвейерной задачи используются булевы переменные {0,1}.

Переменная X_{irj} используется для учета отношений приоритета между работами на каждом j-ом этапе обработки. Переменная Y_{ijk} используется для назначения работ только одному k-ому станку на каждом этапе. Другими

словами, эта переменная служит для обеспечения защиты от многократного назначения обработки на каждом этапе.

Таким образом, чтобы гарантировать назначение каждой работы одному и только одному станку на каждом этапе, уравнение (2.6) должно выполняться для любого возможного расписания.

Необходимо учесть, что невозможно (или не разрешено) начать выполнение работ на следующем этапе, если они не завершены на предыдущем этапе.

Если обозначить через Z общее время выполнения всех работ на всех машинах, конвейерная задача может быть приведена к виду:

$$F(X,Y,C) = Z \to \min$$

Вышеупомянутые ограничения описываются уравнением (2.7) для всех заданий на всех этапах обработки.

Кроме того, разница между временем обработки любых двух заданий, назначенных одному и тому же станку, должна быть такой, чтобы они не перекрывались согласно (2.8). Следует отметить, что если отношение приоритета некоторых из работ или набора работ известно и зафиксировано, то соответствующие ограничения в выражении (2.8) будут либо устранены, либо упрощены.

Рассмотрим частные случаи представленной модели.

1) Ограничения для модели с одной машиной.

Любой порядок работ будет соответствовать критериям максимального завершения и времени выполнения, в то время как правило первого приоритета с кратчайшим временем обработки обеспечит оптимальное расписание для критериев минимального среднего завершения и времени выполнения, а порядок упорядочения по наиболее ранней дате выполнения оптимизируется. минимизация максимальной задержки.

Однако существует несколько других критериев для случаев, когда времена выпуска являются нулями или константами, и для большинства других случаев, когда алгоритмические решения недоступны. В таких

обстоятельствах математическая формулировка может быть полезна для решения задач для оптимальных или почти оптимальных решений.

В случае одиночной машины, поскольку нет ни другой стадии обработки, ни другой машины в своем роде, поэтому индексы *j* и *k* не нужны.

Упрощенная модель для одной машины выглядит следующим образом:

$$C_{i1} \geq r_i + p_i \text{ для всех } i; \qquad (2.10)$$

$$Q(X_{ir}) + C_{i1} - C_{r1} \geq p_i;$$

$$Q(1\text{-}X_{ir}) + C_{r1} - C_{i1} \geq p_r \text{ для всех } i,r, \text{ причем } i < r;$$

$$X_{ir} = \{0,1\} \text{ для всех } i,r, \text{ причем } i < r;$$

$$C_{i1} \geq 0 \text{ для всех } i.$$

Следует учесть, что если время освобождения всех заданий равно нулю, ограничение (1.10) является избыточным.

2) Ограничения для модели параллельной машины.

В случае параллельной машины есть один этап обработки. Однако на этом этапе есть параллельные машины. Поэтому при рассмотрении базовой модели индекс *j* не является необходимым. Это означает, что в дополнение к ограничению целевой функции для задачи планирования параллельной машины достаточно следующих ограничений:

$$C_{i1} \geq r_i + p_i \text{ для всех } i; \qquad (2.11)$$

$$\sum_{k=1}^{M_j} Y_{ik} = 1, \text{ для всех } i;$$

$$Q(2 - Y_{ik} - Y_{rk} + X_{ir}) + C_{i1} - C_{r1} \geq p_{ik}, \text{ для всех } i, j, r, k;$$

$$Q(3 - Y_{ik} - Y_{rk} + X_{ir}) + C_{r1} - C_{i1} \geq p_{rk}, \text{ для } i < r \qquad (2.12)$$

$$Y_{ik} = 0,1 \text{ для всех } i \text{ и } k;$$

$$X_{ir} = 0,1 \text{ для всех } i,r \text{ и } k, \text{ причем } i < r;$$

$$C_{i1} > 0 \text{ для всех } i.$$

Аналогично, если время освобождения всех работ равно нулю, то ограничение (2.11) является избыточным.

Кроме того, для задачи расписания максимального времени завершения, когда времена выпуска всех работ являются нулями или константами, проблему можно легко упростить. В этой ситуации приоритет среди работ на параллельном процессоре не имеет никакого значения.

Поэтому, отбрасывая переменную X_{ir}, множество ограничений в (2.12) можно заменить следующим простым ограничением:

$$\sum_{i=1}^{n} Y_{ik}\, p_{ik} \le Z, \text{ для всех } k \qquad (2.13)$$

Таким образом, использования булевых переменных и введенные ограничения позволяют свести конвейерную задачу к задаче дискретной оптимизации.

Выводы к главе 2

1) Конвейерная задача теории расписаний является одним из частных случаев задачи цеха.

2) Типовая конвейерная задача теории расписаний является задачей нелинейной оптимизации. Для применения алгоритмов ВиГ необходимо привести рассматриваемую задачу к задаче дискретной оптимизации.

3) Использование булевых переменных и введенные ограничения позволяют свести типовую конвейерную задачу теории расписаний к задаче дискретной оптимизации.

4) Метод ВиГ он обладает тем свойством, что при помощи определения границ критерия оценки, допустимости или доминирования позволяет исключать из рассмотрения определенные последовательности еще до того, как для этих последовательностей был вычислен критерий оценки расписания. Благодаря этому метод ВиГ является практически приемлемым для отыскания оптимальных последовательностей.

Глава 3 АЛГОРИТМ ВЕТВЕЙ И ГРАНИЦ ДЛЯ РЕШЕНИЯ КОНВЕЙЕРНОЙ ЗАДАЧИ ТЕОРИИ РАСПИСАНИЙ

Рассмотрим применение метода ветвей и границ на примере известных алгоритмов для решения конвейерных задач теории расписаний.

3.1 Алгоритм ветвей и границ для решения задачи Джонсона для двух станков

Рассмотрим постановку задачи Джонсона для двух станков [22].

Дано:

1) N_{12} – множество работ, с технологическим маршрутом $(1 \rightarrow 2)$;

2) для каждой работы заданы p^1_i и p^2_i – время обслуживания на 1-ом и 2-ом станке, соответственно;

3) целевая функция – суммарное (общее) время обслуживания всех работ, которое требуется минимизировать.

На условия оптимизации наложены следующие ограничения:

− на каждом станке в каждый момент времени выполняется не более одной работы;

− каждая работа может начать выполняться на 2-ом станке, только если она завершилась на 1-ом станке, т.е. момент окончания обслуживания на 1-ом станке не должен превышать момента начала работы на 2-м станке;

− непрерывность работ: если i-тое требование начало обслуживаться в момент времени t на *j*-ом станке, то оно должно обслуживаться на том же станке до момента времени $t + p^j_i$.

Ниже представлены описания классических алгоритмов Джонсона для решения данной задачи.

1. Алгоритм Джонсона 1.

На входе алгоритма – множество работ, на выходе – перестановка π.

Шаг 1.

$\forall i \;\; p_i := \min (p^1_i, p^2_i)$

Упорядочиваем работы следующим образом:

$p_{i1} \le p_{i2} \le \ldots \le p_{in}$ (3.1)

Шаг 2.

$\pi_1 := \varnothing$: $\pi_2 := \varnothing$

Шаг 3.

Пусть N' – упорядоченное по правилу (3.1) множество работ.

Если N'= $\varnothing$, то переходим на ш. 4.

Берем 1-й элемент из множества *N*', и если $p_i = p^1_i$, то $\pi_i := \pi_1 U\{i\}$, иначе, если $p_i = p^2_i$, $\pi_2 := \pi_2 U\{i\}$.

Исключаем i из N'.

Шаг 4.

$\pi := \pi_1 U \pi_2$.

Конец.

2. Алгоритм Джонсона 2.

На входе перестановка π; на выходе – допустимое расписание.

Шаг 1.

Время запуска 1-й работы на 1-ом станке равно 0. Время запуска каждой следующей работы на 1-ом станке равно времени окончания обслуживания предыдущей работы.

Шаг 2.

Время запуска 1-й работы на 2-ом станке равно времени окончания этой же работы на 1-ом станке. Время запуска каждой следующей работы на 2-ом станке равно минимуму из времени окончания обслуживания предыдущей работы и времени окончания обслуживания последней на 1-ом станке.

Конец.

Для решения задачи Джонсона по вышеописанному алгоритму используем онлайн-сервис «Все калькуляторы онлайн (рисунок 3.1) [8].

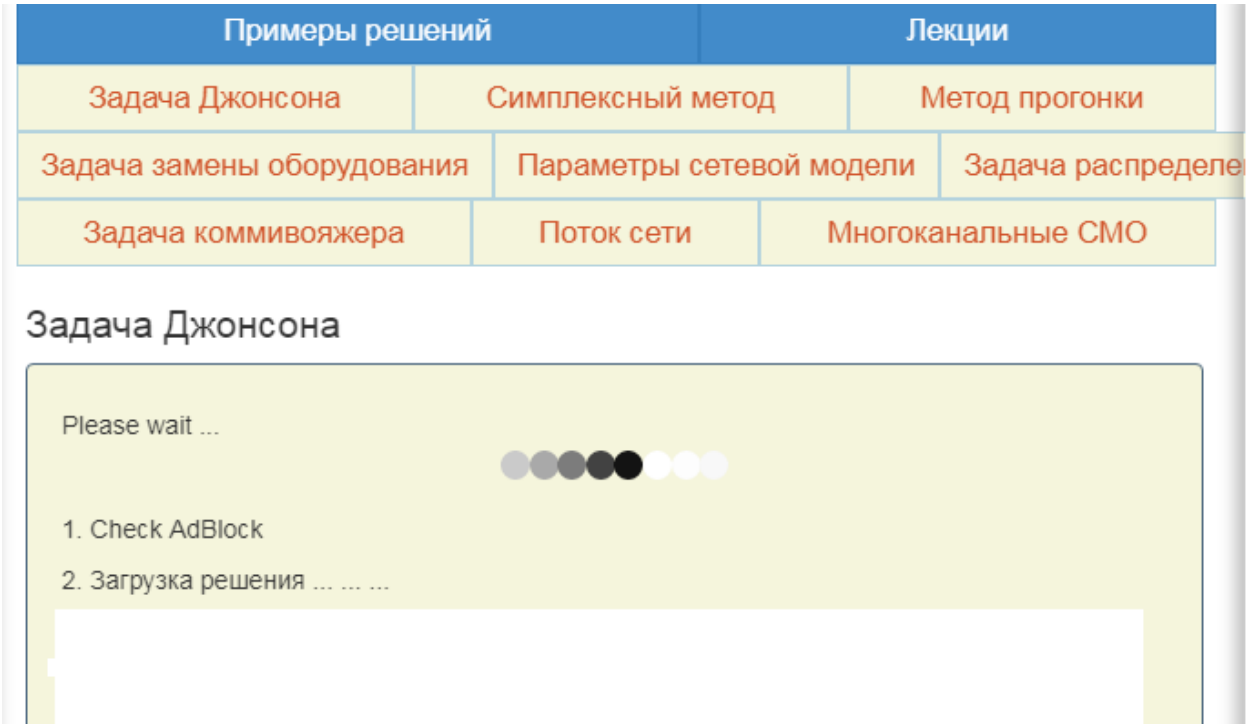

Рисунок 3.1– Экранная форма онлайн-сервиса «Все калькуляторы онлайн» для решения задачи Джонсона

Рассмотрим задачу последовательной обработки на двух машинах N различных деталей, если известно время A_i и B_i обработки *i*-й детали на соответствующих машинах.

Условие задачи представлено в таблице 3.1 [3]:

Таблица 3.1 – Исходные данные задачи Джонсона

i	1	2	3	4	5	6	7
A	5	7	4	3	5	7	6
B	6	5	6	7	4	6	8

Необходимо найти порядок обработки π, минимизирующий время простоя второй машины и тем самым сокращающий общее время обработки деталей.

Далее представлено решение данной задачи, предлагаемое онлайн-сервисом «Все калькуляторы онлайн».

Данные загружается из книги Excel.

Если обозначить через X_i - время простоя в ожидании *i*-й детали, то:

$X_1 + X_2 = \max(A_1 + A_2 - B_1, A_1)$

$X_1 + X_2 + X_3 = \max(A_1 + A_2 + A_3 - B_1 - B_2, A_1 + A_2 - B_1, A_1)$

$\sum X_i = \max(\sum A_i - \sum B_i)$

Если обозначить через $F(t, A_k, B_k/k=1..N)$ - суммарное время обработки N деталей при условии, что вторая машина включается с задержкой t и используется оптимальный порядок обработки, то с учетом принципа оптимальности (независимо от выбора начальной детали порядок выбора последующих должен быть оптимальным) имеем:

$$F(t, A_k, B_k/k = 1..N) = \min(A_i + F(B_i + \max(t-A_i,0), A_k, B_k=1..N, k\neq i))$$

Если после *i*-й детали при оптимальном порядке обрабатывается *j*-я, то:

$$F(t, A_k, B_k/k=1..N) = A_i + A_j + F(t_{ij}, A_k, B_k/k=1..N;\ k\neq i,j),$$

где:

$$t_{ij} = B_i + \max[B_j + \max(t-A_j,0) - A_j,0] = B_i + B_j - A_i - A_j + \max[t,$$
$$\max(t,\max(A_j - A_i - B_j, A_j))].$$

Если $\max(A_i + A_j - B_i, A_i) < \max(A_j + A_i - B_j, A_j)$, то сначала разумнее обрабатывать *j*-ю деталь.

Можно показать, что указанное условие необходимости перестановки эквивалентно условию:

$$\min(A_j, B_i) < \min(A_i, B_j)$$

Соответственно ищем среди всех значений A_i и B_i наименьшее. Если найденное значение совпадает с некоторым A_i, то *i*-ю деталь ставим на обработку первой; если оно совпадает с некоторым B*i*, то последней.

Эту процедуру повторяем для всех остальных деталей.

Шаг № 1.

Минимальное из значений соответствует A_4: 4-ая деталь обрабатывается первой (таблица 3.2).

Таблица 3.2 – Промежуточное расписание (шаг 1)

A	B
3	7

A	B
-	-
-	-
-	-
-	-
-	-
-	-

Шаг № 2.

Минимальное из значений соответствует A_3: 3-ая деталь обрабатывается первой (таблица 3.3).

Таблица 3.3 – Промежуточное расписание (шаг 2)

A	B
3	7
4	6
-	-
-	-
-	-
-	-
-	-

Шаг № 3.

Минимальное из значений равно 4 и соответствует B_5: 5-ая деталь обрабатывается последней (таблица 3.4).

Таблица 3.4 – Промежуточное расписание (шаг 3)

A	B
3	7
4	6
-	-
-	-
-	-

A	B
-	-
5	4

Шаг № 4.

Минимальное из значений соответствует A_1: 1-ая деталь обрабатывается первой (таблица 3.5).

Таблица 3.5 – Промежуточное расписание (шаг 4)

A	B
3	7
4	6
5	6
-	-
-	-
-	-
5	4

Шаг № 5.

Минимальное из значений равно 5 и соответствует B_2: 2-ая деталь обрабатывается последней (таблица 3.6).

Таблица 3.6 – Промежуточное расписание (шаг 5)

A	B
3	7
4	6
5	6
-	-
-	-
7	5
5	4

Шаг № 6.

Минимальное из значений соответствует A_7: 7-ая деталь обрабатывается первой (таблица 3.7).

Таблица 3.7 – Промежуточное расписание (шаг 6)

A	B
3	7
4	6
5	6
6	8
-	-
7	5
5	4

Шаг № 7.

Минимальное из значений равно 6 и соответствует B_6: 6-ая деталь обрабатывается последней (таблица 3.8).

Таблица 3.8 – Промежуточное расписание (шаг 7)

A	B
3	7
4	6
5	6
6	8
7	6
7	5
5	4

В итоге упорядоченная информация принимает вид (таблица 3.9):

Таблица 3.9 – Итоговое расписание

A	B
3	7
4	6
5	6
6	8
7	6

A	B
7	5
5	4

Результат: оптимальная последовательность работ π = (4, 3, 1, 7, 6, 2, 5).

Далее рассмотрим решение описанной задачи с помощью алгоритма ветвей и границ.

3.2 Реализация алгоритма ветвей и границ для решения конвейерной задачи теории расписания

Блок-схема обобщенного алгоритма решения задачи Джонсона, использующего методику вычисления совокупности нижних границ (нижних оценок) по методу ветвей и границ, представлена на рисунке 3.1 [24].

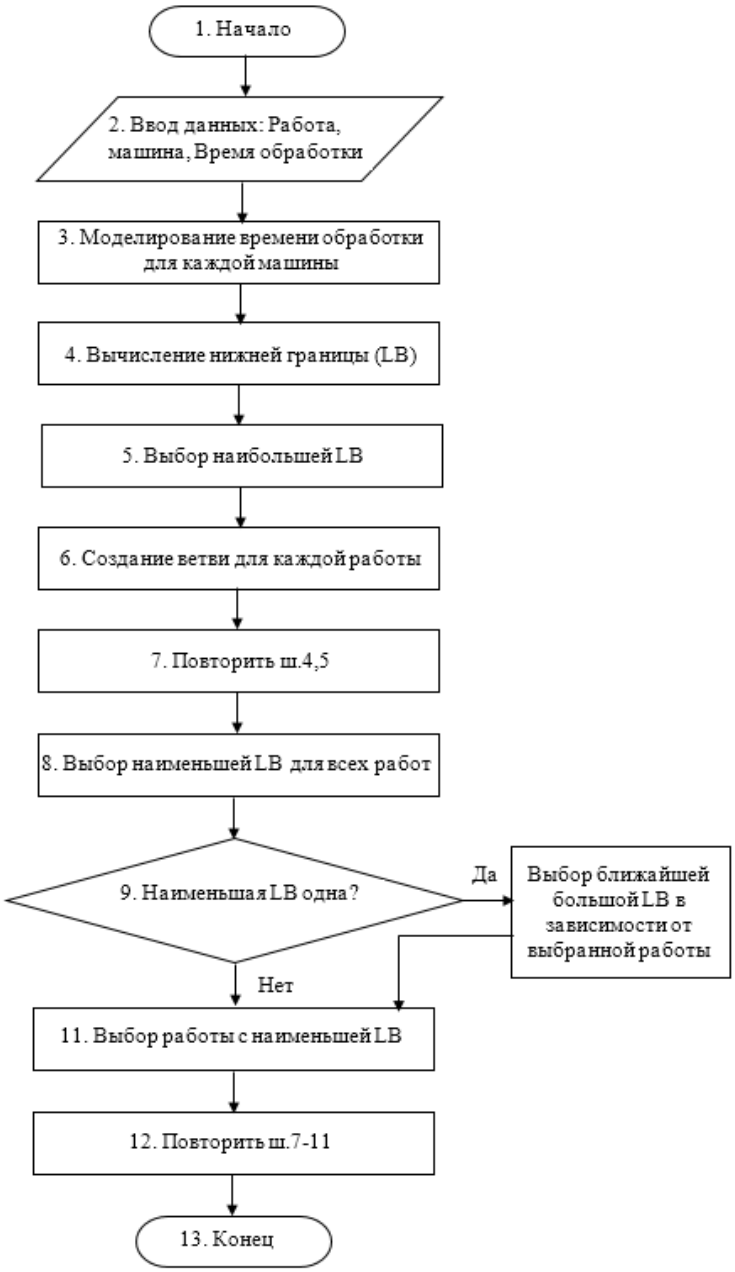

Рисунок 3.1- Блок-схема алгоритма ветвей и границ для решения конвейерной задачи теории расписаний

Рассмотрим пример применения алгоритма с использованием метода ветвей и границ для двухступенчатой конвейерной задачи теории расписаний с интервалом разбивки [21].

Введем обозначения:

A_i – время выполнения i-ой работы на станке A;

B_i – время выполнения i-ой работы на станке B;

S_k – последовательность работ;

L – длина интервала разбивки;

J_r - частичное расписание r запланированных работ

$J_{r'}$ - множество оставшихся (n-r) свободных работ.

Итак, мы имеем n работ (i = 1, 2, 3… n), которые обрабатываются на двух станках A и B в порядке AB.

Цель состоит в получении оптимального расписания, которое минимизирует итоговое время работ на интервале разбивки (a, b), используя метод ветвей и границ.

Алгоритм состоит из следующих шагов:

Шаг 1. Вычисляем:

$$l_1 = (t, J_1)_r + \sum_{i \in J_{r'}} A_i + \min_{i \in J_{r'}}(B_i)$$

$$l_2 = (t, J_2)_r + \min_{i \in J_{r'}}(B_i)$$

Шаг 2. Вычисляем:

$l = \max [l_1, l_2]$

Сначала оценивается l для n классов перестановок, то есть для начинающихся с 1,2,3,…, n, соответственно. Далее пометим соответствующие значения вершин дерева расписания этими значениями.

Шаг 3. Теперь исследуем вершину с самой низкой меткой. Оцениваем l для (n-1) подклассов, начиная с этой вершины, и снова сконцентрируемся на самой нижней вершине метки. Продолжаем, пока не дойдем до конца дерева, представленного двумя отдельными перестановками, для которых оценим общую продолжительность работы. Таким образом, получим оптимальное расписание работ.

Шаг 4. Подготовим таблицу входов и выходов для оптимальной последовательности, полученной на шаге 3, и выявляем влияние интервала разбивки (a, b) на различные работы.

Работа алгоритма завершена.

Код универсального алгоритма ветвей и границ для решения конвейерной задачи на языке Java представлен ниже:

/********** Flowshop.java **********/

```
import java.util.*;
import java.io.*;
import localsolver.*;
public class Flowshop {
    // Количество работ
    private int nbJobs;
    // Количество машин
    private int nbMachines;
    private long initialSeed;
    // Верхняя граница
    private int upperBound;
    // Нижняя граница
    private int lowerBound;
    // Время обработки
    private long[][] processingTime;
    // LocalSolver
    private LocalSolver localsolver;
    private LSExpression jobs;
    private LSExpression makespan;
    private Flowshop(LocalSolver localsolver) {
        this.localsolver = localsolver;
    }
    // Читает данные экземпляра.
    private void readInstance(String fileName) throws IOException {
        try (Scanner input = new Scanner(new File(fileName))) {
            nbJobs = input.nextInt();
            nbMachines = input.nextInt();
            initialSeed = input.nextInt();
            upperBound = input.nextInt();
            lowerBound = input.nextInt();
```

```
                processingTime = new long[nbMachines][nbJobs];
                for (int m = 0; m < nbMachines; m++) {
                    for (int j = 0; j < nbJobs; j++) {
                        processingTime[m][j] = input.nextInt();
                    }
                }
            }
        }
        private void solve(int limit) {
            // Задает модель оптимизации.
            LSModel model = localsolver.getModel();
            // Перестановка работ
            jobs = model.listVar(nbJobs);
            // Все работы должны быть распределены
            model.constraint(model.eq(model.count(jobs), nbJobs));
                LSExpression[] processingTimeArray = new
LSExpression[nbMachines];
            for (int m = 0; m < nbMachines; m++) {
                processingTimeArray[m] = model.array(processingTime[m]);
            }
            // На машине 0 j-е задание заканчивается в то время, которое
            // потребовалось для обработки после завершения предыдущей
        работы
            LSExpression[] end = new LSExpression[nbJobs];
            LSExpression firstEndSelector = model.function((i, prev) ->
model.sum(
                    prev, model.at(processingTimeArray[0], model.at(jobs, i))));
            end[0] = model.array(model.range(0, nbJobs), firstEndSelector);
            // J-я работа на машине m запускается, когда она было выполнена
        на машине n-1
```

```
for (int m = 1; m < nbMachines; ++m)
        {
            final int mL = m;
            LSExpression endSelector = model.function((i, prev) -> model.sum(
                    model.max(prev, model.at(end[mL - 1], i)),
                    model.at(processingTimeArray[mL], model.at(jobs, i))));
            end[m] = model.array(model.range(0, nbJobs), endSelector);
        }
        // Минимизирует время выполнения
        makespan = model.at(end[nbMachines - 1], nbJobs - 1);
        model.minimize(makespan);
        model.close();
        // Параметризует решатель.
        localsolver.getParam().setTimeLimit(limit);
        localsolver.solve();
    }
    // Записывает решение в файл
    private void writeSolution(String fileName) throws IOException {
        try(PrintWriter output = new PrintWriter(fileName)) {
            output.println(makespan.getValue());
            LSCollection jobsCollection = jobs.getCollectionValue();
            for (int j = 0; j < nbJobs; j++) {
                output.print(jobsCollection.get(j) + " ");
            }
            output.println();
        }
    }
    public static void main(String[] args) {
        if (args.length < 1) {
```

```
            System.err.println("Usage: java Flowshop inputFile [outputFile] [timeLimit]");
            System.exit(1);
        }
        String instanceFile = args[0];
        String outputFile = args.length > 1 ? args[1] : null;
        String strTimeLimit = args.length > 2 ? args[2] : "20";
        try (LocalSolver localsolver = new LocalSolver()) {
            Flowshop model = new Flowshop(localsolver);
            model.readInstance(instanceFile);
            model.solve(Integer.parseInt(strTimeLimit));
            if (outputFile != null) {
                model.writeSolution(outputFile);
            }
        } catch (Exception ex) {
            System.err.println(ex);
            ex.printStackTrace();
            System.exit(1);
        }
    }
}
```

Данный алгоритм используется в специализированном онлайн-сервисе «LocalSolver» (рисунок 3.2) [16].

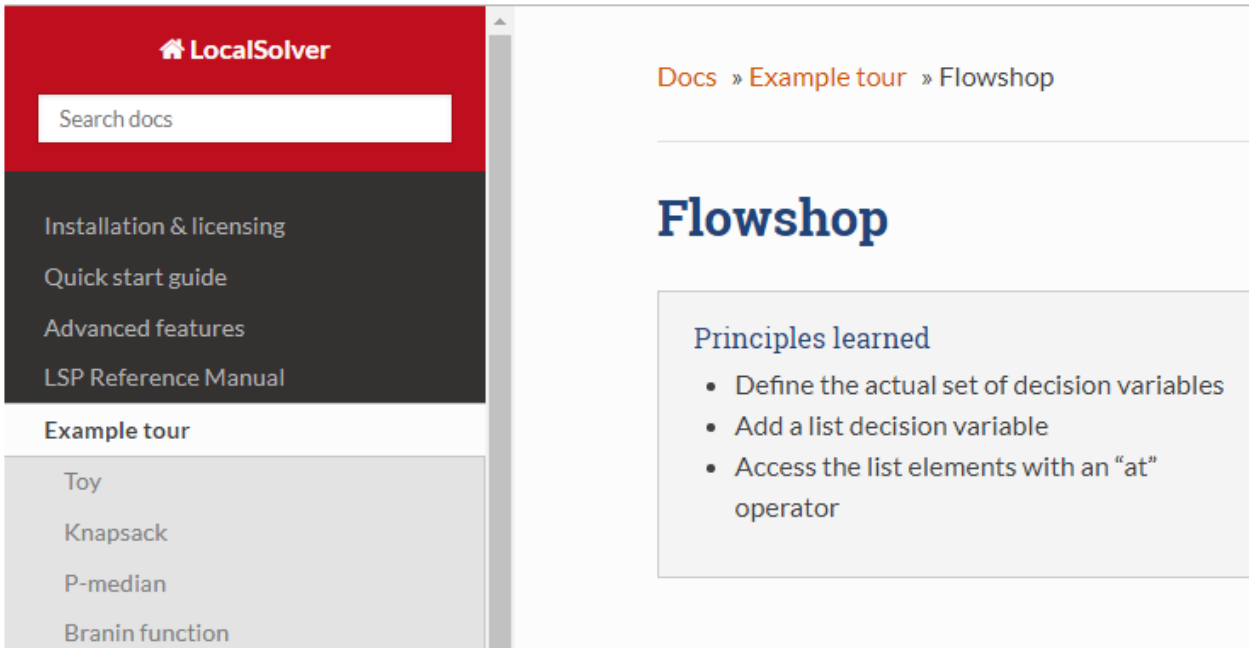

Рисунок 3.2 – Экранная форма онлайн-сервиса LocalSolver

Для разработки программы на основе данного онлайн-сервиса использована технология Electron, позволяющая создавать кроссплатформенные приложения при помощи JavaScript, HTML и CSS. Для обмена данными с веб-сервисом используется формат JSON [10].

Фрагмент кода программы представлен в Приложении А.

Решение задачи (таблица 3.2) представлено в таблицах 3.10-3.16.

Таблица 3.10 – Решение задачи, шаг 1

i	1	2	3	**4**	5	6	7
A	5	7	4	**3**	5	7	6
B	6	5	6	7	4	6	8

π =(4,,,,,,)

Таблица 3.11 – Решение задачи, шаг 2

i	1	2	3	**5**	6	7
A	5	7	**4**	5	7	6
B	6	5	6	**4**	6	8

π =(4,,,,,5)

Таблица 3.12 – Решение задачи, шаг 3

i	1	2	3	**5**	6	7
A	5	7	**4**	5	7	6
B	6	5	6	**4**	6	8

π =(4,,,,,,5)

Таблица 3.13 – Решение задачи, шаг 4

i	1	2	**3**	6	7
A	5	7	**4**	7	6
B	6	**5**	6	6	8

π =(4,3,,,,,5)

Таблица 3.14 – Решение задачи, шаг 5

i	1	**2**	6	7
A	**5**	7	7	6
B	6	**5**	6	8

π =(4,3,,,,2,5)

Таблица 3.15 – Решение задачи, шаг 6

i	**1**	6	7
A	**5**	7	6
B	**6**	**6**	8

π =(4,3,1,,,2,5)

Таблица 3.16 – Решение задачи, шаг 7

i	6	**7**
A	7	**6**
B	**6**	8

Результат: оптимальная последовательность работ π = (4, 3, 1, 7, 6, 2, 5).

Полученный результат совпал с результатом решения с помощью онлайн-сервиса, использующего классический алгоритм Джонсона.

Это подтверждает правильность программного кода алгоритма, основанного на методе ветвей и границ.

Выводы к 3-й главе

1) Для проверки правильности программного кода алгоритма ветвей и границ для решения конвейерной задачи с помощью технологии Electron разработана программа. Далее выполнено сравнение результата решения задачи Джонсона для двух станков, полученного с помощью разработанной программы, с решением, полученным с помощью онлайн-сервиса, использующего классический алгоритм перестановок Джонсона.

2) Результат решения задачи Джонсона на разработанной программе совпал с результатом решения, полученного на онлайн-сервисе, использующего классический алгоритм Джонсона. Это подтверждает правильность программного кода алгоритма, основанного на методе ветвей и границ.

.

ЗАКЛЮЧЕНИЕ

Представленная бакалаврская работа посвящена актуальной проблеме применения метода ветвей и границ для решения конвейерной задачи теории расписаний.

В ходе выполнения бакалаврской работы достигнуты следующие результаты:

1) проанализирована научная и учебно-методическая литература по исследуемой проблеме;

2) описаны принципы метода ветвей и границ решения конвейерной задачи теории расписаний. Как показал анализ, преимуществами метода ВиГ являются обобщенность, адаптивность и применение для решения широкого круга задач дискретной оптимизации, в том числе, для решения конвейерной задачи теории расписания;

3) формализована конвейерная задача теории расписаний для решения с помощью метода ветвей и границ. Анализ известных методов формализации показал, что с помощью булевых переменных и введенных ограничений типовая конвейерная задача теории расписаний может быть сведена к задаче дискретной оптимизации;

4) произведен анализ алгоритма метода ветвей и границ для решения конвейерной задачи теории расписаний и представлен пример его применения для решения задачи Джонсона. Для реализации алгоритма разработана программа на основе специализированного онлайн-сервиса и технологии Electron. Результат решения задачи Джонсона на данной программе совпал с результатом решения с помощью классического алгоритма Джонсона, что подтверждает правильность используемого программного кода алгоритма, основанного на методе ветвей и границ.

Результаты бакалаврской работы могут быть рекомендованы для разработки математического обеспечения систем производственного календарного и сетевого планирования и управления различными проектами.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Нормативно-правовые акты

1. ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем.

Научная и методическая литература

2. Аничкин А.С. Современные модели и методы теории расписаний / А.С. Аничкин, В.А. Семенов // Труды ИСП РАН. -2014. – Т.26(3). – С. 5-50.

3. Лазарев А.С. Теория расписаний: задачи и алгоритмы / А.С. Лазарев, Е.Р. Гафаров. –М.: МГУ, 2011. – 222 с.

4. Левин В.И. Оптимальное планирование работ в конвейерных системах / В.И. Левин, И.Ю. Мирецкий // Автоматика и телемеханика. – 1996. - № 6. –С. 3-30.

5. Коффман Э.Г. Теория расписаний и вычислительные машины / Э.Г. Коффман. – М.: Наука, 1984. – 336 с.

6. Прилуцкий М.Х. Метод ветвей и границ с эвристическими оценками для конвейерной задачи теории расписаний / М.Х. Прилуцкий, В.С. Власов // Вестник Нижегородского университета им. Н.И. Лобачевского. -2008. -№ 3. - С. 147–153.

Электронные ресурсы

7. Метод ветвей и границ в применении к теории расписаний [Электронный ресурс]. - Режим доступа: https://megapredmet.ru/1-51714.html (дата обращения 04.05.2019).

8. Онлайн-калькулятор «Задача Джонсона» [Электронный ресурс]. - Режим доступа: https://math.semestr.ru/dinam/jonson.php (дата обращения 04.05.2019).

9. Струченков В. И. Дискретная оптимизация. Модели, методы, алгоритмы решения прикладных задач [Электронный ресурс] / В. И. Струченков. — М. : СОЛОН-ПРЕСС, 2016. — 192 с. — Режим доступа: http://www.iprbookshop.ru/53817.html (дата обращения 04.05.2019).

10. Технология Electron [Электронный ресурс]. - Режим доступа: https://electronjs.org (дата обращения 04.05.2019)

11. Электронный учебник «Экономико-математические методы». Метод ветвей и границ [Электронный ресурс]. - Режим доступа: http://www.math.mrsu.ru/text/courses/method/metod_vetvei_i_granic.htm (дата обращения 04.05.2019).

12. Branch and bound [Электронный ресурс]. - Режим доступа: https://en.wikipedia.org/wiki/Branch_and_bound (дата обращения 04.05.2019).

13. Branch-and-bound and Cutting Plane methods [Электронный ресурс]. - Режим доступа: http://www.cse.chalmers.se/~grohe/ashkanp/Session10.pdf (дата обращения 04.05.2019).

14. Branch-and-Bound Method [Электронный ресурс]. - Режим доступа: ocw.nctu.edu.tw/course/ip002/lecture_IP2.pdf (дата обращения 04.05.2019).

15. Discrete programming [Электронный ресурс]. - Режим доступа: https://www.encyclopediaofmath.org/index.php/Discrete_programming (дата обращения 04.05.2019).

16. LocalSolver: Flow Shop [Электронный ресурс]. - Режим доступа: https://www.localsolver.com/docs/last/exampletour/flowshop.html/ (дата обращения 04.05.2019).

17. Scheduling theory [Электронный ресурс]. - Режим доступа: https://www.encyclopediaofmath.org/index.php/Scheduling_theory (дата обращения 04.05.2019).

18. Schmeling J. Linear and Combinatorial Optimization http://www.maths.lth.se/matematiklth/personal/joerg/linopt/f7eng.pdf (дата обращения 04.05.2019).

Литература на иностранном языке

19. Bellman R. Mathematical Aspects of Scheduling Theory // J. Soc. Indust. and Appl. Math., 1956., Vol 4(3), pp. 168-205.

20. Brah S.A. and others. Mathematical Modeling of Scheduling Problems // Int. J. of Information and Management Sciences, Vol. 1, 1991.

21. Gupta D. Branch and Bound Technique for Two Stage Flow Shop Scheduling with Breakdown Interval // Int. J. of Advan. Sciennt. Research, Iss. 2, Vol. 2, 2012.

22. Johnson S. M. Optimal Two- and Three-Stage Production Schedules with Setup Times Included / / Nav. Res. Log. Quart., 1954., Vol. 1(1), pp. 61-68.

23. Land A. H. and Doig A. G. An automatic method of solving discrete programming problems, 1960, pp. 497-520.

24. Talapatra and others. Application of Branch and Bound algorithm for solving flow shop scheduling problem comparing it with tabu search algorithm // International Conference on Mechanical, Industrial and Energy Engineering 2014, Khulna.

ПРИЛОЖЕНИЕ А

Фрагмент программного кода приложения

```
body > .row > .col-sm-3 {
  opacity: 0;
  width: 23%;
}
body > .row > .col-sm-3:after {
   display: block;
   content: " ";
   width: 100%;
   height: 100%;
   background: red;
   position: absolute;
   top: 0;
   left: 0;

}
body > .row {
          margin: 0;
}
body > div:nth-child(2),
body > .row > .col-sm-6 > .uplink,
body > .row > .col-sm-6 > .btn-group,
```

```
body > .row > .col-sm-6 > .adsbygoogle,
#maintop,
body > .footer,
#MathLeftBlock,
#MathRightBlock,
body > .row > .col-sm-3 > div:nth-child(6),
#soloway240,
#MathLeftSpan,
#MathTopAdBlock,
#mainleft,
#forma1,
#forma2,
#forma3,
#disqus_thread {
  display:none!important;
}
.ref_cat h1:before{
   content: 'Программа для решения конвейерной задачи. Разработал Кашкин
P.P.';
   display: block;
   padding-bottom: 10px;
   font-size: 20px;
}
#premain {
   position: relative;
}
/*#premain a {
        display: none;
}*/
/*#premain a:last-child {
```

```
            display: block;
}*/
/*#premain a:last-child:after {
    display: block;
    content: " ";
    width: 100%;
    height: 195px;
    background: white;
    margin-top: -195px;
    z-index: 5;
    position: relative;
}*/
#balance1 {
            display: none;
}
#premain {
   position: relative;
}

#premain:after {
            display: block;
            content: " ";
            width: 100%;
            height: 195px;
            background: white;
            margin-top: -195px;
            z-index: 5;
            position: relative;
}
```

```
/*@media screen and (min-width: 1377px) {
        #premain:after {
            height: 170px;
            margin-top: -170px;
        }
}*/
```