



Московский государственный университет имени М.В. Ломоносова  
Факультет вычислительной математики и кибернетики  
Кафедра автоматизации систем вычислительных комплексов

Савицкий Илья Павлович

# Жадные алгоритмы для построения многопроцессорного списочного расписания

Дипломная работа

**Научный руководитель:**  
Доцент, к.т.н  
Костенко Валерий Алексеевич

Москва, 2022

### **Аннотация**

Построение многопроцессорного расписания это NP-трудная задача. Не существует полиномиального алгоритма. В данной работе приводятся два возможных варианта решения задачи с дополнительными ограничениями на количество передач или сбалансированность распределения работ на процессорах при помощи алгоритма, сочетающего жадные стратегии и ограниченный перебор.

# Содержание

<b>1</b>	<b>Введение</b>	<b>4</b>
<b>2</b>	<b>Цели и задачи курсовой работы</b>	<b>5</b>
<b>3</b>	<b>Постановка задачи</b>	<b>6</b>
3.1	Дано	6
3.2	Определение расписания	6
3.2.1	Способы представления расписаний	6
3.3	Требуется	7
3.4	Различные постановки задачи	8
<b>4</b>	<b>Обзор предметной области</b>	<b>9</b>
4.1	Критерии обзора	9
4.2	Конструктивные алгоритмы	9
4.2.1	Жадные алгоритмы	9
4.2.2	Структура жадных алгоритмов построения многопроцессорного расписания.	9
4.2.3	Другие конструктивные алгоритмы	10
4.3	Итерационные алгоритмы	10
<b>5</b>	<b>Алгоритм построения расписания</b>	<b>11</b>
5.1	Алгоритмы построения расписания	11
5.1.1	Общая схема жадных алгоритмов	11
5.1.2	Жадный алгоритм с жадным критерием	11
5.1.3	Жадный алгоритм с EDF эвристикой	11
5.2	Вспомогательные алгоритмы	13
5.2.1	Алгоритм локальной оптимизации разбиения	13
5.2.2	Алгоритм распределения задач на процессоры	13
5.2.3	Алгоритм постановки задачи на процессор	13
<b>6</b>	<b>Программная реализация алгоритма</b>	<b>15</b>
6.1	Описание кода программной реализации	15
6.2	Описание интерфейса программной реализации	16
6.2.1	Параметры командной строки	16
6.2.2	Описание конфигурационных файлов	16
6.2.3	Описание выходных файлов	17
<b>7</b>	<b>Экспериментальное исследование алгоритма</b>	<b>19</b>
7.1	Цели и методика экспериментального исследования	19
7.2	Экспериментальный стенд	19
7.3	Исследование качества решений	19
7.3.1	Жадный алгоритм с жадным критерием	19
7.3.2	Жадный алгоритм с EDF эвристикой	24
7.4	Исследование временной сложности алгоритма	28
7.4.1	Жадный алгоритм с жадным критерием	28
7.4.2	Жадный алгоритм с EDF эвристикой	31
<b>8</b>	<b>Заключение</b>	<b>35</b>
	<b>Приложение 1. Классы входных данных</b>	<b>36</b>

# 1 Введение

Классическая задача построения расписания хорошо изучена и досканально описана в [3]. Поскольку данная задача принадлежит к классу NP-трудных, не существует алгоритма, который за полиномиальное время даст точный ответ, но существуют алгоритмы, которые дают приближенные результаты. Большинство таких алгоритмов разделяются на две категории: *конструктивные* и *итерационные*. Из основных примеров можно выделить (большинство из них упомянуты в [14]):

- Конструктивные алгоритмы

1. Алгоритмы, основанные на поиске максимального потока в сети
2. Алгоритмы, основанные на методах динамического программирования
3. Алгоитмы, основанные на методе ветвей и границ
4. Жадные алгоритмы
5. Жадные алгоритмы с процедурой ограниченного перебора

- Итерационные алгоритмы

1. Генетические алгоритмы
2. Дифференциальная эволюция
3. Алгоритм имитации отжига
4. Алгоритм муравьиных колоний

Конструктивные алгоритмы работают, строя и дополняя частичные расписания до тех пор, пока все работы не будут размещены. Итерационные же алгоритмы строят приближения расписания и оптимизируют их.

В данной работе рассматриваются жадные алгоритмы с процедурой ограниченного перебора. Особенностью таких алгоритмов является баланс между двумя процессами построения расписания. Жадные стратегии строят расписание быстро, однако очень быстро могут зайти в тупик при построении расписания. В таком случае, если расписание строится си сильным отклонением от оптимального, процедура ограниченного перебора корректирует его.

## 2 Цели и задачи курсовой работы

Целью этой курсовой работы является разработка алгоритмов построения многопроцессорного расписания с дополнительными ограничениями на основе жадных алгоритмов.

Для достижения указанной цели требуется:

1. Провести обзор алгоритмов построения списочных расписаний с целью выявления жадных критериев и схем ограниченного перебора которые могут быть модифицированы для решения данной задачи.
2. Разработать алгоритмы.
3. Реализовать алгоритмы.
4. Провести исследование свойств алгоритма.

## 3 Постановка задачи

### 3.1 Дано

1. Ориентированный граф работ  $G$  без циклов, в котором дуги - зависимости по данным, а вершины - задания. Вершин  $n$ , дуг  $m$
2. Вычислительная система, состоящая из  $p$  различных процессоров
3. Матрица  $C_{ij}$  длительности выполнения работ на процессорах,  $i = 1 \dots n, j = 1 \dots p$ . Каждая строка этой матрицы - длины выполнения  $n$ -й задачи на  $p$  процессорах.
4. Матрица  $D_{kl}$  передач данных между процессорами,  $k = 1 \dots p, l = 1 \dots p, D_{kk} = 0$ .  $D_{ij}$ -й элемент этой матрицы - время передачи данных между процессорами  $i$  и  $j$ .

### 3.2 Определение расписания

Расписание программы определено, если заданы:

1. Множества процессоров и работ
2. Привязка - всюду определенная на множестве работ функция, которая задает распределение работ по процессорам
3. Порядок - заданные ограничения на последовательность выполнения работ и является отношением частичного порядка, удовлетворяющим условиям ацикличности и транзитивности. Отношение порядка на множестве работ, распределенных на один процессор, является отношением полного порядка.

#### 3.2.1 Способы представления расписаний

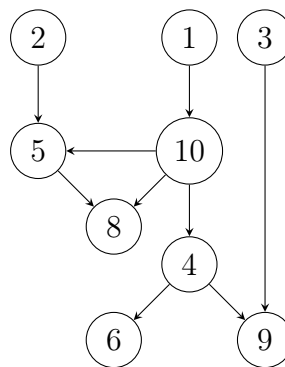


Рис. 1: Граф  $G$  потока данных

Пусть дан следующий граф потока данных, изображенный на рисунке 1.

Пусть в оптимальном расписании работы 1, 10, 4, 6 будут поставлены на  $Pr1$ . 2, 5, 8 - на  $Pr2$ , а 3 и 9 - на  $Pr3$ . Рассмотрим как такое расписание будет выглядеть в различных формах:

1. Графическая форма представления.

В такой форме представления, изображенной на рисунке 2, расписания каждой задаче сопоставляется пара из номера процессора и порядкового номера работы на процессоре, а так же секущие дуги, если задачи зависят друг от друга.

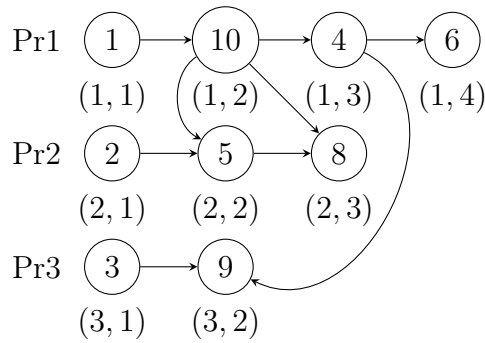


Рис. 2: Графическое представление расписания

2. Временная диаграмма.

	$t = 0$	1	2	3	4	5
Pr1	$T_1$	$T_{10}$		$T_4$	$T_6$	
Pr2		$T_2$		$T_5$	$T_8$	
Pr3	$T_3$		$\emptyset$		$T_9$	

Рис. 3: Представление расписания в виде временной диаграммы

В такой форме представления, представленной на рисунке 3, расписания каждой задаче сопоставляется пара из номера процессора и времени старта задачи на процессоре.

Доказано, что эти формы полностью эквивалентны и, имея одну, возможно построить другую. В предложенном решении расписание строится в виде временной диаграммы.

### 3.3 Требуется

1. Построить расписание  $HP$ , то есть для  $i$ -й работы определить время начала ее выполнения  $s_i$  и процессор  $p_i$  на котором она будет выполняться
2. В расписании требуется минимизировать время выполнения набора работ, данных в графе  $G$
3. В задаче так же присутствуют дополнительные ограничения, котрым расписание обязано удовлетворять.

Ограничения на корректность расписания следующие:

1. Каждый процессор может одновременно выполнять не больше одной работы;
2. Прерывание работ недопустимо, перенос частично выполненной работы на другой процессор недопустим;
3. Если между двумя работами есть зависимость на данным, то между завершением работы-отправителя и сиартом работы-получателя должен быть интервал времени, не меньший чем задержка на передачу данных между ними (с учетом привязки работ к процессорам и маршрута передачи данных).

### 3.4 Различные постановки задачи

1. Задача с однородными процессорами (длительность выполнения работы не зависит от того, на каком процессоре она выполняется) и дополнительными ограничениями на количество передач:
  - $CR = \frac{m_{ip}}{m} < 0.4$ , где  $m_{ip}$  - количество передач данных между работами на каждый процессор. (Cut ratio)
2. Задача с неоднородными процессорами, но без дополнительных ограничений на расписание



## 4 Обзор предметной области

### 4.1 Критерии обзора

Ниже приведены критерии, по которым будут рассматриваться и сравниваться алгоритмы

1. Насколько сильно рассматриваемая в статье задача отличается от решаемой. Можно ли взять алгоритм, описанный в статье за базу для алгоритма для решения данной задачи.
2. Порядок сложности алгоритма.
3. На данных какой размерности протестирован алгоритм.

### 4.2 Конструктивные алгоритмы

#### 4.2.1 Жадные алгоритмы

Жадные алгоритмы подразумевают декомпозицию задачи на ряд более простых подзадач. На каждом шаге решение принимается исходя из принципа получения оптимального решения для очередной подзадачи. То есть, на каждом шаге алгоритм делает выбор, оптимальный с точки зрения получения решения очередной подзадачи, предполагая, что эти локально-оптимальные решения приведут к приемлемому решению задачи. Какие-либо жадные стратегии, гарантированно получающие оптимальное расписание, на настоящий момент времени неизвестны, за исключением небольшого числа вариантов задач составления расписаний не принадлежащих к классу NP-полных. Например, известен жадный алгоритм, получающий точное решение для задачи обслуживания одним процессором максимального числа работ из заданного набора работ с фиксированными сроками начала и окончания [5]. Набор локальных критериев оптимизации сильно зависит от класса архитектуры. Для архитектур, в которых возможно последствие (распределяемый в расписание рабочий интервал оказывает влияние на времена инициализации ранее распределенных рабочих интервалов) возникает проблема выбора локальных критериев оптимизации, позволяющих учесть эффект последствия (на настоящий момент времени какие-либо обоснованные решения этой проблемы не известны). Кроме того, единого локального критерия (или набора и способа их использования), приводящего к наилучшему конечному результату, для решения всех подзадач не существует. Более того, при усложнении архитектуры набор и способ использования локальных критериев оказывает более сильное влияние на конечный результат. Таким образом, применение жадных алгоритмов для составления расписаний классом архитектур без последствия или даже без разделяемых ресурсов, если их влияние на значение функции построения временной диаграммы не может быть локализовано, а также проблемой выбора критериев оптимизации индивидуально для каждой подзадачи.

#### 4.2.2 Структура жадных алгоритмов построения многопроцессорного расписания.

При построении расписания жадным алгоритмом для каждой задачи необходимо определить два параметра:

1. Привязку задачи к процессору  $p_i$
2. Время старта задачи на процессоре  $s_i$

Каждый из этих параметров может быть определен своим жадным критерием. Время старта (как показано в [13]) единственным образом определяется из порядка работ на процессоре. Следовательно, имеет роль порядок, в котором работы добавляются в расписание.

Таким образом, для построения расписания достаточно определить:

1. Привязку задачи к процессору  $p_i$
2. Ее номер в очереди на добавление в расписание  $q_i$

#### 4.2.3 Другие конструктивные алгоритмы

(Написать про [1])

### 4.3 Итерационные алгоритмы

Итерационные алгоритмы работают путем создания аппроксимированного варианта решения и последующего его улучшения. Однако, большинство из них рандомизированные и, следовательно, из нескольких различных запусков теоретически возможно получить различные расписания (несмотря на то что они все сходятся). Более того, многие из таких алгоритмов плохо масштабируемы. Муравьиные алгоритмы разобраны в [15], имитации отжига - в [9].

Название алгоритма	Рандомизированность	Итерационный	Возможность масштабирования
Генетические алгоритмы	Рандомный	Итерационный	+/-
Алгоритм имитации отжига	Рандомный	Итерационный	+
Муравьиные алгоритмы	Рандомный	Итерационный	-
Жадные стратегии	Детерминированный	Конструктивный	+

Таблица 1: Существующие алгоритмы

Обзоры этих алгоритмов для схожих задач представлены в [3; 4; 12]

- Достоинства:

1. Поскольку большинство из итерационных методов - рандомизированные алгоритмы, они меньше зависят от подбора параметров.
2. Некоторые из итерационных алгоритмов, например, муравьиные алгоритмы, могут адаптироваться к изменению начальных условий, что не релевантно в рассматриваемой постановке задачи.

- Недостатки:

1. Некоторые из итерационных алгоритмов могут быть плохо масштабируемы.
2. Многие из итерационных алгоритмов требуют генерации начальных состояний расписаний, от которых они могут сильно зависеть.

## 5 Алгоритм построения расписания

### 5.1 Алгоритмы построения расписания

#### 5.1.1 Общая схема жадных алгоритмов

Жадные алгоритмы, представленные в данной работе, построены по следующей схеме:

1. Выбрать работу для постановки в расписание.
2. Выбрать процессор  $s$  и время начала выполнения задачи из п.1 на выбранном процессоре.
3. Поставить работу на процессор.
4. Остановиться, если все задачи поставлены, иначе перейти к п.1

#### 5.1.2 Жадный алгоритм с жадным критерием

Жадный алгоритм с жадным критерием следует общей схеме, описанной в 5.1.1

Эту схему можно уточнить путем выбора критерия отбора в п.1 и критерия выбора процессора и начала времени выполнения в п.2. Для постановки задачи с дополнительными ограничениями, такого как  $CR$ , так же может быть неудача в постановки задачи в расписание, в случае невозможности постановки без нарушения дополнительного ограничения. В таком случае, алгоритм завершается с неудачей.

Задача  $d_i$  называется **доступной для постановки** в расписание, в случае, если либо все ее предшественники уже установлены в расписание, либо все ее предшественники уже поставлены в расписание. Назовем множеством всех доступных для постановки задач  $D = (d_0, d_1, \dots, d_n)$ .

Жадный алгоритм с жадным критерием выбирает задачу для постановки по следующему критерию:

1. Задача  $d_i \in D$  доступна для постановки.
2. Пусть  $Succ(d)$  - функция, определяющая количество непосредственных последователей работы в графе. Тогда  $\forall d_j \in D, d_j \neq d_i : Succ(d_j) < Succ(d_i)$ .

Для постановки с дополнительным ограничением  $CR$  жадный алгоритм с жадным критерием берет распределение работ по процессорам из специального алгоритма распределения (5.2.2), поэтому выбор процессора в п.2 всегда заранее детерминирован. Выбор начала выполнения работы в расписание производится в соответствии с алгоритмом постановки задачи на процессор (5.2.3).

Для постановки без дополнительных ограничений, жадный алгоритм с жадным критерием производит пробную постановку на каждый процессор и выбирает процессор с самым ранним временем завершения работы с учетом алгоритма постановки задачи на процессор (5.2.3).

#### 5.1.3 Жадный алгоритм с EDF эвристикой

Данный алгоритм следует общей схеме, описанной в пункте 5.1.1, однако отличается от алгоритма, описанного в пункте 5.1.2 критерием выбора работы на постановку.

Эвристика “самый ранний директивный срок первый” (earliest deadline first, или **EDF**) упорядочивает работы по возрастанию директивных сроков и выбирает работу с наименьшим директивным сроком на постановку. Однако, постановка задачи не предполагает у

задач директивных сроков, поэтому в данном алгоритме у каждой работы строятся фиктивные директивные сроки.

В случае, если существует директивный срок всего расписания  $d$ , то директивный срок  $d_A$  работы  $A$  может быть рассчитан следующим образом (при известном распределении работ на процессоры):

1. Найти длиннейший путь в графе потока управления от  $A$  до работы  $S : Succ(S) = \emptyset$ .
2. Рассчитать длину этого пути. Длина пути равна сумме всех передач задержек данных и времен выполнения работ. Задержки передач данных известны, так как известно распределение работ на процессоры. Пусть длина этого пути равна  $p$ .
3.  $d_A := d - p$

Видно, что работа  $A$  должна завершиться до  $d_A$ ; иначе путь, найденный в п.1 завершится позже, чем  $d$ , даже если процессоры ни имеют никакой другой нагрузки, кроме этих работ.

Даже без известного директивного срока расписания, EDF эвристика все еще может быть использована для сортировки работ по уменьшению “потенциальной длины пути до конца расписания”, учитывая, что расписание всегда завершится какой-либо работой  $S : Succ(S) = \emptyset$ . Также, нет необходимости вводить настоящие директивные сроки, в которые работы должны быть завершены. Таким образом, можно выставить директивный срок расписания в 0, и получить формальные директивные сроки по алгоритму, представленному выше. Такие директивные сроки могут быть отрицательными, что не препятствует сортировать работы по их возрвствнию.

Описанный алгоритм без модификаций применим к задаче с дополнительным ограничением  $CR$ , поскольку распределение работ на процессоры может быть рассчитано заранее, и поэтому, время задержек межпроцессорных передач известно заранее. Для данных с однородными процессорами строится взвешенное разбиение (5.2.2), для постановки с неоднородными процессорами строится невзвешенное разбиение, которое впоследствии улучшается алгоритмом локальной оптимизации (5.2.1).

Однако, для постановки задачи без дополнительных ограничений привязка задач к процессорам заранее неизвестна, поэтому, для вычисления директивных сроков не учитываются задержки межпроцессорной передачи данных, а время выполнения данной задачи считается усредненным по всем процессорам. Например, если в системе три процессора, на которых задача выполняется 1, 1 и 4 у.е., то для расчета директивного срока время выполнения данной задачи считается  $(1 + 1 + 4)/3 = 2$ . Такая аппроксимация не нарушает работу алгоритма, поскольку директивные сроки требуется только для сортировки работ.

Жадный алгоритм с EDF эвристикой начинается с вычисления фиктивных директивных сроков, после чего выполняется цикл, описанный в 5.1.1.

Еще не добавленная работа с минимальным фиктивным директивным сроком выбирается как очередной кандидат на добавление в расписание.

Аналогично алгоритму, описанному в 5.1.2, для задачи с дополнительным ограничением  $CR$  для выбора процессора для постановки очередной задачи используется распределение, построенное алгоритмом распределения задач на процессоры (5.2.2), а для постановки без дополнительных ограничений производит пробную постановку на каждый процессор с самым ранним временем завершения работы с учетом алгоритма постановки задачи на процессор (5.2.3).

## 5.2 Вспомогательные алгоритмы

### 5.2.1 Алгоритм локальной оптимизации разбиения

Этот алгоритм используется только для постановки  $CR$  с неоднородными процессорами. Подразумевается, что процессоры упорядочены по возрастанию производительности, то есть  $\forall$  процессоров  $P_1, P_2$ , и работа  $A$  выполняется на процессоре  $P_1$  дольше, чем на  $P_2$ , то и работа  $B$  выполняется на процессоре  $P_1$  дольше, чем на  $P_2$ . Таким образом, если переназначить работу с  $P_1$  на  $P_2$ , то время выполнения расписания сократится.

С примерно равным количеством работ на всех процессорах на невзвешенном разбиении METIS, самые медленные процессору будут самыми загруженными. Задача данного алгоритма - "разгрузить" самые загруженные процессоры путем перемещения работ с него на менее загруженные процессоры.

Алгоритм имеет следующую структуру:

1. Выбрать самый загруженный процессор  $P_1$
2. Для каждой работы  $A$ , поставленной на  $P_1$ , в порядке убывания времени выполнения:
  - (a) Выбрать самый быстрый процессор  $P_2$  из процессоров, удовлетворяющих следующему условию: если перенести работу  $A$  с  $P_1$  на  $P_2$ , то  $\max(\text{загрузка } P_1, \text{загрузка } P_2)$  уменьшается и выполняется ограничение  $CR$
  - (b) Если такой  $P_2$  был найден, то перенести эту задачу и перейти к пункту 2a; иначе рассмотреть следующую по времени выполнения задачу на  $P_1$ .
3. Если задачи на  $P_1$  кончились, то остановить алгоритм.

### 5.2.2 Алгоритм распределения задач на процессоры

В качестве алгоритма распределения задач на процессоры был выбран алгоритм разбиения графа на кластеры METIS [7].

Для построения распределения работ на процессоры запускается алгоритм кластеризации графа с количеством кластеров, равным количеству процессоров, после чего каждый кластер распределенных задач присваивается одному процессору.

Для задачи с дополнительным ограничением  $CR$  используется взвешенное распределение METIS, где каждой вершине придается вес, равный времени выполнения задачи на процессоре. Поскольку в этой постановке процессоры равны, конкретный процессор в которого берется время выполнения не имеет значения.

Разбиение, лучшее по балансу кластеров, может нарушать ограничение  $CR$ , в случае, если большие группы плотно взаимодействующих работ распределятся на разные процессоры. Эта проблема решается варьированием параметра `ufactor` алгоритма METIS, который контролирует отношение максимального количества работ в подграфе к вредному количеству работ в подграфе. Другими словами, `ufactor` позволяет контролировать дисбаланс в количестве вершин в кластере. С увеличением этого параметра,  $CR$  понижается. Для генерации распределения, удовлетворяющего дополнительному ограничению  $CR$ , достаточно генерировать распределения с постепенным увеличением `ufactor` до тех пор, пока очередное распределение не выполнит  $CR$ .

### 5.2.3 Алгоритм постановки задачи на процессор

При постановке задачи на заданный процессор достаточно вычислить время начала выполнения задачи  $t$  такое, чтобы каждое частичное расписание после добавления оставалось

корректным. Начальное время  $t$  выбирается как минимальное время, удовлетворяющее следующим условиям:

1. Все передачи данных от предшествующих задач завершились до  $t$ .
2. Существует свободный интервал времени (простой процессора или после завершения последней поставленной задачи), начинающийся в  $t$  и длительностью, больший или равный времени выполнения работы, в который не выполняется ни одна работа. В некоторых случаях, задача будет поставлена до начала другой, не связанной с ней задачей, в случае, если времени простоя достаточно. В сложных графах потока управления, такие простои возникают в частичных расписаниях часто, а значит всегда есть смысл их заполнять.

## 6 Программная реализация алгоритма

### 6.1 Описание кода программной реализации

Код реализации выложен на C++ в репозитории [10]. Диаграмма калссов реализации представлена на рисунке 4.

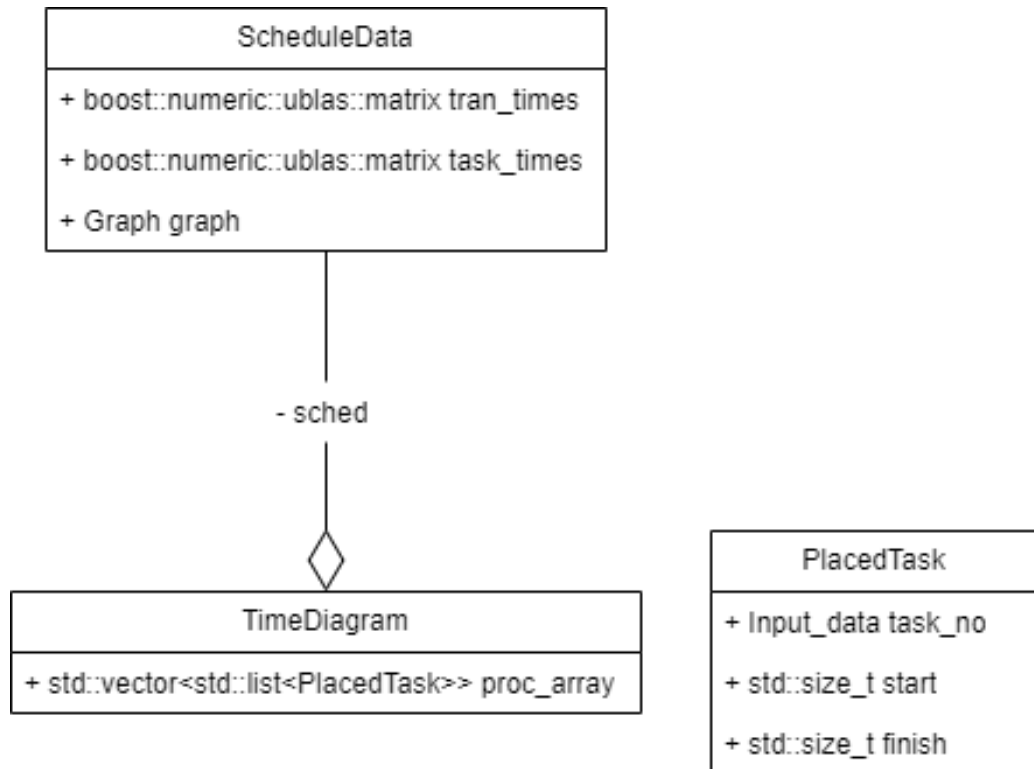


Рис. 4: UML-диаграмма реализации

Среди представленных классов:

- **ScheduleData** - класс, хранящий в себе входные данные и выполняющий всю необходимую их предобработку.
- **TimeDiagram** - класс, хранящий в себе частичное или полное расписание.
- **PlacedTask** - класс, хранящий в себе информацию о поставленной в расписание работе.

Жадные алгоритмы реализованы в функциях, не инкапсулированных в классах:

- `construct_time_schedule()` - жадный алгоритм с жадным критерием.
- `greedy_EDF_heuristic()` - Жадный алгоритм с EDF эвристикой.

В репозиторий включены следующие библиотеки:

1. METIS 5.1.0 [8] - библиотека для разбиения графов.
2. json 3.11.2 [6] - библиотека для работы с форматом JSON. Используется для составления выходных файлов.
3. toml11 3.7.1 [11] - библиотека для работы с форматом TOML. Используется для чтения конфигурационных файлов.

Также, у реализации есть зависимость, не включенная в репозиторий - `boost 1.80` [2]. Для сборки проекта используется `CMake`. Инструкция по сборке приведена в листинге 1. Для сборки документации (на английском) используется `Doxygen`. Инструкция по сборке документации приведена в листинге 2

```
mkdir build
cd build
cmake ..
make
```

Листинг 1: Сборка программной реализации

```
doxygen Doxyfile
```

Листинг 2: Сборка документации

## 6.2 Описание интерфейса программной реализации

### 6.2.1 Параметры командной строки

Из исходного кода реализации алгоритма собирается утилита, с интерфейсом, описанным в листинге 3 и таблице 2.

```
opts <algorithm_name> --input <input_file> --output <output_file>
↪ --conf <config_file> --log <log_level>
```

Листинг 3: Шаблон запуска утилиты построения расписания

Имя	Описание
<code>algorithm_name</code>	Название алгоритма для построения расписания
<code>input</code>	Путь к файлу с входными данными
<code>output</code>	Путь к файлу с выходными данными
<code>conf</code>	Путь к файлу с конфигурацией
<code>log</code>	Уровень логирования

Таблица 2: Параметры командной строки программы

### 6.2.2 Описание конфигурационных файлов

В качестве формата конфигурационных файлов был выбран формат `toml`. Пример конфигурационного файла приведен в листинге 4 и таблице 3. Конфигурационный файл содержит два раздела:

- Раздел `[general]`, отвечающий за общие параметры построения расписания.
- Раздел `[greedy]`, отвечающий за параметры, относящиеся только к жадному алгоритму.



Поле	Описание
<code>criteria</code>	Критерий, дополнительное ограничение которого будет выполняться (CR / NO)
<code>CR_bound</code>	Верхняя граница ограничения $CR$ (если используется)
<code>inp_class</code>	Класс типа входных данных <ul style="list-style-type: none"> <li>• <code>class_1</code> для постановки с однородными процессорами</li> <li>• <code>class_general</code> для постановки с неоднородными процессорами</li> </ul>
<code>cr_con</code>	Переключение жадного критерия в жадном алгоритме с жадными критериями с максимального количества потомков на максимальное количество предков.

Таблица 3: Параметры конфигурационного файла.

```

1  [general]
2  criteria = "BF"
3  CR_bound = 0.4
4  inp_class = "class_1"
5
6  [greedy]
7  cr_con = false

```

Листинг 4: Пример конфигурационного файла

### 6.2.3 Описание выходных файлов

В качестве формата выходных файлов был выбран формат `json`. Пример конфигурационного файла приведен в таблице 4. Конфигурационный файл содержит информацию о характеристиках построенного расписания, а так же информацию о привязках и порядке постановке работ на процессорах.

Поле	Описание
<code>CR</code>	Значение ограничения $CR$ построенного расписания.
<code>algo_time</code>	Время выполнения алгоритма, в миллисекундах
<code>criteria</code>	Дополнительное ограничение, используемое для построения расписания
<code>nodes</code>	Количество работ во входном графе.
<code>time</code>	Время выполнения построенного расписания
<code>procs</code>	Словарь с номерами процессоров в качестве ключей и массивами поставленных на соответствующий процессор работами. Каждая поставленная работа состоит из: <ul style="list-style-type: none"> <li>• <code>task_dur</code> - время выполнения работы на поставленный процессор.</li> <li>• <code>task_no</code> - идентификатор работы.</li> <li>• <code>task_start</code> - время начала выполнения работы на процессоре.</li> </ul>

Таблица 4: Поля выходного файла

```

1      {
2          "CR": 0.3221312,
3          "algo_time": 300,
4          "criteria": "CR",
5          "nodes": 2000,
6          "procs": {
7              "0": [
8                  {
9                      "task_dur": 5,
10                     "task_no": 1202,
11                     "task_start": 0
12                 },
13                 {
14                     "task_dur": 3,
15                     "task_no": 1608,
16                     "task_start": 5
17                 },
18                 ...
19             ],
20             "1": [
21                 ...
22             ],
23             ...
24         },
25         "time": 2211
26     }

```

Листинг 5: Пример выходного файла

## 7 Экспериментальное исследование алгоритма

### 7.1 Цели и методика экспериментального исследования

Целями экспериментального исследования было поставлено исследование:

- Качество решений, предоставляемых алгоритмами.
- Временная сложность алгоритма.

Для проведения экспериментов было сгенерировано 3 набора данных:

1. Набор данных с известных оптимумом, для постановки задачи с дополнительным ограничением  $CR$
2. Набор данных, основанных на слоистых графах, без известного оптимума, но с однородными процессорами. Данные из пункта 1 имеют свойство идеально сбалансированного разбиения, то есть разбиение от METIS всегда построит разбиение, близкое к разбиению идеального расписания. Чтобы проверить, как ведут себя алгоритмы на данных без такого свойства, были добавлено исследование на данных слоистых графов. Данный набор так же используется для исследований алгоритма для постановки задачи с дополнительным ограничением  $CR$ .
3. Набор данных, основанный на слоистых графах, без известного оптимума, но с неоднородными процессорами. Используется для постановки задачи без дополнительных ограничений

Схема генерации слоистых графов описана в [Canon\_2019].

### 7.2 Экспериментальный стенд

Эксперименты были проведены на машине, обладающей следующими характеристиками:

- CPU Intel Xeon E5-2605 v4, 2.2ГГц
- 62Гб оперативной памяти

### 7.3 Исследование качества решений

#### 7.3.1 Жадный алгоритм с жадным критерием

##### 7.3.1.1 Постановка CR

На рисунках 5 и 6 показано качество решений, генерируемых жадных алгоритмом с жадными критерием на данных с известным оптимумом. Цветом на рисунке 5 и значением на оси  $Oy$  на рисунке 6 показано отношение длительности расписания, построенного алгоритмом к длительности оптимального расписания. Значения всегда больше 1, чем меньше, тем лучше.

Точность алгоритма повышается с увеличением количества процессоров и уменьшается с повышением количества процессоров в системе.

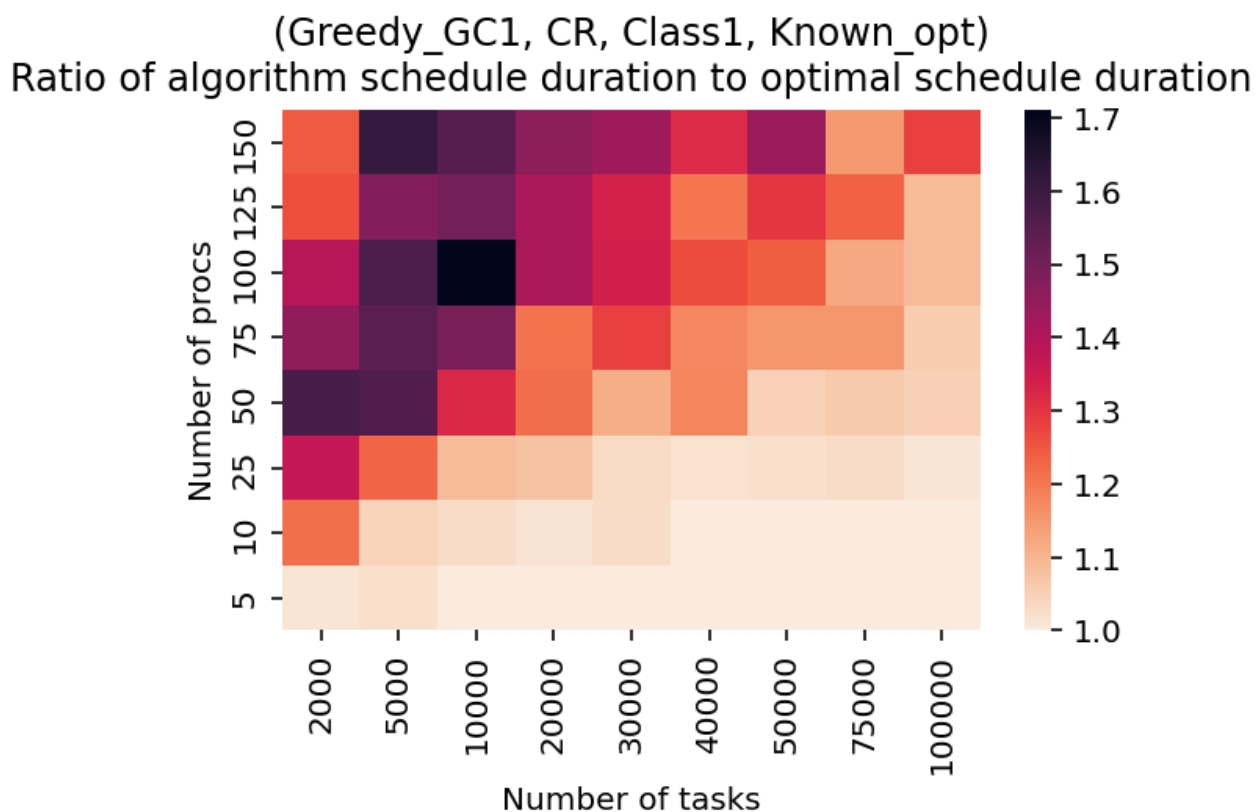


Рис. 5: Отношение времени выполнения расписания к оптимальному времени выполнения (тепловая карта)

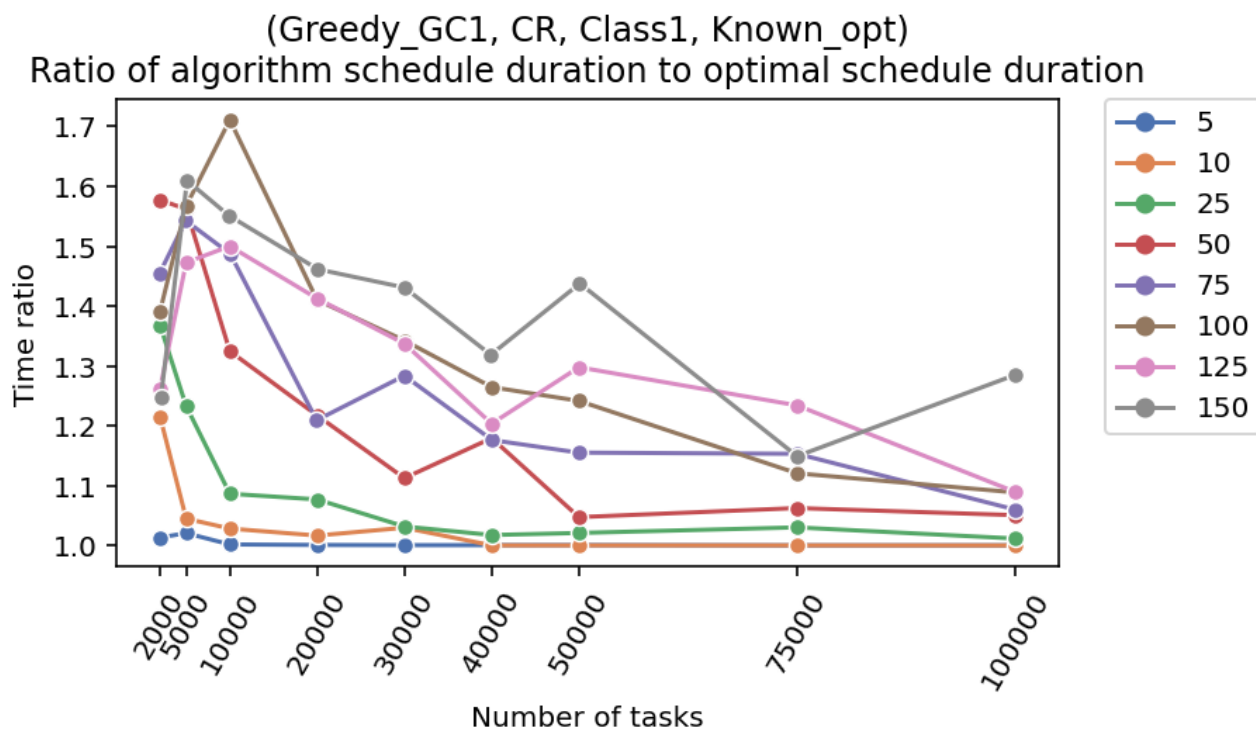


Рис. 6: Отношение времени выполнения расписания к оптимальному времени выполнения

(Greedy\_GC1, CR, Class1, Known\_opt) CR values

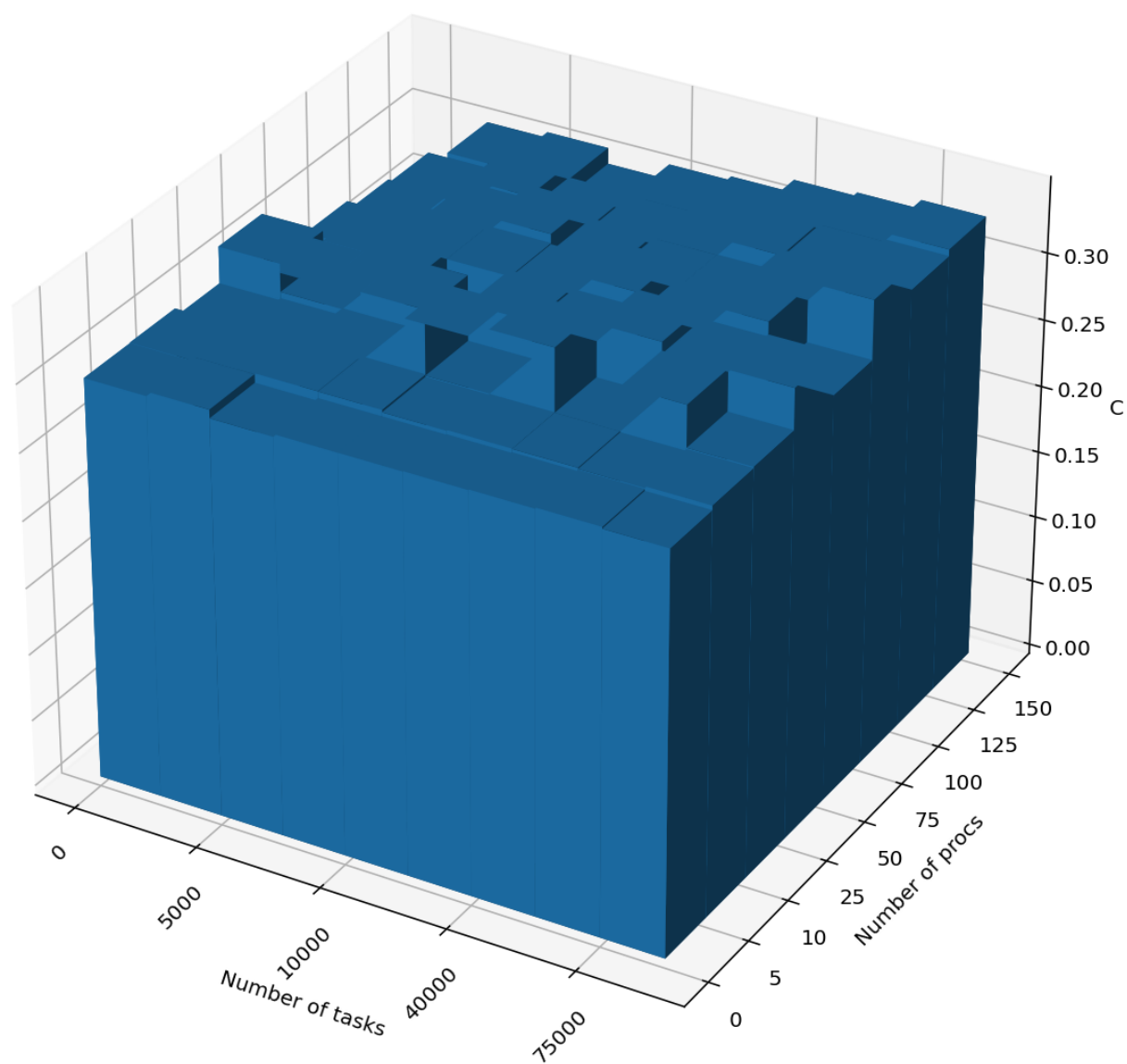


Рис. 7: 12345

(Greedy\_GC1, CR, Class1, Layered) CR values

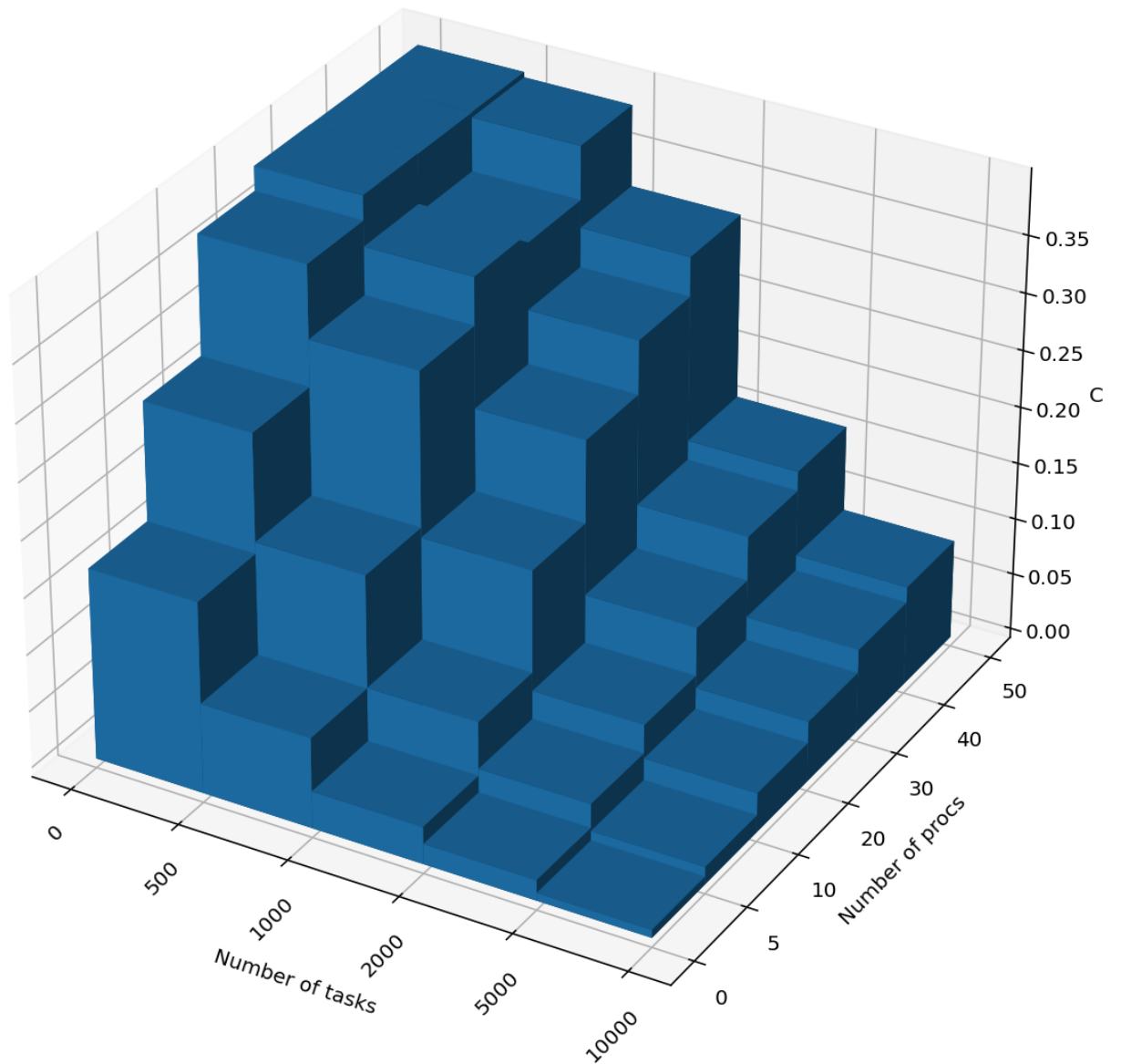


Рис. 8: 12345

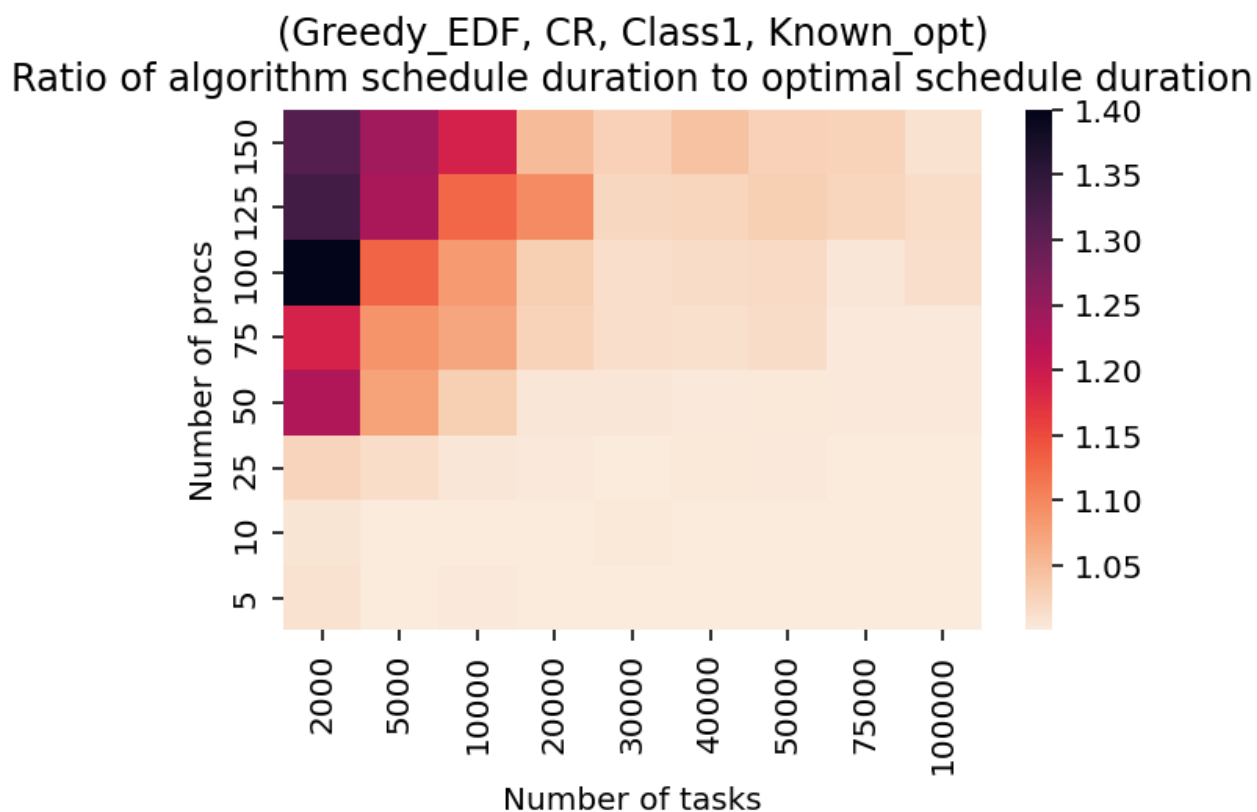


Рис. 9: Отношение времени выполнения расписания к оптимальному времени выполнения (тепловая карта)

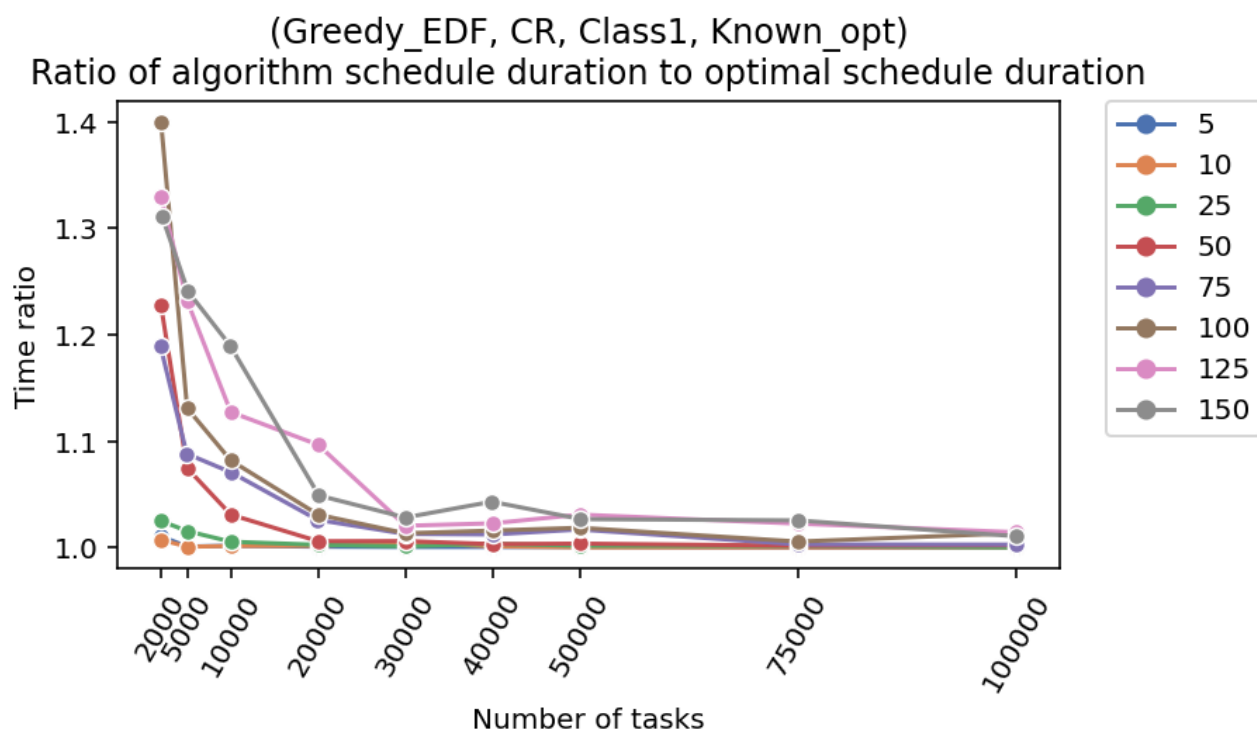


Рис. 10: Отношение времени выполнения расписания к оптимальному времени выполнения

### 7.3.2 Жадный алгоритм с EDF эвристикой

#### 7.3.2.1 Постановка CR

На рисунках 9 и 10 показано качество решений, генерируемых жадным алгоритмом с жадными критерием на данных с известным оптимумом. Цветом на рисунке 9 и значением на оси  $Oy$  на рисунке 10 показано отношение длительности расписания, построенного алгоритмом к длительности оптимального расписания. Значения всегда больше 1, чем меньше, тем лучше.

Точность алгоритма повышается с увеличением количества работ, однако ухудшение решения с увеличением количества процессоров в системе менее значительно, чем в жадном алгоритме с жадным критерием.

(Greedy\_EDF, CR, Class1, Known\_opt) CR values

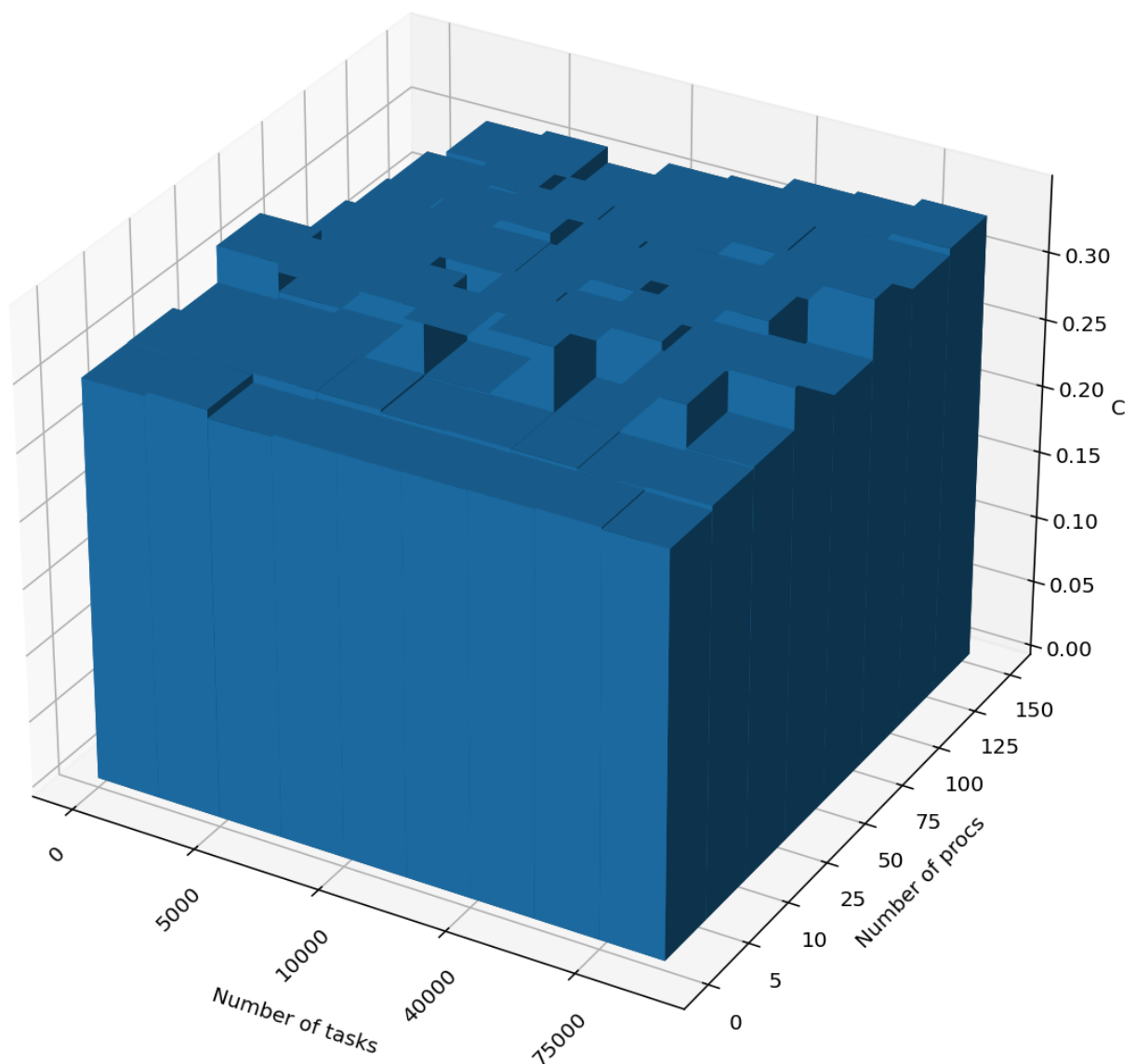


Рис. 11: 12345

На рисунках 12 и 13 показано качество решений, генерируемых жадным алгоритмом с EDF эвристикой на данных, построенных на слоистых графах. Цветом на рисунке 12 и



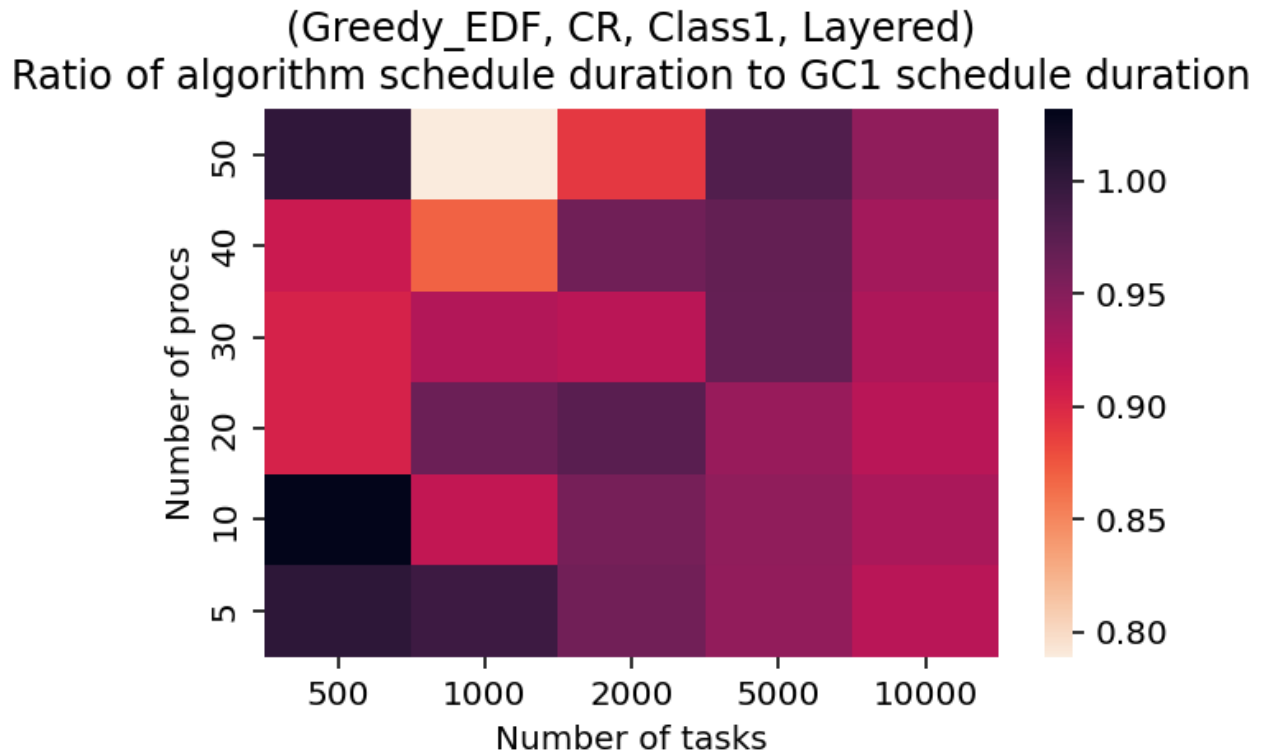


Рис. 12: Отношение времени выполнения расписания к времени выполнения расписания, построенного при помощи жадного алгоритма с жадным критерием (тепловая карта)

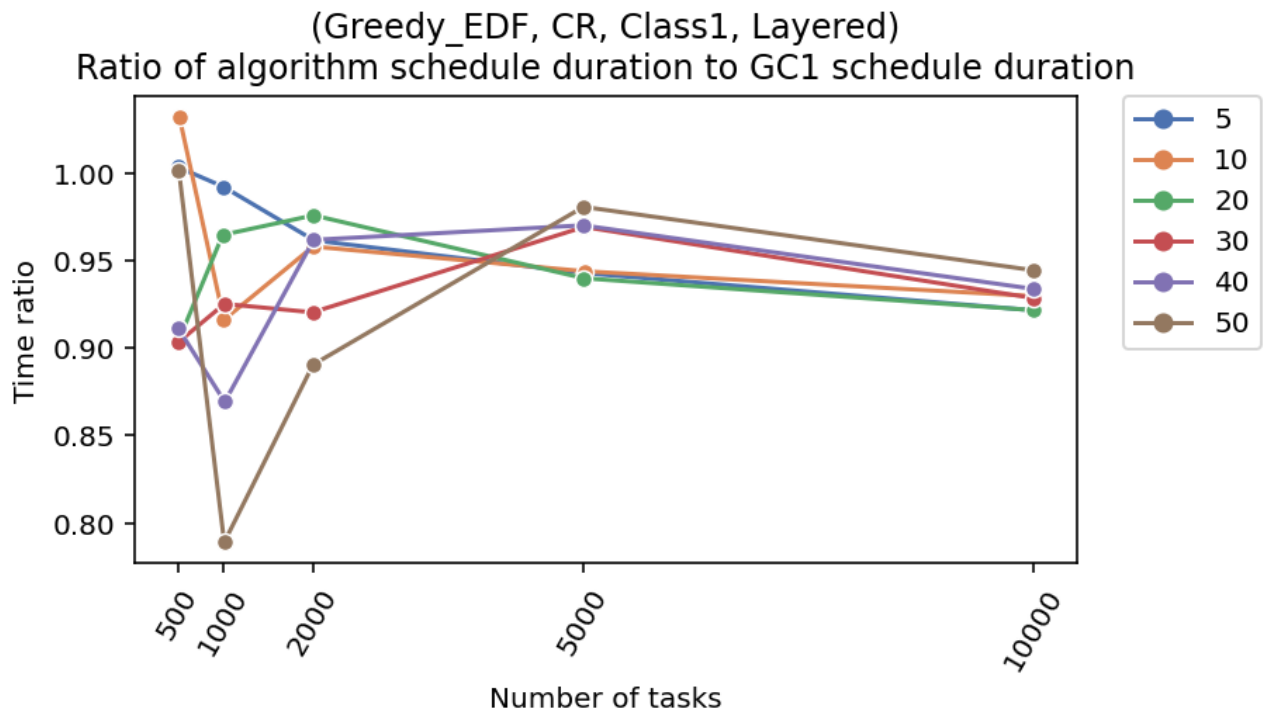


Рис. 13: Отношение времени выполнения расписания к времени выполнения расписания, построенного при помощи жадного алгоритма с жадным критерием

значением на оси  $Oy$  на рисунке 13 показано отношение длительности расписания расписания, построенного жадным алгоритмом с жадным критерием. Значения всегда больше 1, чем меньше, тем лучше.

В большинстве случаев, качество решений жадного алгоритма с EDF эвристикой лучше качества решений жадного алгоритма с жадным критерием, однако преимущество остается в пределах 10%, кроме двух выбросов в районе 1000 работ.

(Greedy\_EDF, CR, Class1, Layered) CR values

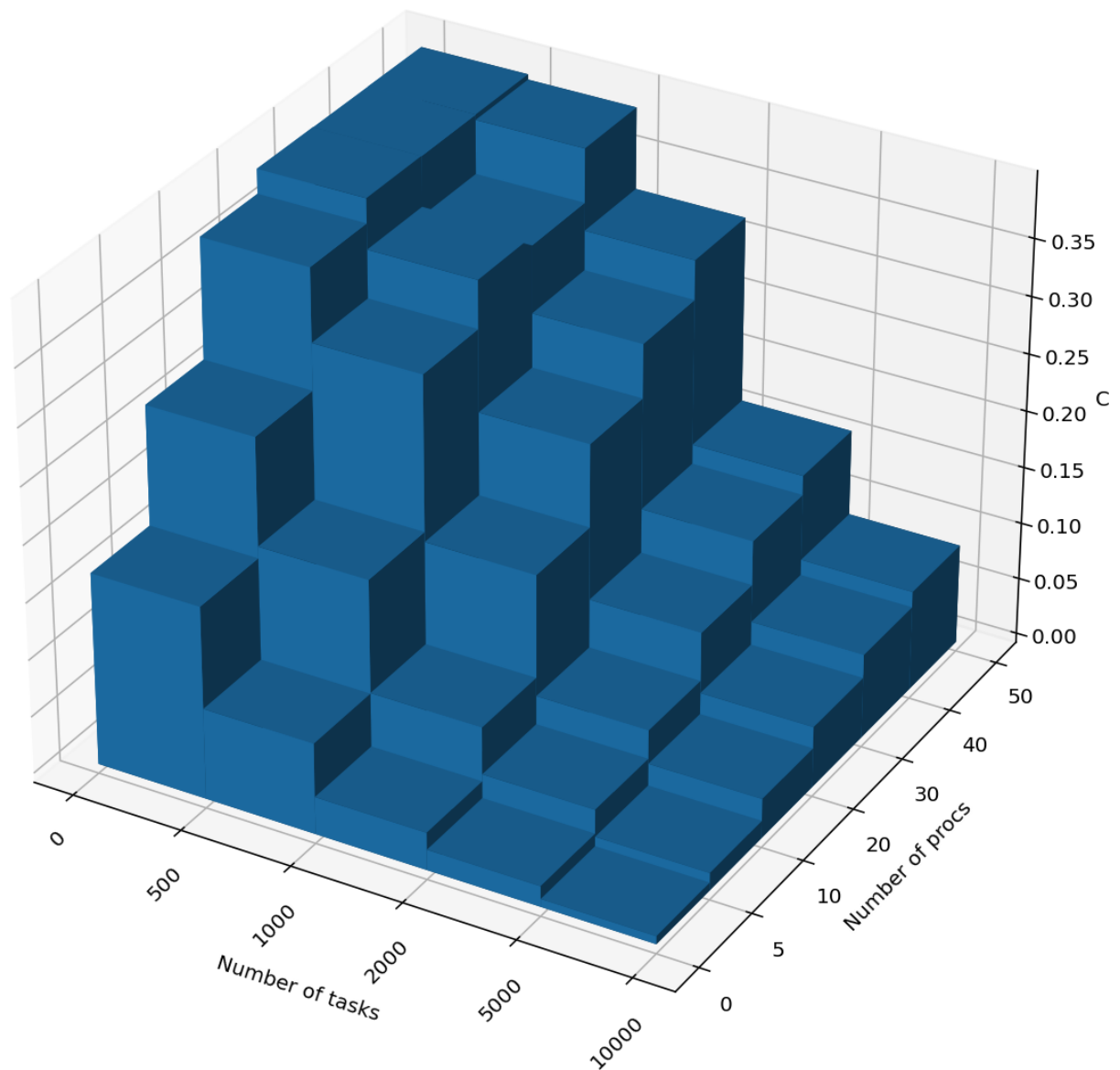


Рис. 14: 12345

### 7.3.2.2 Постановка NO

На рисунках 15 и 16 показано качество решений, генерируемых жадным алгоритмом с EDF эвристикой на данных, построенных на слоистых графах. Цветом на рисунке 15 и значением на оси  $Oy$  на рисунке 16 показано отношение длительности расписания расписания, построенного жадным алгоритмом с жадным критерием. Значения всегда больше 1,

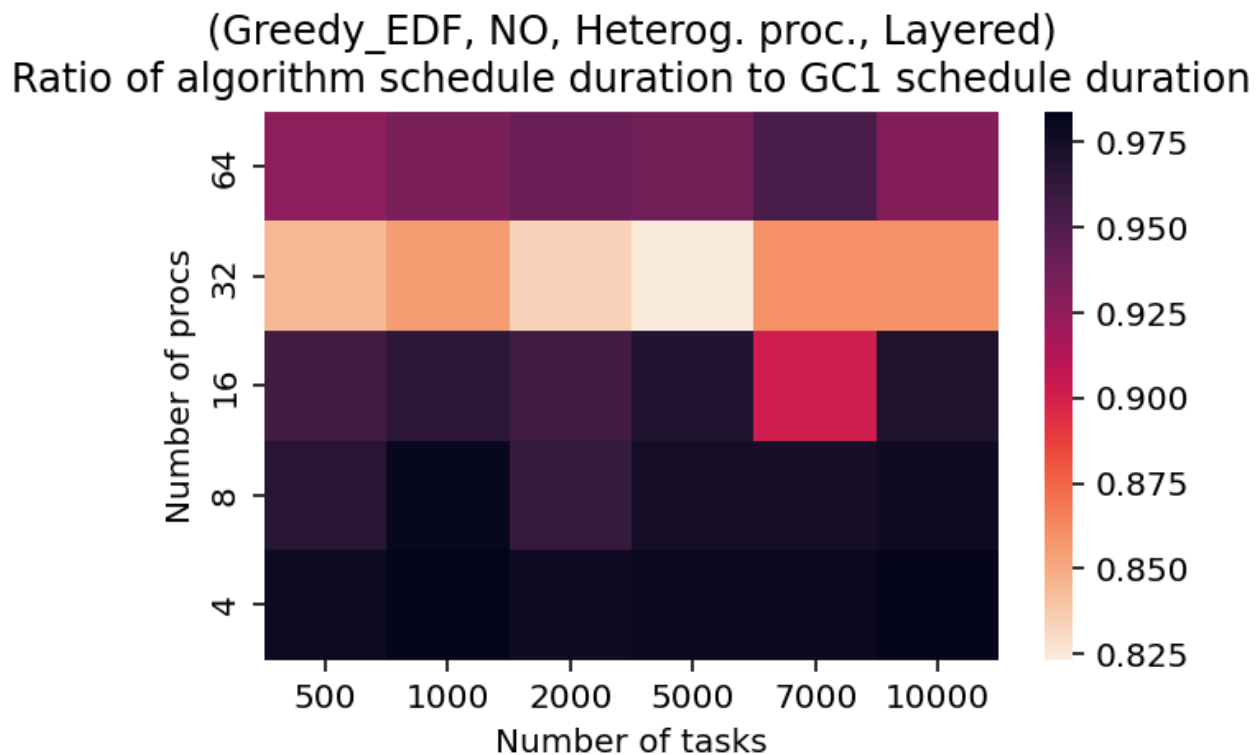


Рис. 15: Отношение времени выполнения расписания к времени выполнения расписания, построенного при помощи жадного алгоритма с жадным критерием (тепловая карта)

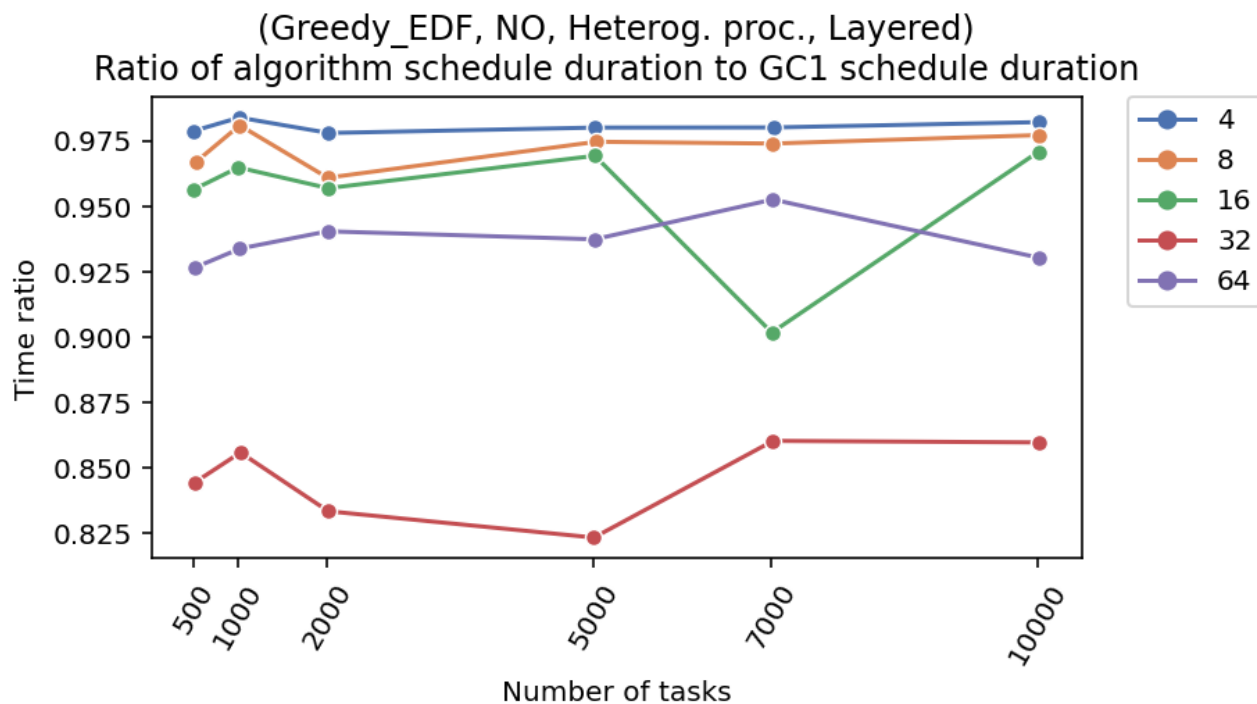


Рис. 16: Отношение времени выполнения расписания к времени выполнения расписания, построенного при помощи жадного алгоритма с жадным критерием

чем меньше, тем лучше.

Во всех случаях, качество решений выше качества решений жадного алгоритма с жадным критерием, но это улучшение не превышает 10%, за исключением случая с 32 процессорами.

## 7.4 Исследование временной сложности алгоритма

### 7.4.1 Жадный алгоритм с жадным критерием

#### 7.4.1.1 Постановка CR

(Greedy\_GC1, CR, Class1, Known\_opt) Algorithm execution time, s

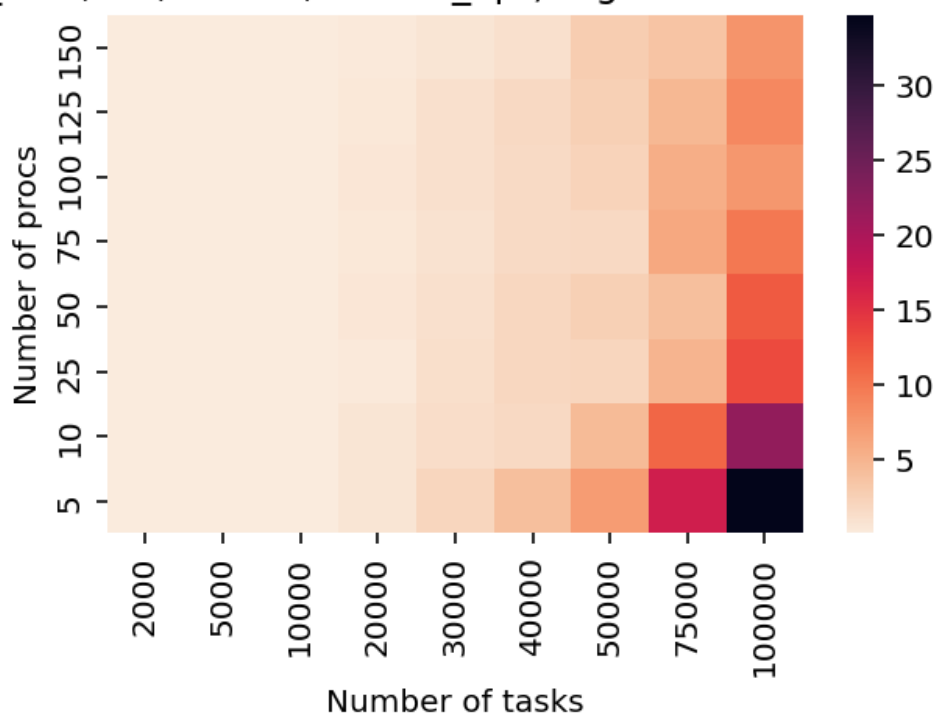


Рис. 17: Время выполнения алгоритма, в секундах (тепловая карта)

На рисунках 17 и 18 показано время выполнения жадного алгоритма с жадным критерием, включая прогоны METIS. Время выполнения растет с увеличением количества вершин. При равном количестве работ, выше время выполнения при меньшем количестве процессоров. Причина в том, что при равном количестве работ и понижении количества процессоров повышается количество работ, распределенных на процессор, что приводит к большему количеству пропусков в расписании, что значит, что алгоритм постановки работы в расписании отработает быстрее, т.к. он работает до первого найденного доступного простоя на процессоре.

На рисунках 19 и 20 показано время выполнения алгоритма на наборе данных, основанных на слоистых графах. Кроме двух выбросов, соотносящихся с самым малым количеством работ и самым большим количеством процессоров в системе, нет существенной зависимости времени выполнения от количества процессоров в системе.

#### 7.4.1.2 Постановка NO

На рисунках 21 и 22 показано время выполнения алгоритма на наборе данных с неоднородными процессорами. Время выполнения увеличивается с увеличением количества

(Greedy\_GC1, CR, Class1, Known\_opt) Algorithm execution time, s

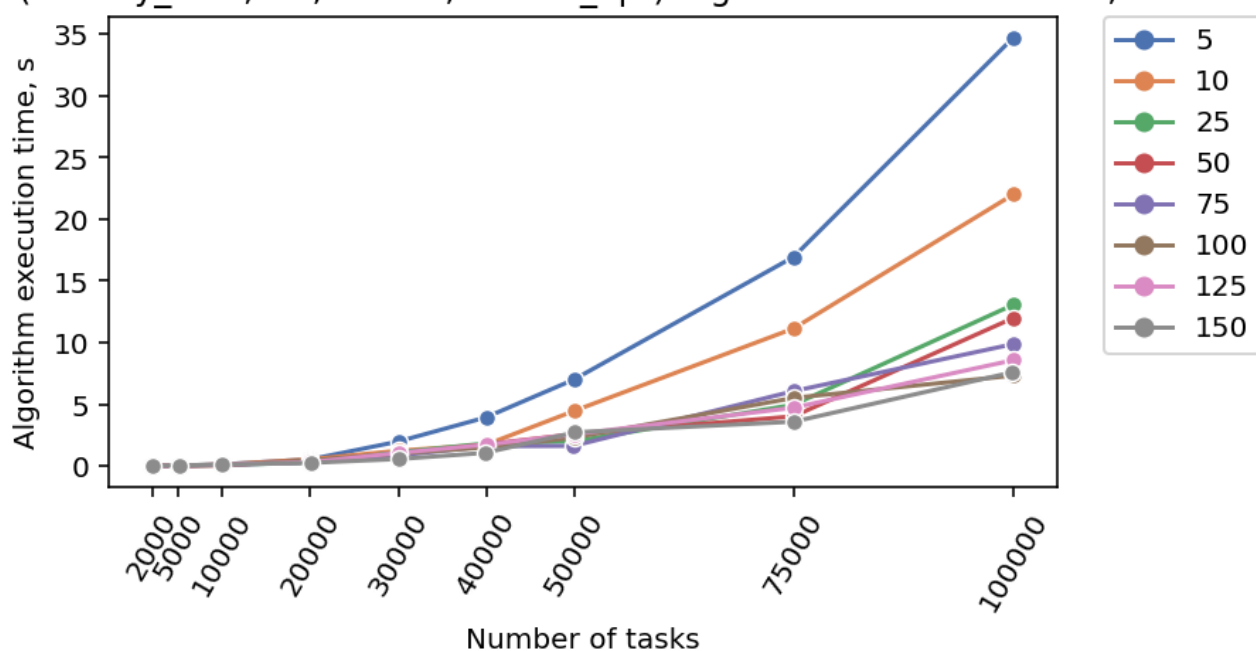


Рис. 18: Время выполнения алгоритма, в секундах

(Greedy\_GC1, CR, Class1, Layered) Algorithm execution time, ms

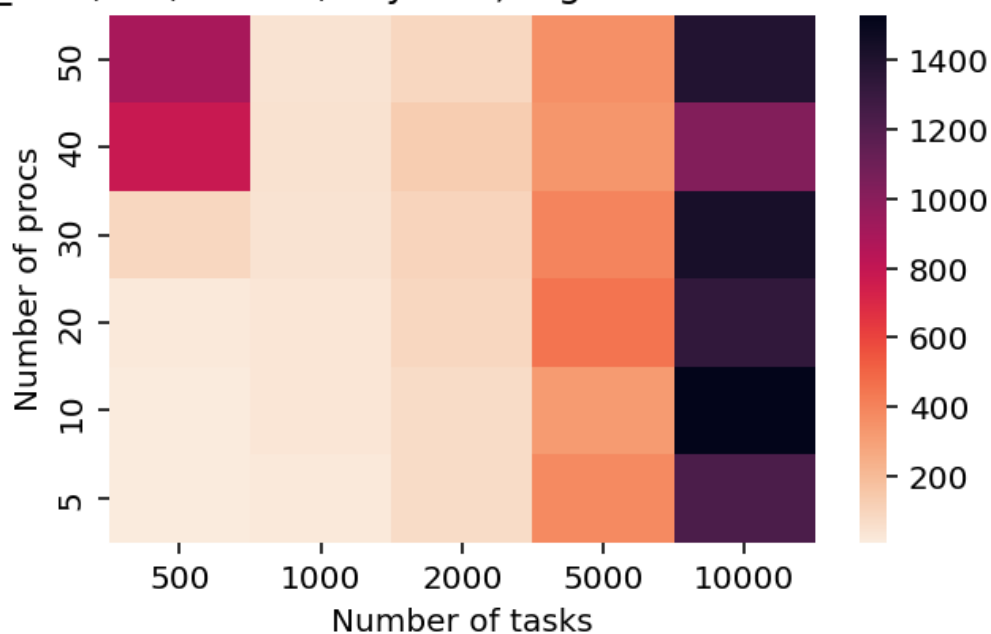


Рис. 19: Время выполнения алгоритма, в миллисекундах (тепловая карта)

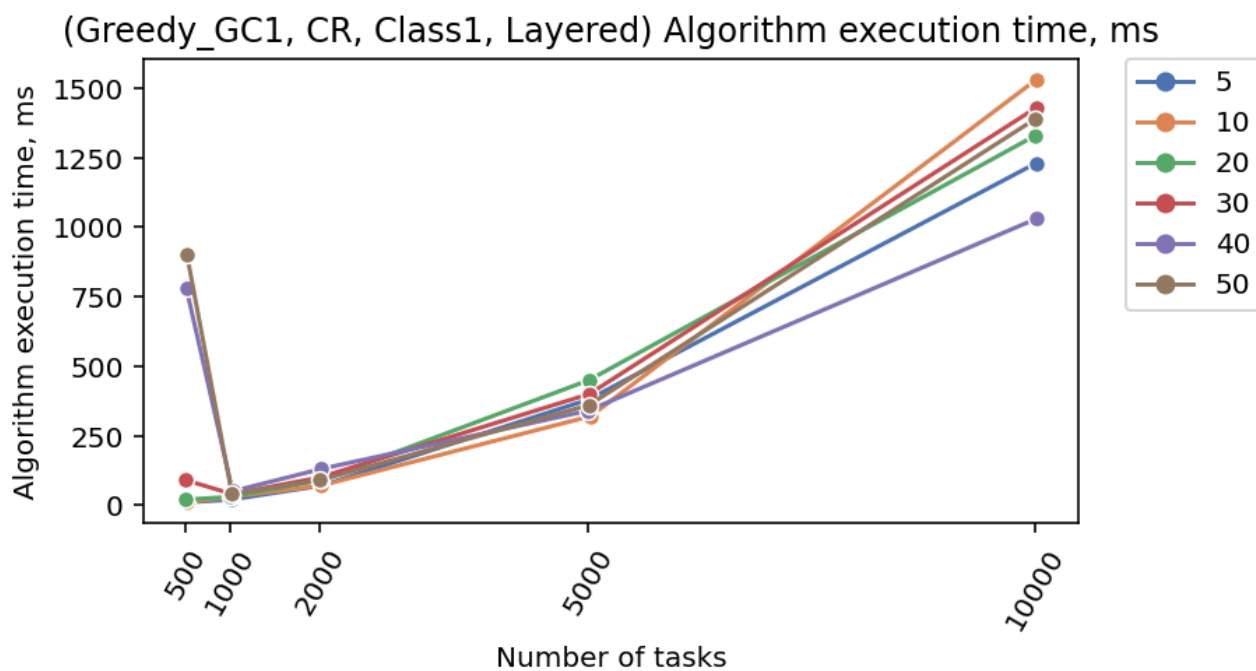


Рис. 20: Время выполнения алгоритма, в миллисекундах

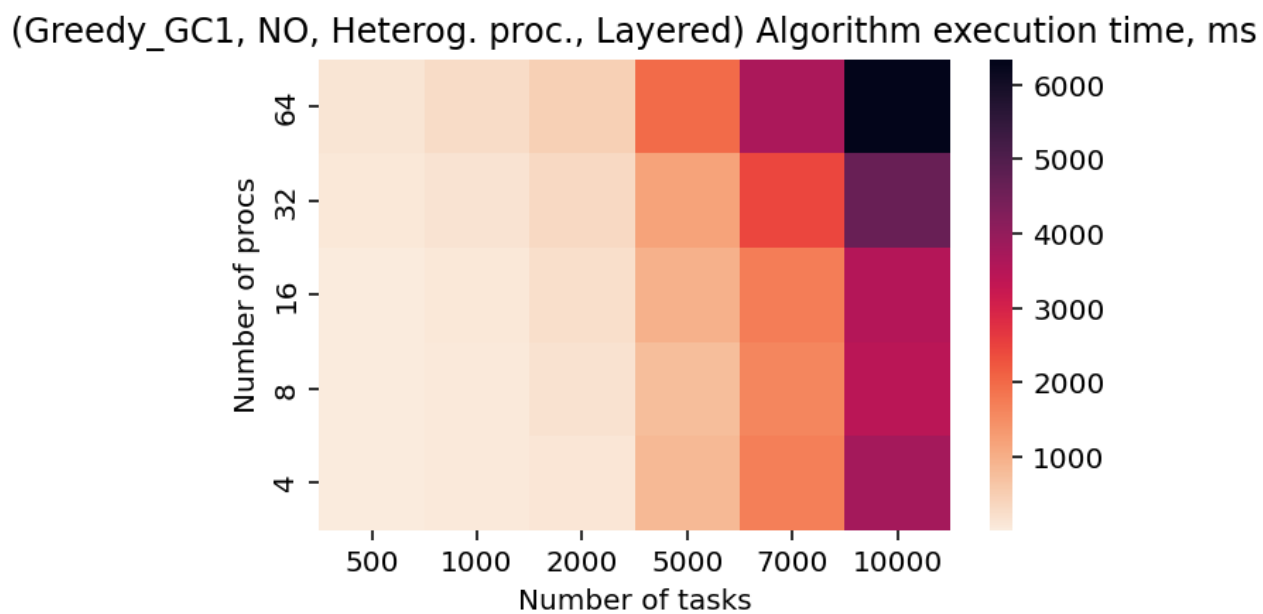


Рис. 21: Время выполнения алгоритма, в миллисекундах (тепловая карта)

(Greedy\_GC1, NO, Heterog. proc., Layered) Algorithm execution time, ms

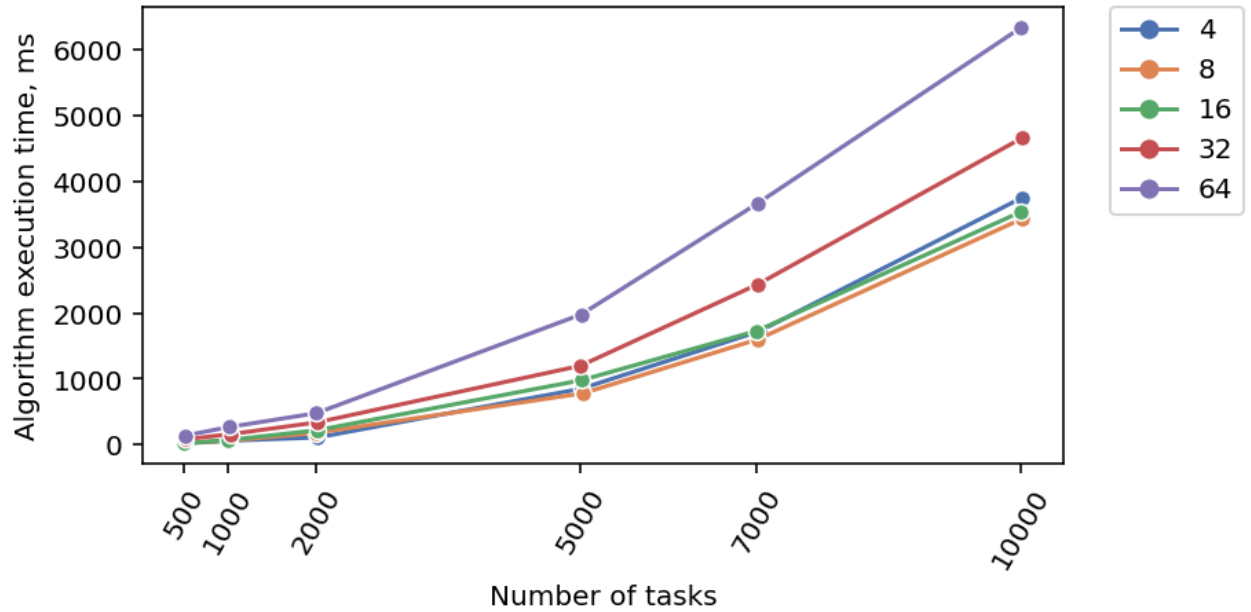


Рис. 22: Время выполнения алгоритма, в миллисекундах

задач и с увеличением количества процессоров.

#### 7.4.2 Жадный алгоритм с EDF эвристикой

##### 7.4.2.1 Постановка CR

(Greedy\_EDF, CR, Class1, Known\_opt) Algorithm execution time, s

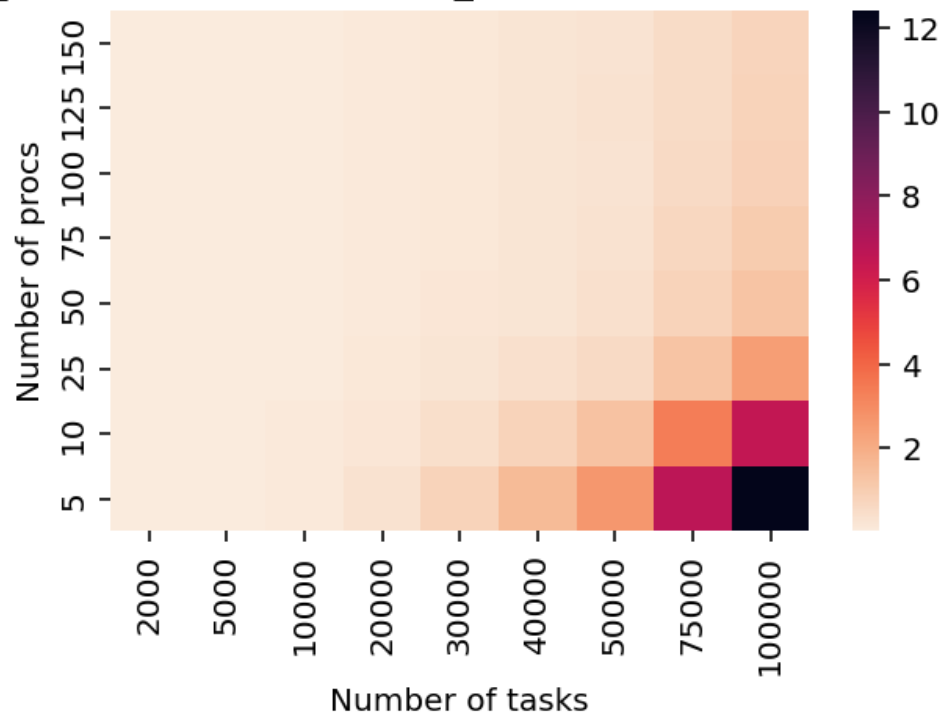


Рис. 23: Время выполнения алгоритма, в секундах (тепловая карта)

(Greedy\_EDF, CR, Class1, Known\_opt) Algorithm execution time, s

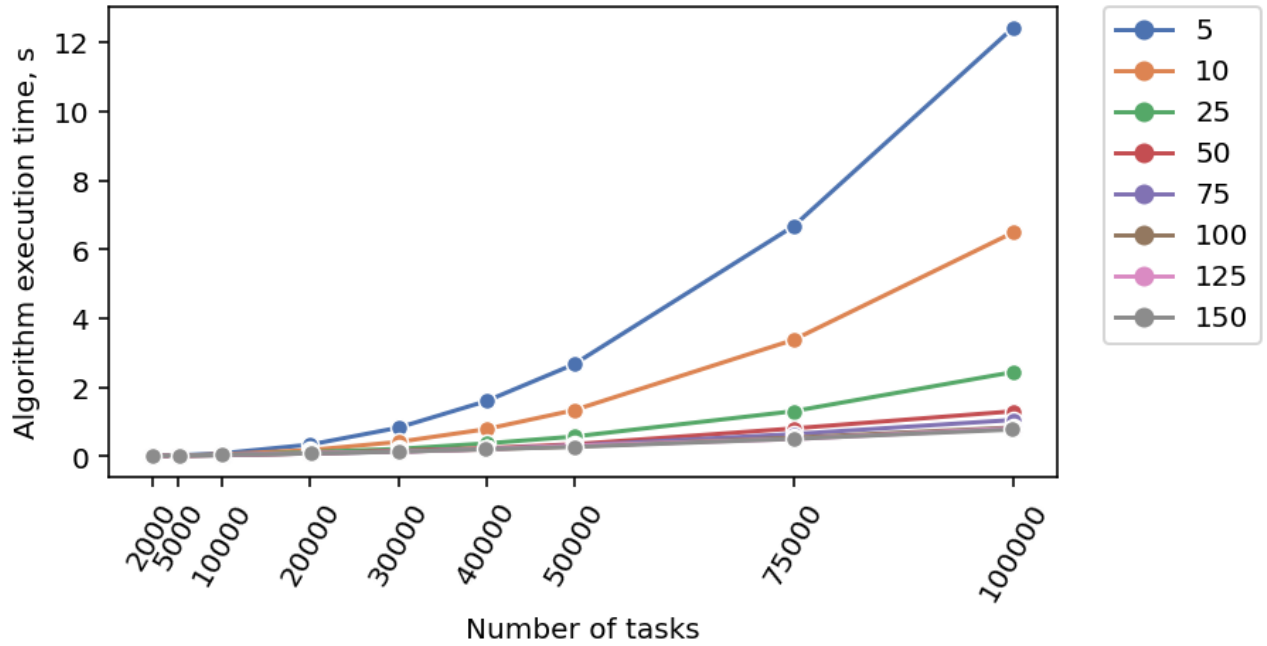


Рис. 24: Время выполнения алгоритма, в секундах

На рисунках 23 и 24 показано время выполнения жадного алгоритма с EDF эвристикой, включая прогоны METIS. Время выполнения растет с увеличением количества вершин, при этом в несколько раз меньшим времени, затраченного на прогон жадного алгоритма с жадным критерием. При равном количестве работ, выше время выполнения при меньшем количестве процессоров. Причина схожа с причинами подобного явления для жадного алгоритма с жадным критерием, поскольку они разделяют одну процедуру поиска нового места в расписании.

(Greedy\_EDF, CR, Class1, Layered) Algorithm execution time, ms

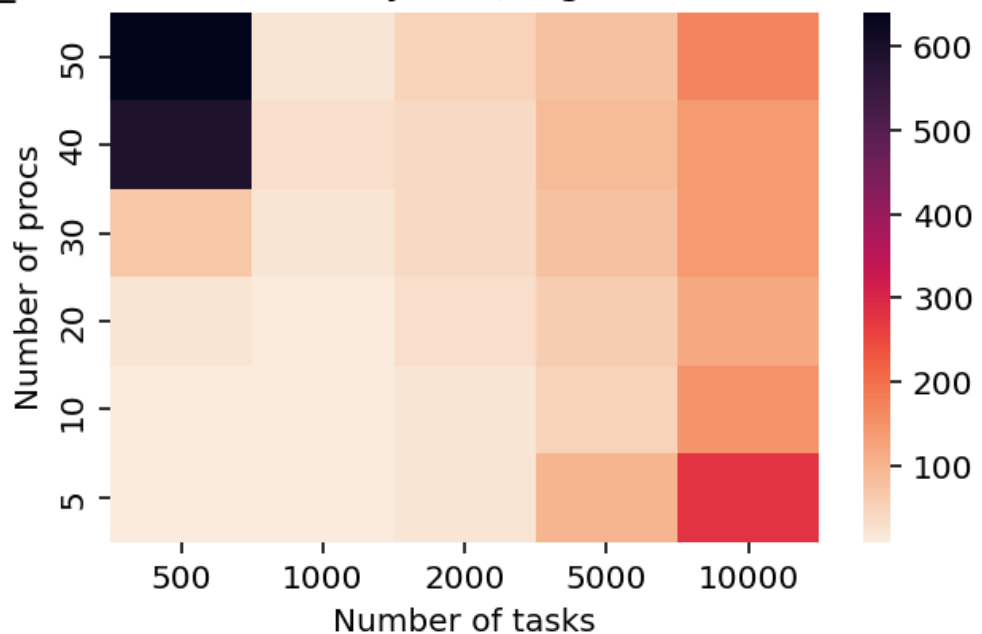


Рис. 25: Время выполнения алгоритма, в миллисекундах (тепловая карта)



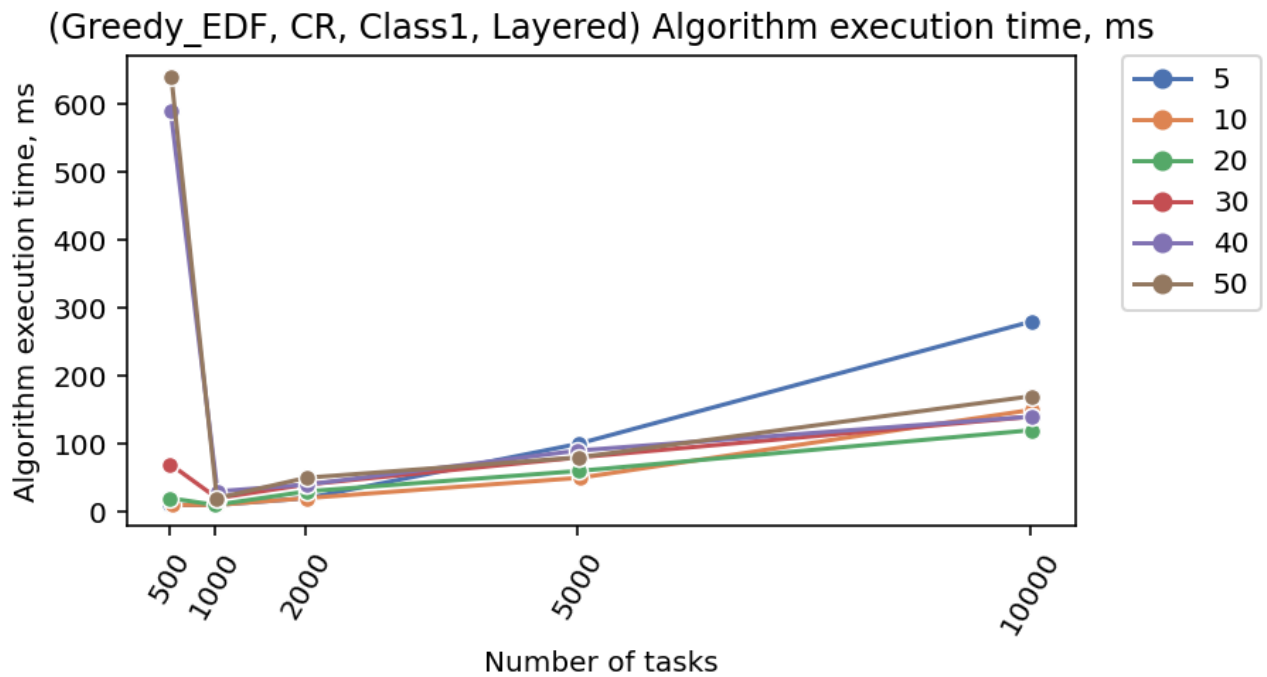


Рис. 26: Время выполнения алгоритма, в миллисекундах

На рисунках 25 и 26 показано время выполнения жадного алгоритма с EDF эвристикой, включая прогоны METIS, на слоистых данных, основанных на слоистых графах. Как и для жадного алгоритма с жадной эвристикой, на данных видно два выброса, которые соотносятся с самым большим количеством процессором и самым маленьким количеством работ в исходных данных. Также не существует значимой зависимости между количеством процессоров в системе и временем, затраченным на построение расписания. Алгоритм работает в несколько раз быстрее жадного алгоритма с жадным критерием.

#### 7.4.2.2 Постановка NO

На рисунках 25 и 26 показано время выполнения жадного алгоритма с EDF эвристикой, включая прогоны METIS, на слоистых данных с неоднородными процессорами. Алгоритм выполняется быстрее жадного алгоритма с жадным критерием, однако разница во времени выполнения незначительна. Время выполнения алгоритма увеличивается с увеличением количества работ и процессоров в системе.

(Greedy\_EDF, NO, Heterog. proc., Layered) Algorithm execution time, ms

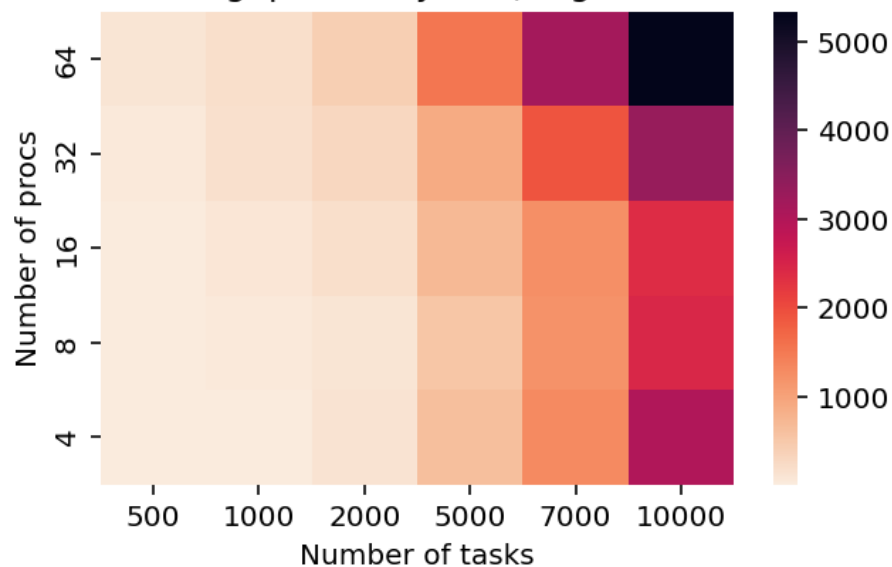


Рис. 27: Время выполнения алгоритма, в миллисекундах (тепловая карта)

(Greedy\_EDF, NO, Heterog. proc., Layered) Algorithm execution time, ms

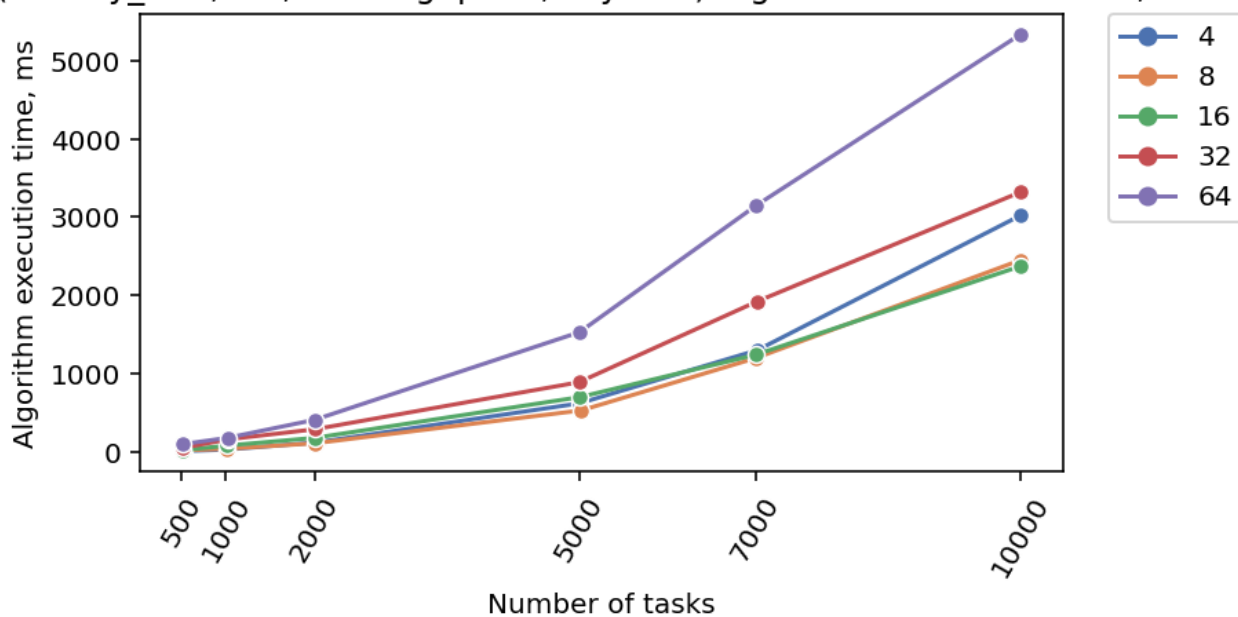


Рис. 28: Время выполнения алгоритма, в миллисекундах

## 8 Заключение

В ходе выполнения курсовой работы были достигнуты все ее цели, а именно:

1. Проведен обзор существующих решений задач. Произведено сравнений других стратегий с жадными критериями и ограниченным перебором. На основе обзора произведен выбор подхода, основанного на комбинации жадных стратегий и ограниченного перебора.
2. Разработан и реализован алгоритм.
3. Произведено исследование свойств алгоритма, которое показало что алгоритм генерирует расписание, превосходящее оптимальное на 8%. Более 90% задач размещены при помощи жадной стратегии.

Предложенный алгоритм, сочетающий жадные стратегии и ограниченный перебор, строит расписание, то есть каждой работе сопоставляет процессор и время старта работы.

При исследовании алгоритма были подобраны оптимальные параметры и определены направления дальнейшего улучшения и исследования алгоритма.

## Список литературы

1. *Akbari M., Rashidi H.* AN EFFICIENT ALGORITHM FOR COMPILE-TIME TASK SCHEDULING PROBLEM ON HETEROGENEOUS COMPUTING SYSTEMS. — 2015. — ЯНВ. — DOI: 10.7813/2075-4124.2015/7-1/A.45.
2. Boost C++ libraries. — URL: <https://www.boost.org/> (дата обр. 02.04.2023).
3. *Coffman E. G.* Computer and job-shop scheduling theory. — Nashville, TN : John Wiley & Sons, 02.1976. — ISBN 0471163198.
4. *Davis R. I., Burns A.* A Survey of Hard Real-Time Scheduling for Multiprocessor Systems // ACM Computing Surveys. — 2011. — Окт. — Т. 43, № 4.
5. Introduction to Algorithms / Т. Н. Cormen [и др.]. — 2nd. — The MIT Press, 2001. — ISBN 0262032937.
6. JSON parsing library. — URL: <https://github.com/nlohmann/json> (дата обр. 02.04.2023).
7. *Karypis G.* METIS and ParMETIS // Encyclopedia of Parallel Computing / под ред. D. Padua. — Boston, MA : Springer US, 2011. — С. 1117—1124. — ISBN 978-0-387-09766-4. — DOI: 10.1007/978-0-387-09766-4\_500. — URL: [https://doi.org/10.1007/978-0-387-09766-4\\_500](https://doi.org/10.1007/978-0-387-09766-4_500).
8. METIS library. — URL: <https://github.com/KarypisLab/METIS> (дата обр. 10.04.2023).
9. *Rzadca K. S. F.* Heterogeneous Multiprocessor Scheduling with Differential Evolution // IEEE Congress on Evolutionary Computation. — 2005. — Т. 3.
10. *Savitsky I.* multiprocessor-scheduling. — URL: <https://github.com/ipsavitsky/greedy-scheduling> (дата обр. 23.05.2022).
11. TOML parsing library. — URL: <https://github.com/ToruNiina/toml11> (дата обр. 02.04.2023).
12. *К.В. Шахбазян Т. Т.* Обзор методов составления расписаний для многопроцессорных систем // Журнал советской математики. — 1981. — Т. 15, № 5. — С. 651—669.
13. *Калашников А. В.* Алгоритмы оптимизации расписаний, основанные на исправлении неоптимальных фрагментов. — 2004.
14. *Костенко В. А.* Алгоритмы комбинаторной оптимизации, сочетающие жадные стратегии и ограниченный перебор // Известия Российской академии наук. Теория и системы управления. — 2017. — № 2. — С. 48—56.
15. *Штовба С. Д.* Муравьиные алгоритмы // Exponenta Pro. Математика в приложениях. — 2003. — № 4. — С. 70—75.

## ПРИЛОЖЕНИЕ 1

### Классы входных данных

В данных, присланных от Хуавей существует разделение на 2 класса.

1. Первый класс (примеры DAG\_A и DAG\_B) характеризуется относительно небольшим масштабом графа работ, небольшим числом процессоров, полнотой графа связности процессоров и одинаковыми задержками между любыми двумя процессорами.
2. Второй класс (примеры DAG\_C и DAG\_D) характеризуется относительно большим масштабом графа работ, большим числом процессоров

Критерии	Примеры входных данных			
	DAG_A	DAG_B	DAG_C	DAG_D
Масштаб графа работ	45 вершин; 75 ребер	1121 вершина; 6229 ребер	197494 вершин; 719389 ребер	1823309 вершин; 6172920 ребер
Разброс длительностей работ	1-10	1-10	все работы одной длины	все работы одной длины
Связность графа работ	1.66	5.55	3.64	3.83
Количество процессоров	2	10	256	4096
Полный граф связности процессоров	да	да	нет	да
Одинаковые задержки на передачу данных	да	да	да	нет

Таблица 5: Сравнение примеров из классов данных