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

Выпускная квалификационная работа

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

Москва, 2023

Аннотация

Построение многопроцессорного расписания это NP-трудная задача. Точный полиномиальный алгоритм для решения этой задачи неизвестен. В данной работе приводятся два возможных варианта решения задачи с дополнительным ограничением на количество передач при помощи жадных алгоритмов.

Содержание

1 Введение

Общая задача построения списочных многопроцессорных расписаний заключается в следующем [2; 3]. Задан ориентированный ациклический граф потока данных. Вершинами графа являются работы. Для каждой работы задано время, требуемое для ее выполнения. Дуги задают отношение частичного порядка на множестве работ. Если от i-ой вершины идет дуга к j-ой вершине, то работа j может начать выполняться только после завершения работы i. Требуется построить расписание выполнения работ на заданном числе процессоров. Расписание корректно если выполняются следующие ограничения:

1. каждая работа должна быть назначена на процессор;
2. любую работу обслуживает лишь один процессор;
3. частичный порядок, заданный в графе потока данных, сохранен в расписании.

Целевой функцией является время выполнения расписания. Прерывания работ недопустимы.

Рассматриваемая в работе задача построения списочных расписаний отличается от общей задачи наличием дополнительного ограничения на корректность расписаний. Задается ограничение на максимально возможное число передач между процессорами. Для решения задач построения расписаний большой размерности на практике обычно используются жадные алгоритмы [5]. Их достоинством является низкая вычислительная сложность. Основным недостатком жадных алгоритмов является сильная зависимость качества получаемых решений от класса исходных данных.

Наличие дополнительного ограничения может приводить к тому, что жадные алгоритмы могут заходить в тупик: есть еще не распределенные в расписание работы, но ни одна из них не может быть размещена в расписание без нарушения дополнительных ограничений на корректность расписания. Для того чтобы исправить этот недостаток был предложен алгоритм, основанный на подходе сочетающий жадные стратегии и ограниченный перебор [9]. Однако при небольшой глубине перебора как показало экспериментальное исследование он также может заходить в тупик. Гарантировано он не будет заходить в тупик если глубина перебора равна числу работ, но при этом он будет иметь не полиномиальную сложность.

В работе рассматриваются два алгоритма построения списочных расписаний для классической задачи и с дополнительным ограничением на количество межпроцессорных передач, сочетающих жадные стратегии и алгоритм разбиения графа потока данных на подграфы.

2 Цели и задачи этой работы

Целью этой работы является разработка детерминированного алгоритма построения многопроцессорного расписания с дополнительными ограничениями.

Для достижения указанной цели требуется:

1. Провести аналитический обзор детерминированных алгоритмов построения списочных расписаний с целью выявления алгоритмов, которые возможно модифицировать под поставленную задачу и имеют хорошую возможность масштабирования.

2. Разработать и реализовать алгоритмы.

3. Провести экспериментальное исследование по критериям качества решений и временной сложности алгоритмов.

3 Задача построения списочных расписаний

В задаче построения расписания заданы:

1. Граф потока управления G без циклов, в котором дуги - зависимости по данным, а вершины - задания. Вершин n, дуг m
2. Вычислительная система, состоящая из p процессоров.
3. Матрица $C_{n\times p}$ времени выполнения работ на процессорах. Каждая строка этой матрицы - длины выполнения n-й задачи на p процессорах.
4. Время d, затрачиваемое на межпроцессорную передачу.

Расписание 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 являющиеся отношением частичного порядка, удовлетворяющее условиям ацикличности и транзитивности. Отношение порядка на множестве работ, распределенных на один процессор, является отношением полного порядка.

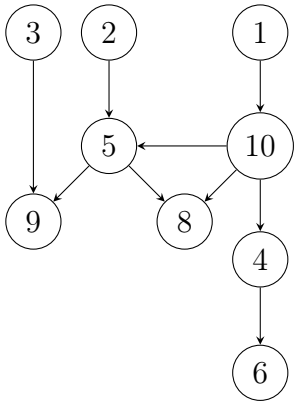


Рис. 1: Граф G потока данных

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

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

1. Графическое представление.

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

2. Временная диаграмма.

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

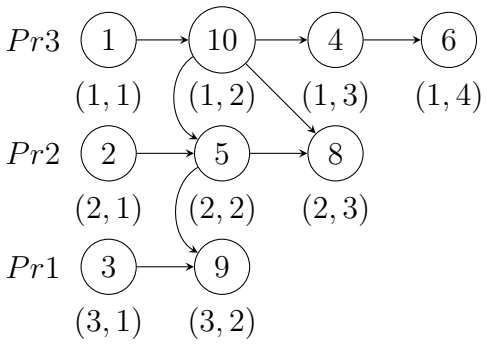


Рис. 2: Графическое представление расписания

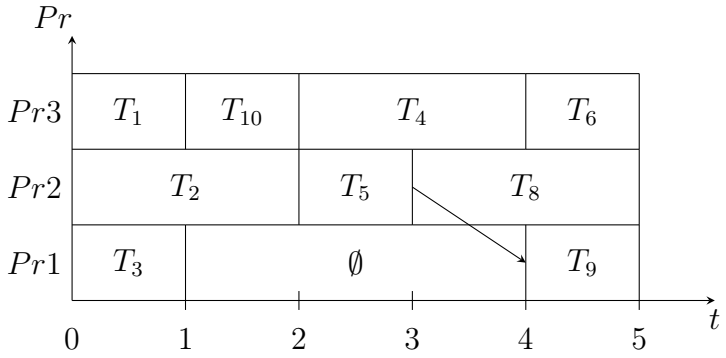


Рис. 3: Представление расписания в виде временной диаграммы

Доказано, что эти представления полностью эквивалентны и, имея одно, возможно построить другое. В предложенном решении расписание строится в виде временной диаграммы. Стоит отметить, что во временной диаграмме иогут возникать простои в расписании, как, например, на рисунке 3 в системе с длиной межпроцессорной передачи $d = 1$, между работами T_3 и T_9 на процессоре $Pr1$, поскольку работа T_9 зависит от работы T_5, которая размещена на другой процессор.

Из входных данных задачи требуется 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, минимизировать время выполнения расписания, выполнив дополнительные ограничения.

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 стартом 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).

3.1 Различные постановки задачи

1. Задача без дополнительных ограничений на расписание. Далее эта постановка будет упоминаться как **постановка** NO

2. Задача с дополнительным ограничением на количество передач:

   - $CR = \frac{m_{ip}}{m} < 0.4$, где m_{ip} - количество передач данных между работами на каждый процессор. Далее эта постановка будет упоминаться как **постановка** CR

4 Обзор предметной области

4.1 Критерии обзора

Ниже приведены критерии, по которым будут рассматриваться и сравниваться алгоритмы

1. Возможность модифицировать алгоритм под поставленную задачу. Из необходимых модификаций требуется добавить возможность учета дополнительного критерия межпроцессорных передач и возможность учета затрат на межпроцессорные передачи.

2. Возможность масштабирования алгоритма.

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 последействие (распределяемый в расписание рабочий интервал оказывает влияние на времена инициализации ранее распределенных рабочих интервалов) возникает проблема выбора локальных критериев оптимизации, позволяющих учесть эффект последействия (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 последействия 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.

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 [6]) 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

4.2.2 Алгоритмы, основанные на методе динамического программирования

Алгоритмы динамического программирования разбивают сложную задачу на более простые подзадачи и находят обратную связь между их оптимальными решениями. Алгоритмы из этой области могут предоставлять глобальные оптимальные решения, но их недостатком является неполиномиальная сложность. В частности, с точки зрения NP-сложных задач планирования сложность этих алгоритмов является экспоненциальной функцией размера входных данных. Кроме того, для нахождения обратной зависимости между оптимальными решениями и подзадачами необходима модель аналитической системы. Это означает, что алгоритм не подходит для решения данной задачи, так как имеет высокую вычислительную сложность [1].

4.2.3 Алгоритмы, основанные на методе ветвей и границ

Алгоритм ветвей и границ является эффективным методом решения производных задач оптимизации. Этот метод делит пространство потенциальных решений на различные области, дает точные оценки их значения по отношению к целевой функции и сокращает ненужные участки, где невозможно найти оптимальное решение. Хотя глобальные оптимальные решения могут быть получены с помощью этого метода, его сложность для большинства задач комбинаторной оптимизации неполиномиальна. Особенно для NP-сложных задач планирования сложность является факториальной функцией размера входных данных. Следовательно, эти алгоритмы не подходят для решения проблемы.

Алгоритм, обсуждаемый в [7], был протестирован с использованием наборов данных с числом процессоров до 16 и графов, содержащих до 100 вершин. Результаты показали ожидаемый экспоненциальный рост времени работы.

4.2.4 Алгоритмы, основанные на нахождении максимального потока в сети

Алгоритмические методы составления многопроцессорных расписаний включают поиск максимального потока в транспортной сети, которые, по сути, переводят процесс построения расписания в поиск максимально возможного потока в указанной сети. Указанная сеть строится на основе набора задач, процессоров и параметров исходного задания. После построения сети выполняется процесс поиска максимального потока. Затем, с помощью значений потока в сети, возможно построить многопроцессорное расписание [4]. Этот конкретный алгоритм подходит только для задач, допускающих прерывания в вычислительной системе, поэтому в данной задаче его нельзя использовать.

4.3 Выводы из обзора предметной области

В таблице 1 представлены сокращенные результаты обзора.

В результате обзора предметной области, под критерии масштабируемости и соответствия задаче подходит жадный алгоритм. По результатам не было найдено работ, точно соответствующим постановке задачи, поэтому предложенный подход требуется модифицировать.

Название алгоритма	Возможность модификации алгоритма	Возможность масштабирования алгоритма
Метод ветвей и границ	✓	✗
Метод динамического программирования	✓	✗
Алгоритм поиска максимального потока	✗	✓
Жадные алгоритмы	✓	✓

Таблица 1: Существующие детерминированные алгоритмы

5 Алгоритм построения расписания

5.1 Основные алгоритмы

5.1.1 Общее описание жадных алгоритмов построения многопроцессорного расписания

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

1. Выбрать работу для постановки в расписание.
2. Выбрать процессор 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

Данная схема представлена на рисунке 4

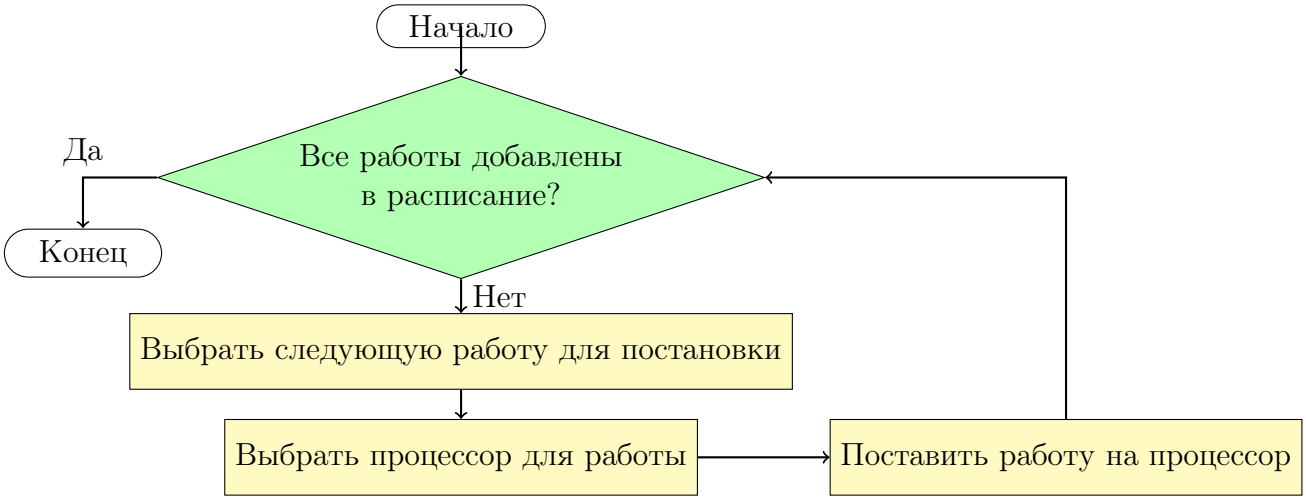


Рис. 4: Блок-схема жадных алгоритмов построения расписания

5.1.2 Жадный алгоритм с выбором по количеству потомков

Данный алгоритм 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 постановке задачи в

расписание, в случае невозможности постановки без нарушения дополнительного ограничения. В таком случае, алгоритм завершается.

Задача d_i называется **доступной для постановки** в расписание, в случае, если либо у нее нет предшественников, либо все ее предшественники уже постановлены в расписание. Назовем множеством всех доступных для постановки задач $D = (d_0, d_1, \ldots, d_n)$.

Жадный алгоритм с выбором по числу потомков выбирает задачу для постановки по следующему критерию: пусть $Succ(d)$ - функция, определяющая количество непосредственных последователей работы в графе. Тогда $\forall d_j \in D, d_j \neq d_i : Succ(d_j) < Succ(d_i)$.

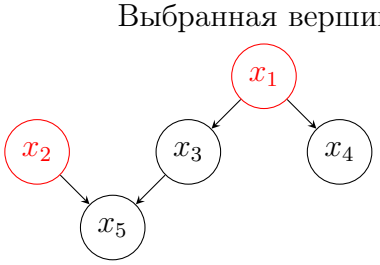


Рис. 5: Критерий выбора работы для постановки в расписание

На рисунке 5 $D = (x_1, x_2)$, из которых $Succ(x_1) = 2, Succ(x_2) = 1$. На постановку будет выбрана вершина x_1.

Для постановки с дополнительным ограничением CR жадный алгоритм с выбором по числу потомков берет распределение работ по процессорам из специального алгоритма распределения (описание алгоритма приведено в разделе 5.2.2), поэтому выбор процессора в п.2 всегда заранее определен. Выбор начала выполнения работы в расписание производится в соответствии с алгоритмом постановки задачи на процессор (описание алгоритма приведено в разделе 5.2.3).

Для постановки без дополнительных ограничений, жадный алгоритм с выбором по числу потомков производит пробную постановку на каждый процессор и выбирает процессор с самым ранним временем завершения работы с учетом алгоритма постановки задачи на процессор (описание алгоритма приведено в разделе 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 работы A до работы S : $Succ(S) = \emptyset$.

2. Рассчитать длину этого пути. Длина пути равна сумме всех задержек передач данных и времен выполнения работ. Задержки передач данных известны, так как известно распределение работ на процессоры. Пусть длина этого пути равна p.

3. $d_A \coloneqq d - p$

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 возрастанию.

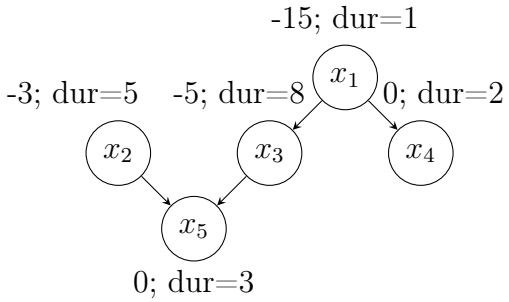


Рис. 6: Пример распространения фиктивных директивных сроков

На рисунке 6 представлен пример распространения фиктивных директивных сроков по графу потока данных. На данном примере все передачи, кроме передачи между работами x_2 и x_5, межпроцессорные. Время d межпроцессорной передачи равно 2. Фиктивные директивные сроки листовых вершин x_5 и x_4 равны 0. Таким образом для расчета остальных директивных сроков:

1. $x_2 = 0 - 3$ (фиктивный директивный срок потомка) $= -3$
2. $x_3 = 0 - 3 - 2$ (затрата на межпроцессорную передачу) $= -5$
3. $x_1 = \min(-5 - 8 - 2, 0 - 2 - 2) = \min(-15, -4) = -15$, из фиктивных директивных сроков от нескольких потомков выбирается минимальный.

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 распределение (5.2.2), для постановки с неоднородными процессорами строится невзвешенное распределение, 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 производится пробная постановка 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 распределения

Этот алгоритм используется только для постановки CR с неоднородными процессорами. Подразумевается, что процессоры упорядочены по возрастанию производительности, то есть для любых процессоров P_1, P_2, если 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 распределении от специального алгоритма распределения вершин по подграфам 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 $\max(\text{загрузка}P_1, \text{загрузка}P_2)$, где загрузка$P$ это количество работ, распределенных на процессор P, 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 распределения графа на подграфы METIS [8].

Для построения распределения работ на процессоры запускается алгоритм разбиения графа на подграфы с количеством подграфов, равным количеству процессоров и минимизируемым критерием - минимизицией количества ребер между подграфами, после чего каждый подграф 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 равный, конкретный процессор в которого берется время выполнения не имеет значения.

Распределение, лучшее по балансу подграфов, 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 среднему количеству работ в подграфе. Другими словами, `ufactor` позволяет контролировать дисбаланс в количестве вершин в подграфе. С увеличением этого параметра, CR понижается. Для генерации распределения, удовлетворяющего дополнительное ограничение 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, работа будет поставлена до начала другой, не связанной с ней работы, 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.

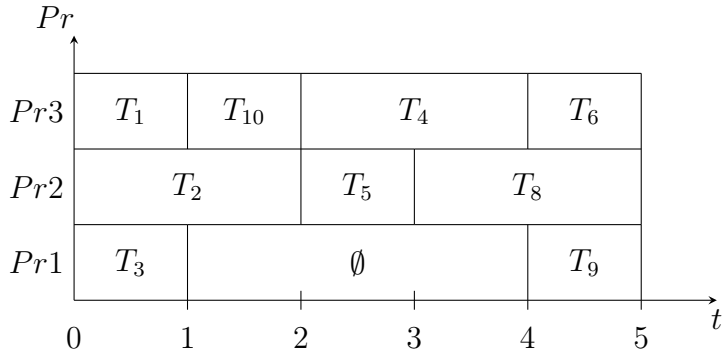


Рис. 7: Расписание до добавления работы T_7

Пусть до добавления работы T_7 расписание находится в состоянии 7. T_7 распределена на процессор $Pr1$ и имеет длительность 1, имеет одного предка T_2, поставленного на процессор $Pr2$. Длительность межпроцессорной передачи равна 1. Таким образом, есть возможность поставить работу в простой, начинающийся в отрезок времени 3, как на рисунке 8.

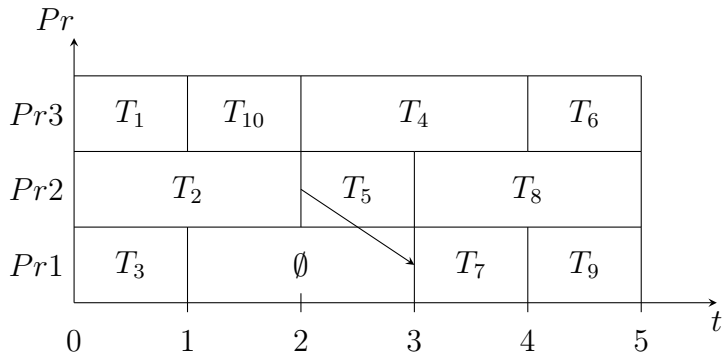


Рис. 8: Расписание после добавления работы T_7

6 Программная реализация алгоритма

6.1 Описание кода программной реализации

Код реализации выложен на `C++` в репозитории [14]. Диаграмма классов реализации представлена на рисунке 9.

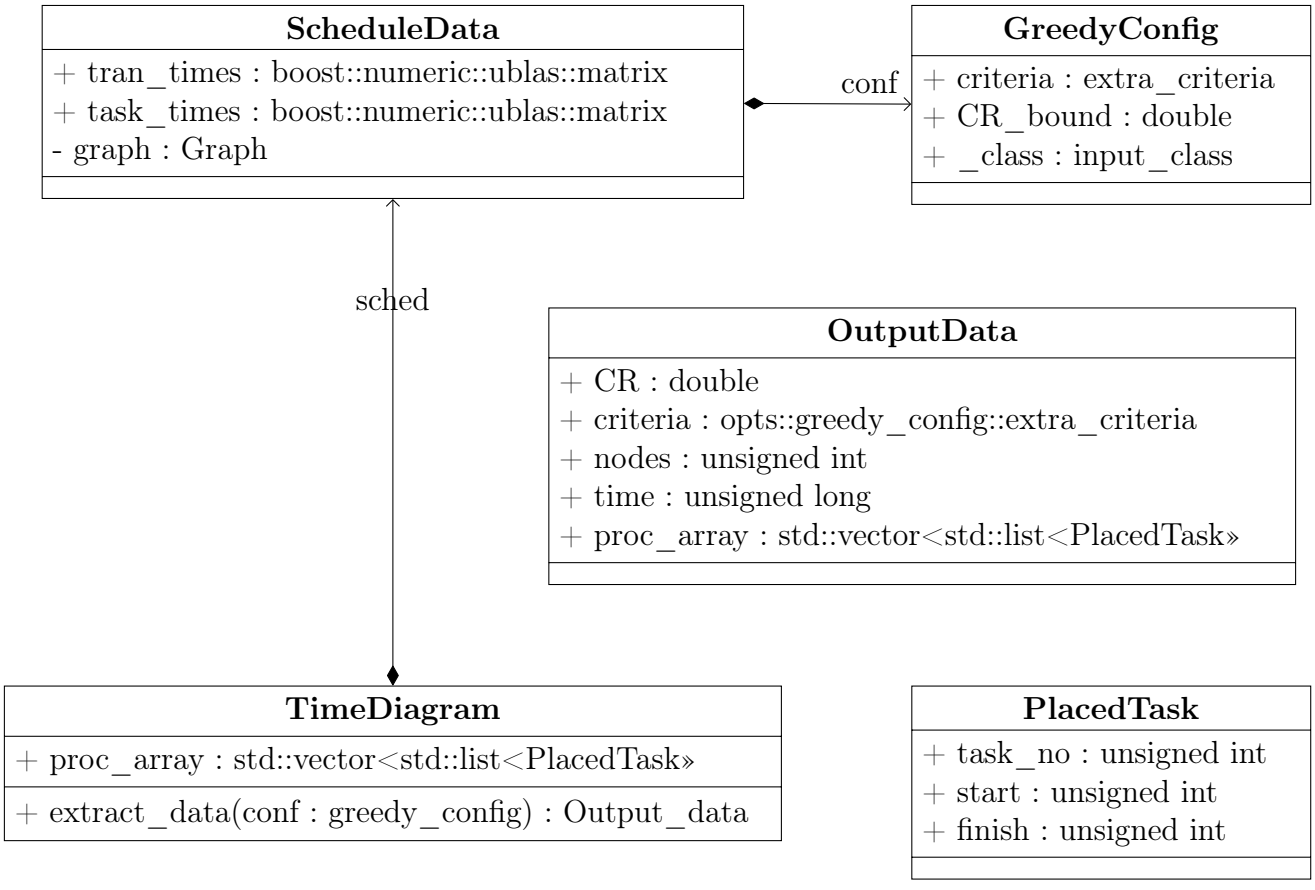


Рис. 9: UML-диаграмма реализации

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 выбором по числу потомков.
- `greedy_EDF_heuristic()` - Жадный алгоритм с EDF эвристикой.

В репозиторий включены следующие библиотеки:

1. `METIS` 5.1.0 [13] - библиотека для распределения графов.
2. `json` 3.11.2 [12] - библиотека для работы с форматом JSON. Используется для составления выходных файлов.
3. `toml11` 3.7.1 [15] - библиотека для работы с форматом TOML. Используется для чтения конфигурационных файлов.

Также, у реализации есть зависимость, не включенная в репозиторий - `boost` 1.80 [11]. 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
`algorithm_name`	Название алгоритма для построения расписания
`input`	Путь к файлу с входными данными
`output`	Путь к файлу с выходными данными
`conf`	Путь к файлу с конфигурацией
`log`	Уровень логирования

Таблица 2: Параметры командной строки программы

```toml
[general]
criteria = "CR"
CR_bound = 0.4
inp_class = "class_1"

[greedy]
cr_con = false
```

Листинг 4: Пример конфигурационного файла

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
`criteria`	Критерий, дополнительное ограничение котрого будет выполняться (CR / NO)
`CR_bound`	Верхняя граница ограничения CR (если используется)
`inp_class`	Класс типа входных данных • `class_1` для постановки с однородными процессорами • `class_general` для постановки с неоднородными процессорами
`cr_con`	Переключение жадного критерия в жадном алгоритме с жадными критериями с максимального количества потомков на максимальное количество предков.

Таблица 3: Параметры конфигурационного файла.

6.2.3 Описание входных файлов

В качестве формата входных файлов был разработан специальный формат, определяющий собой вычислительную систему. Он представляет собой текстовый файл, разделы которого приведены в таблице 4.

`8 3 5`	Числа n (количество работ), p (количество процессоров) и m (количество ребер)

`1 3 2 4 6 8 1 1`
`1 3 2 4 6 8 1 1`
`2 6 4 8 12 16 2 2`
	Матрица C времени выполнения работ на процессорах размером $p \times n$. C_{ij} соответствует времени выполнения работы i на процессоре j.

`2`	Время, затрачиваемое на межпроцессорную передачу.

`0 1`
`0 5`
`1 2`
`1 3`
`2 7`
	Пары, соответствующие ребрам в графе, представленные в виде m строк. i j означает, что в графе потока управления пристуствует ребро из вершины i в вершину j.

Таблица 4: Поля входного файла

6.2.4 Описание выходных файлов

В качестве формата выходных файлов был выбран формат `json`. Пример выходного файла приведен в листинге 5 и таблице 5. Выходной файл содержит информацию о характеристиках построенного расписания, а так же информацию о привязках и порядке постановке работ на процессорах.

Поле	Описание
`CR`	Значение ограничения CR построенного расписания.
`algo_time`	Время выполнения алгоритма, в миллисекундах
`criteria`	Дополнительное ограничение, используемое для построения рапсисания
`nodes`	Количество работ во входном графе.
`time`	Время выполнения построенного расписания
`procs`	Словарь с номерами процессоров в качестве ключей и массивами поставленных на соответствующий процессор работами. Каждая поставленная работа состоит из: • `task_dur` - время выполнения работы на распределенный процессор. • `task_no` - идентификатор работы. • `task_start` - время начала выполнения работы на процессоре.

Таблица 5: Поля выходного файла

.

```
{
    "CR": 0.3221312,
    "algo_time": 300,
    "criteria": "CR",
    "nodes": 2000,
    "procs": {
        "0": [
            {
                "task_dur": 5,
                "task_no": 1202,
                "task_start": 0
            },
            {
                "task_dur": 3,
                "task_no": 1608,
                "task_start": 5
            },
            ...
        ],
        "1": [
            ...
        ],
        ...
    },
    "time": 2211
}
```

Листинг 5: Пример выходного файла

7 Экспериментальное исследование алгоритма

7.1 Цели и методика экспериментального исследования

Целями экспериментального исследования было поставлено исследование:

- Качество решений, получаемых алгоритмами.
- Время работы алгоритма.

Для проведения экспериментов было сгенерированно 3 набора данных:

1. Набор данных с известным оптимумом
2. Набор данных, основанных на слоистых графах, без известного оптимума, но с однородными процессорами. Данные из пункта 1 имеют свойство идеально сбалансированного распределения, то есть от METIS всегда построит распределение, близкое к распределению идеального расписания. Чтобы проверить, как ведут себя алгоритмы на данных без такого свойства, были добавлено исследование на данных слоистых графах.
3. Набор данных, основанный на слоистых графах, без известного оптимума, но с неоднородными процессорами.

Схема генерации слоистых графов описана в [10].

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 выбором по числу потомков

7.3.1.1 Постановка задачи с ограничением на количество передач

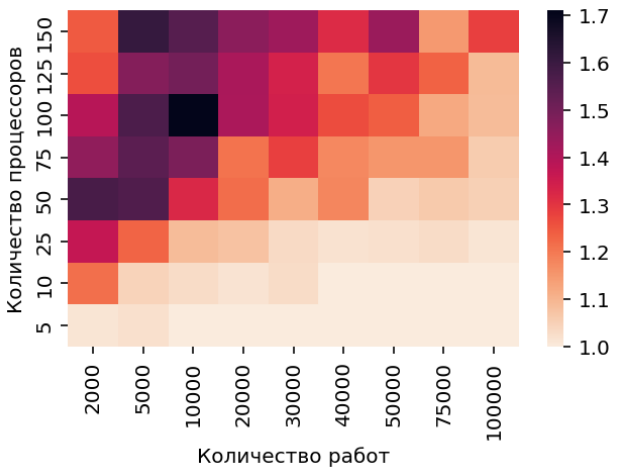
На рисунках 10a и 10b показано качество решений, генерируемых жадных алгоритмом с выбором по числу потомков на данных с известным оптимумом. Цветом на рисунке 10a и значением на оси Oy на рисунке 10b показано отношение длительности расписания, построенного алгоритмом к длительности оптимального расписания. Значения всегда больше 1, чем меньше, тем лучше.

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

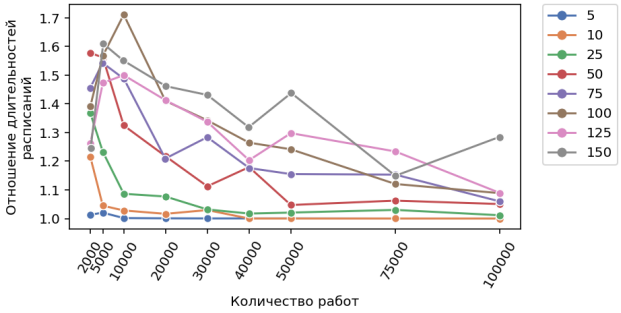
7.3.1.2 Постановка задачи без ограничений

На рисунках 11a и 11b показано качество решений, генерируемых жадных алгоритмом с выбором по числу потомков на данных с известным оптимумом. Цветом на рисунке 11a и значением на оси Oy на рисунке 11b показано отношение длительности расписания, построенного алгоритмом к длительности оптимального расписания. Значения всегда больше 1, чем меньше, тем лучше.

Точность алгоритма возрастает с увеличением количества работ и понижается с увеличением количества процессоров.

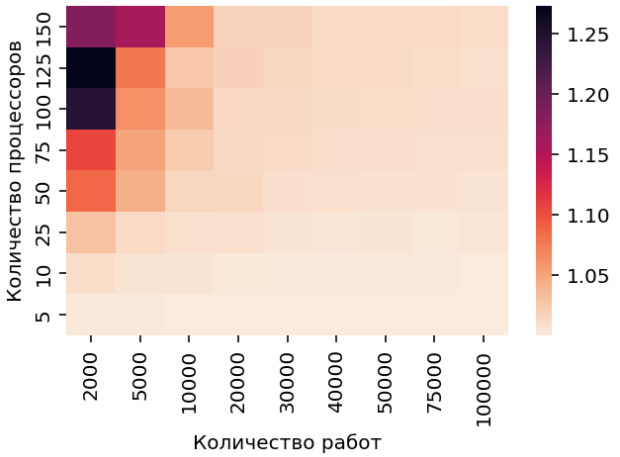


(a) Тепловая карта

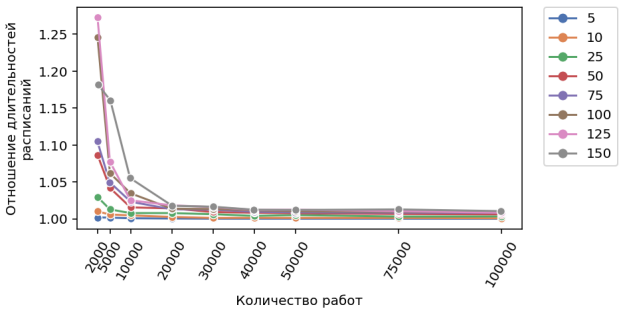


(b) Сводный график

Рис. 10: Отношение времени выполнения расписания к оптимальному времени выполнения на данных с известным оптимумом на постановке CR



(a) Тепловая карта

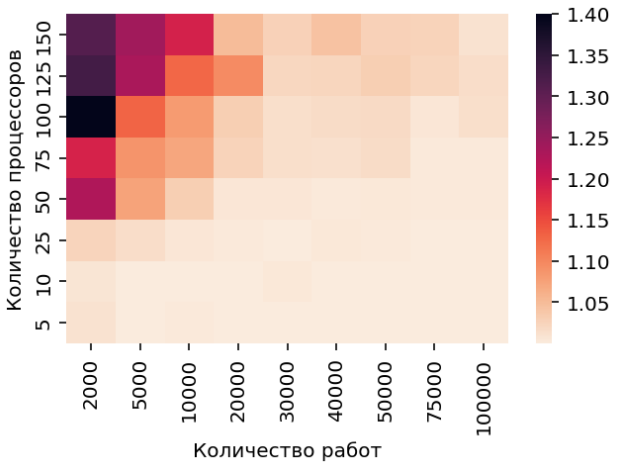


(b) Сводный график

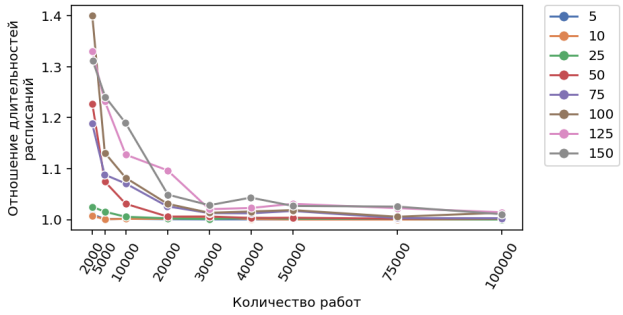
Рис. 11: Отношение времени выполнения расписания к оптимальному времени выполнения на данных с известным оптимумом на постановке NO

7.3.2 Жадный алгоритм с EDF эвристикой

7.3.2.1 Постановка задачи с ограничением на количество передач



(a) Тепловая карта

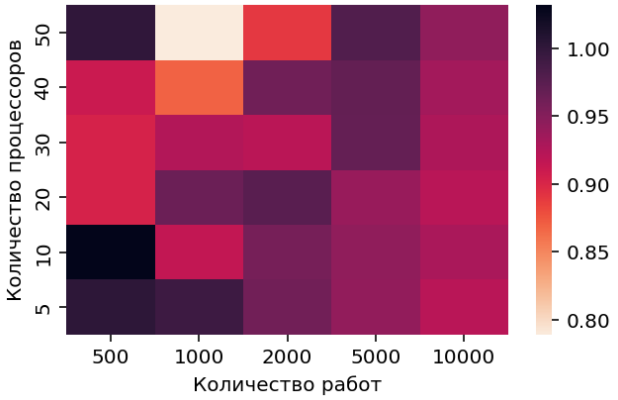


(b) Сводный график

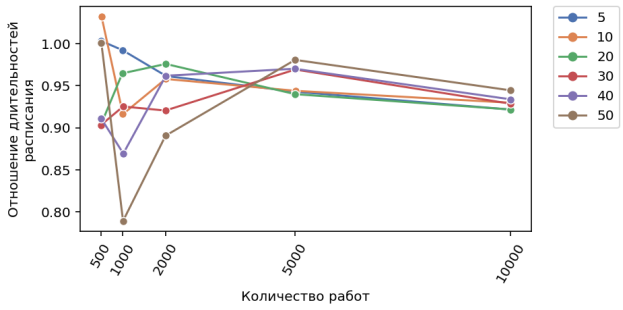
Рис. 12: Отношение времени выполнения расписания к оптимальному времени выполнения на данных с известным оптимумом на постановке CR

На рисунках 12a и 12b показано качество решений, генерируемых жадным алгоритмом с EDF эвристикой на данных с известным оптимумом. Цветом на рисунке 12a и значением на оси Oy на рисунке 12b 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 выбором по числу потомков.



(a) Тепловая карта



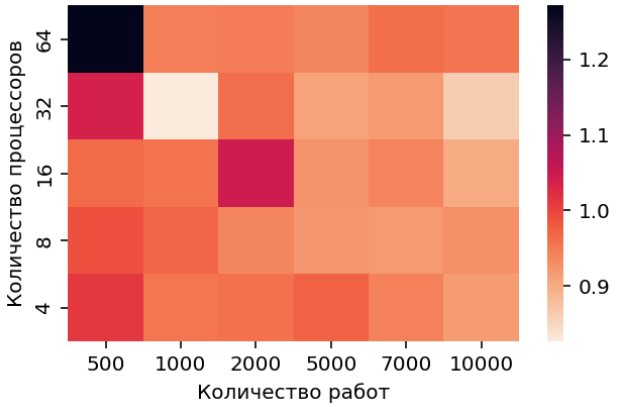
(b) Сводный график

Рис. 13: Отношение времени выполнения расписания, построенного при помощи жадного алгороитма с EDF эвристикой к времени выполнения расписания, построенного при помощи жадного алгоритма с выбором по числе потомков на наборе данных, основанных на слоистых данных на постановке CR

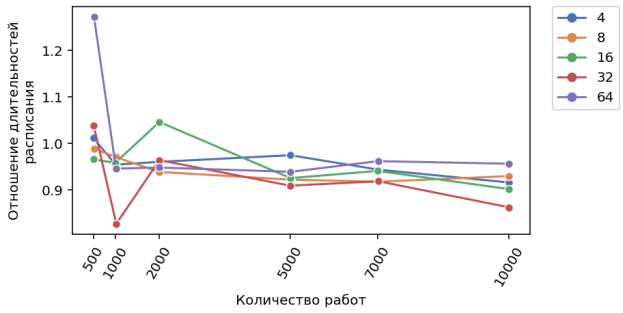
На рисунках 13a и 13b показано качество решений, генерируемых жадным алгоритмом с EDF эвристикой на данных, построенных на слоистых графах. Цветом на рисунке 13a

и значением на оси Oy на рисунке 13b показано отношение длительности расписании расписании, построенного жадным алгоритмом с выбором по числу потомков.

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



(a) Тепловая карта



(b) Сводный график

Рис. 14: Отношение времени выполнения расписания, построенного при помощи жадного алгороитма с EDF эвристикой к времени выполнения расписания, построенного при помощи жадного алгоритма с выбором по числе потомков на наборе данных, основанных на неоднородных процессорах на постановке CR

На рисунках 14a и 14b показано качество решений, генерируемых жадным алгоритмом с EDF эвристикой на данных, построенных на слоистых графах. Цветом на рисунке 14a и значением на оси Oy на рисунке 14b показано отношение длительности расписании расписании, построенного жадным алгоритмом с выбором по числу потомков.

Алгоритм не дает значимых улучшений решения по сравнению с жадным алгоритмом с выбором по числу потомков, кроме случая с 500 работами на 64 процессорах, в котором решение, строимое жадным алгоритмом с EDF эвристикой значительно хуже решения, построенного жадным алгоритмом с выбором по числу потомков.

7.3.2.2 Постановка задачи без ограничений

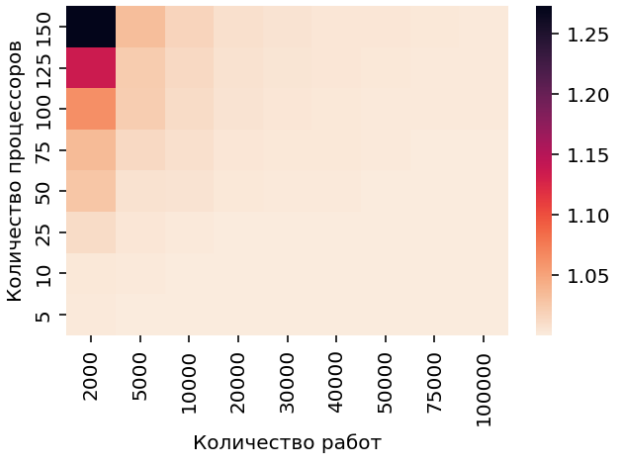
На рисунках 15a и 15b показано качество решений, генерируемых жадным алгоритмом с EDF эвристикой на данных с известным оптимумом. Цветом на рисунке 15a и значением на оси Oy на рисунке 15b показано отношение длительности расписания, построенного жадным алгоритмом с выбором по числу потомков. Значения всегда больше 1, чем меньше, тем лучше.

Алгоритм хорошо справляется с задачей, в большинстве случаев отклонение от оптимума не превышает 5%. Для 2000 и 5000 работ точность алгоритма значительно выше таковой для жадного алгоритма с выбором по числу потомков.

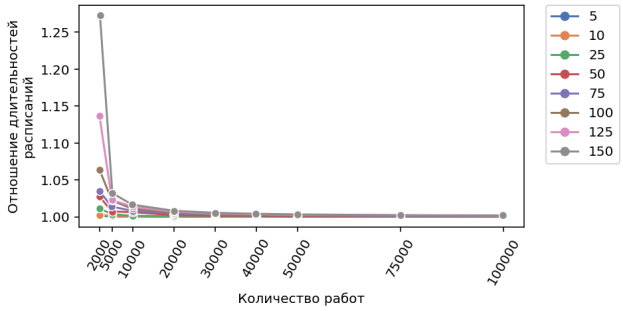
На рисунках 16a и 16b показано качество решений, генерируемых жадным алгоритмом с EDF эвристикой на данных, построенных на слоистых графах. Цветом на рисунке 16a и значением на оси Oy на рисунке 16b показано отношение длительности расписания, построенного жадным алгоритмом с выбором по числу потомков.

Во всех случаях, качество решений выше качества решений жадного алгоритма с выбором по числу потомков, но это улучшение не превышает 10%.

На рисунках 17a и 17b показано качество решений, генерируемых жадным алгоритмом с EDF эвристикой на данных, построенных на слоистых графах. Цветом на рисунке 17a

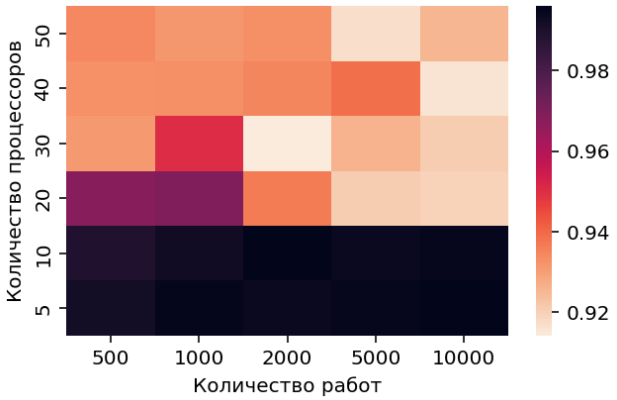


(a) Тепловая карта

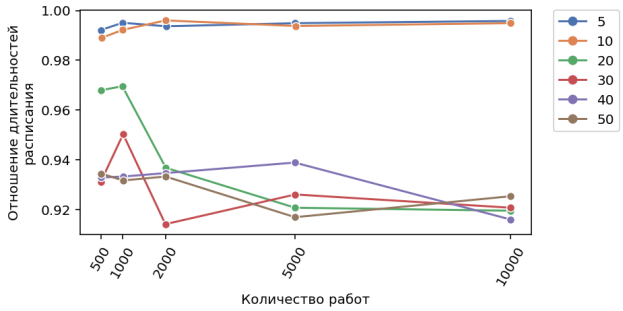


(b) Сводный график

Рис. 15: Отношение времени выполнения расписания, построенного жадным алгоритмом с EDF эвристикой к оптимальному времени выполнения расписания на постановке NO

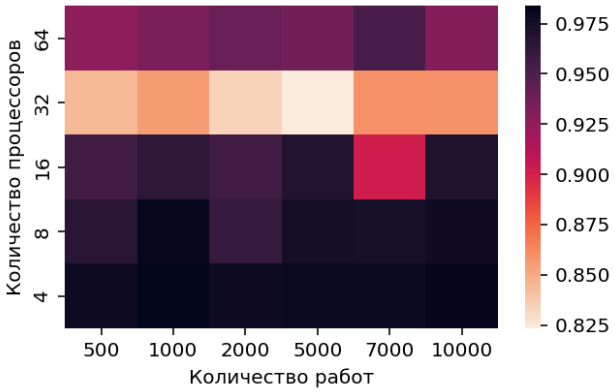


(a) Тепловая карта

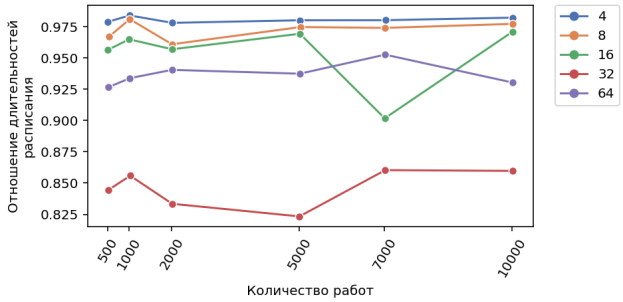


(b) Сводный график

Рис. 16: Отношение времени выполнения расписания, построенного жадным алгоритмом с EDF эвристикой к времени выполнения расписания, построенного жадным алгоритмом с выбором по числу потомков на данных, основанных на слоистых графах, на постановке NO



(a) Тепловая карта



(b) Сводный график

Рис. 17: Отношение времени выполнения расписания, построенного жадным алгоритмом с EDF эвристикой к времени выполнения расписания, построенного жадным алгоритмом с выбором по числу потомков на данных, основанных на неоднородных графах, на постановке NO

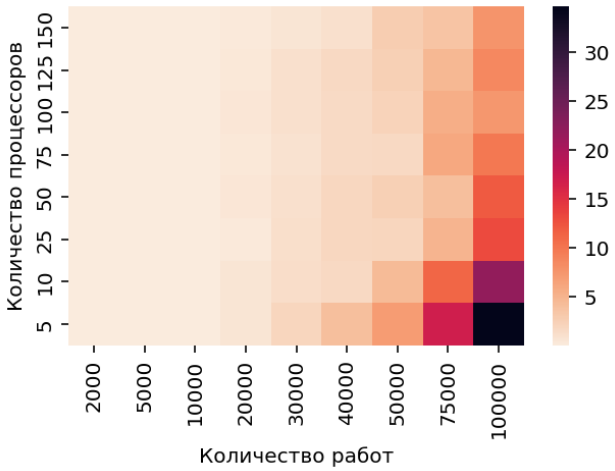
и значением на оси Oy на рисунке 17b показано отношение длительности расписании расписании, построенного жадным алгоритмом с выбором по числу потомков. Значения всегда больше 1, чем меньше, тем лучше.

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

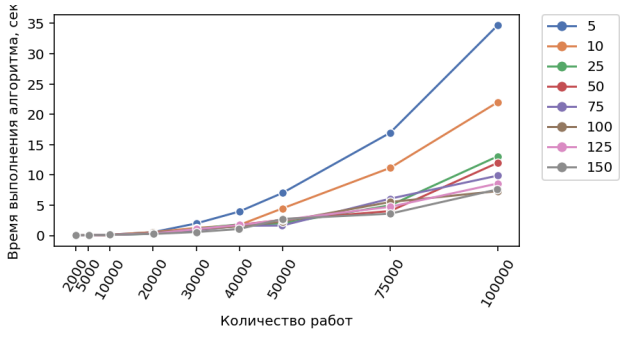
7.4 Исследование временной сложности алгоритма

7.4.1 Жадный алгоритм с выбором по числу потомков

7.4.1.1 Постановка задачи с ограничением на количество передач



(a) Тепловая карта

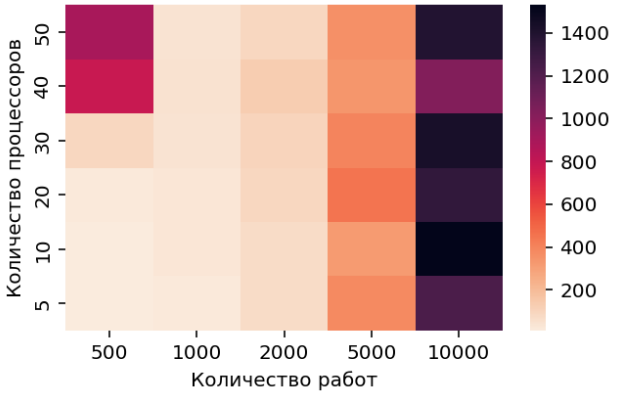


(b) Сводный график

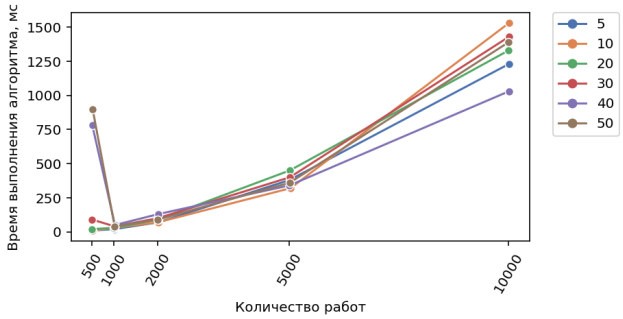
Рис. 18: Время выполнения жадного алгоритма с выбором по числу потомков на данных с известным оптимумом, на постановке CR, в секундах

На рисунках 18a и 18b показано время выполнения жадного алгоритма с выбором по числу потомков, включая прогоны METIS. Время выполнения растет с увеличением ко-

личества 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.



(a) Тепловая карта

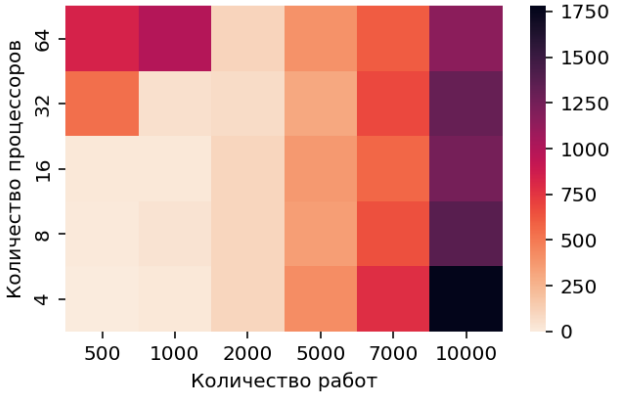


(b) Сводный график

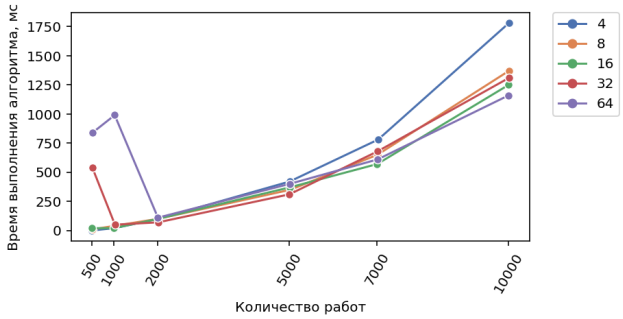
Рис. 19: Время выполнения жадного алгоритма с выбором по числу потомков на данных, основанных на слоистых графах, на постановке CR, в миллисекундах

На рисунках 19a и 19b 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.



(a) Тепловая карта



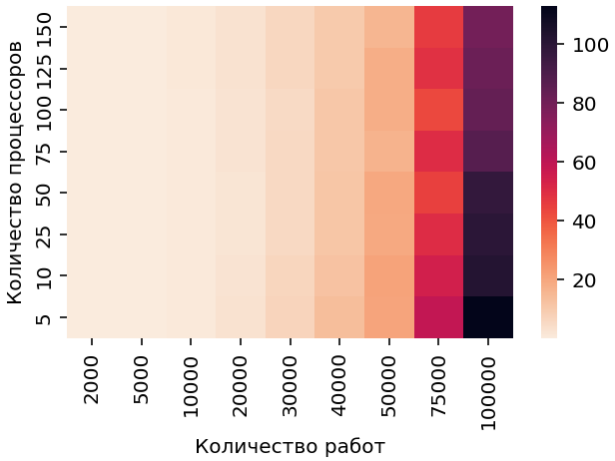
(b) Сводный график

Рис. 20: Время выполнения жадного алгоритма с выбором по числу потомков на данных, основанных на неоднородных процессорах, на постановке CR, в миллисекундах

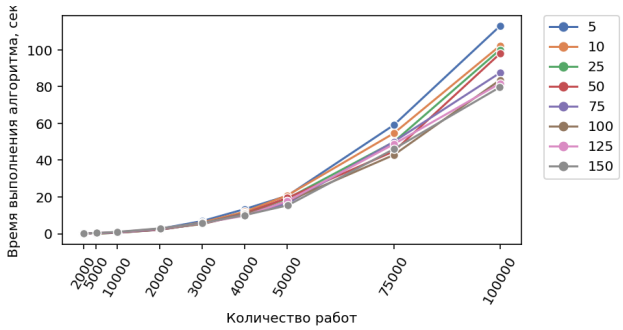
На рисунках 20a и 20b показано время выполнения алгоритма на наборе данных, основанных на слоистых графах с неоднородными процессорами.

Время выполнения алгоритма растет с увеличением количества работ, кроме трех выбросов. Эти выбросы соответствуют самому малому количеству работ и самому высокому количеству процессоров.

7.4.1.2 Постановка задачи без ограничений



(a) Тепловая карта

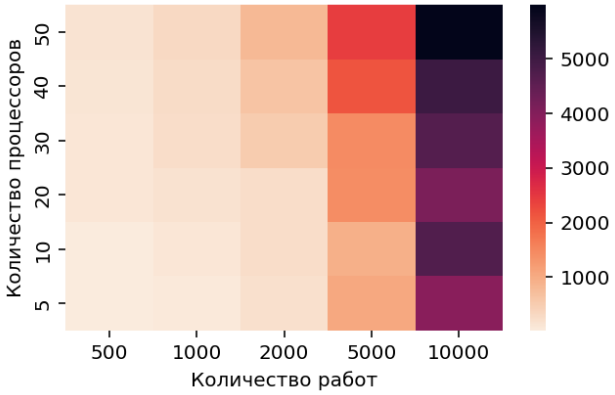


(b) Сводный график

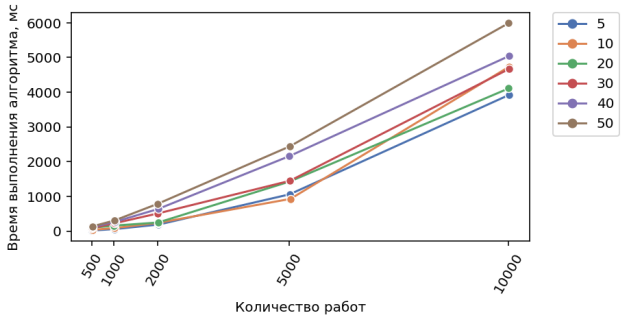
Рис. 21: Время выполнения жадного алгоритма с выбором по числу потомков на данных с известным оптимумом, на постановке NO, в секундах

На рисунках 21a и 21b показано время выполнения алгоритма на наборе данных с известным оптимумом. Время выполнения растет с увеличением количества задач.

При одинаковом количестве работ, быстрее выполняется алгоритм с большим количеством процессоров, по тем же причинам, что и в постановке с дополнительным ограничением CR.



(a) Тепловая карта

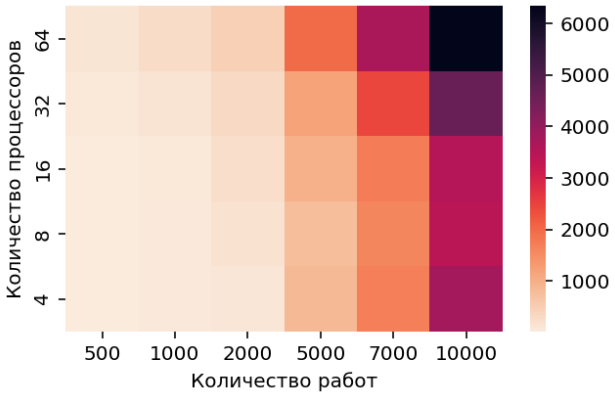


(b) Сводный график

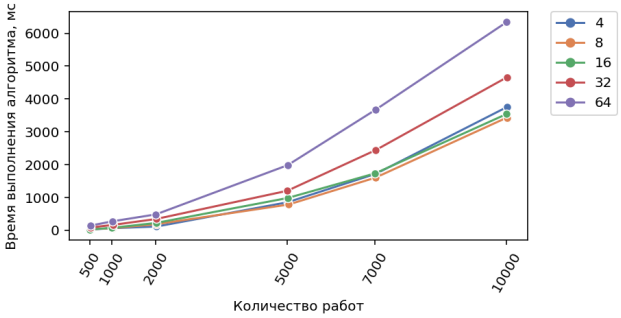
Рис. 22: Время выполнения жадного алгоритма с выбором по числу потомков на данных, основанных на слоистых графах, на постановке NO, в миллисекундах

На рисунках 21a и 21b показано время выполнения алгоритма на наборе данных, основанных на слоистых графах. Время выполнения растет с увеличением количества работ и количества процессоров.

На рисунках 23a и 23b показано время выполнения алгоритма на наборе данных с неоднородными процессорами. Время выполнения растет с увеличением количества работ и количества процессоров.



(a) Тепловая карта

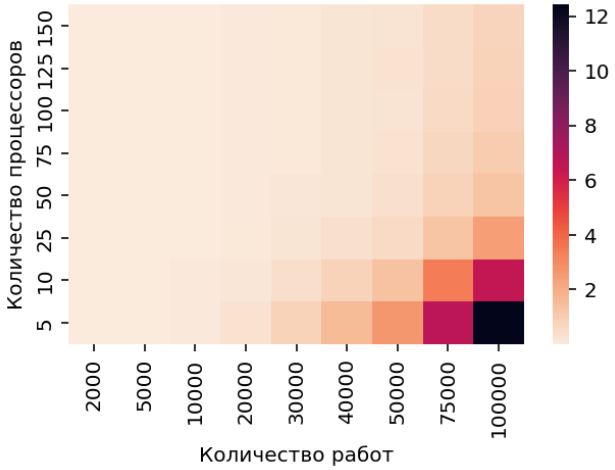


(b) Сводный график

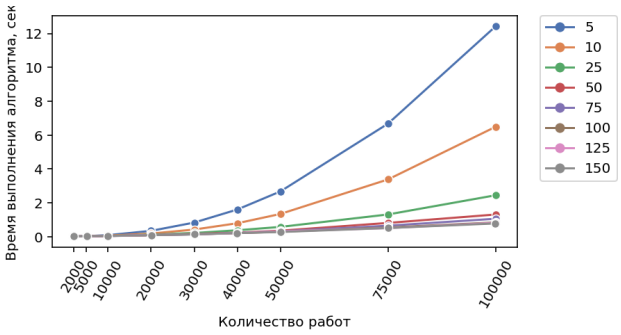
Рис. 23: Время выполнения жадного алгоритма с выбором по числу потомков на данных, основанных на неоднородных процессорах, на постановке NO, в миллисекундах

7.4.2 Жадный алгоритм с EDF эвристикой

7.4.2.1 Постановка задачи с ограничением на количество передач



(a) Тепловая карта

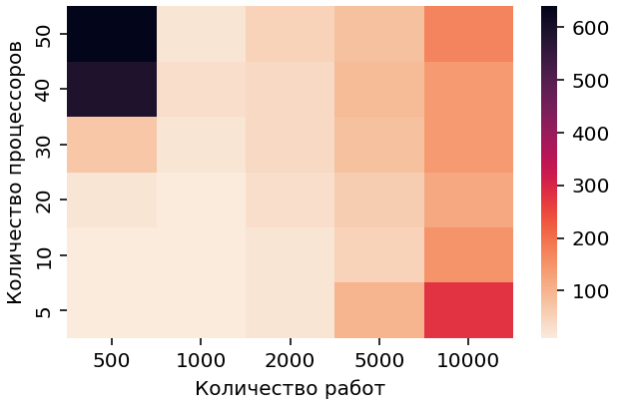


(b) Сводный график

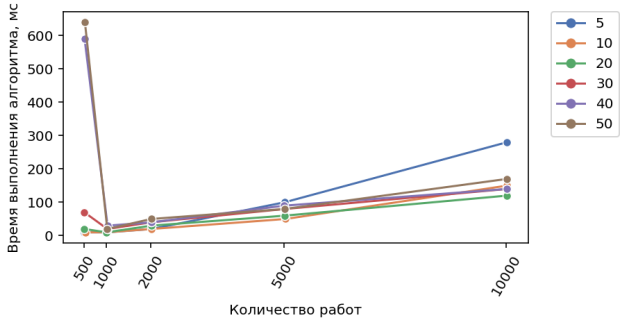
Рис. 24: Время выполнения жадного алгоритма с EDF эвристикой на данных с известным оптимумом, на постановке CR, в секундах

На рисунках 24a и 24b 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 выбором по числу потомков. При равном количестве работ, выше время выполнения при меньшем количестве процессоров. Причина схожа с причинами подобного явления для жадного алгоритма с выбором по числу потомков, поскольку они разделяют одну процедуру поиска нового места в расписании.

На рисунках 25a и 25b показано время выполнения жадного алгоритма с EDF эвристикой, включая прогоны METIS, на слоистых данных, основанных на слоистых графах. Как и для жадного алгоритма с выбором по числу потомков, 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.



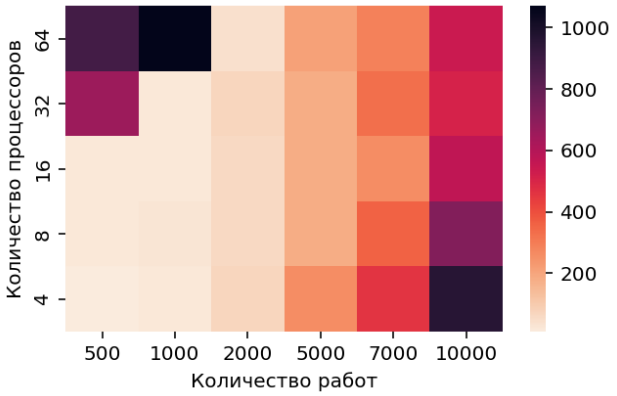
(a) Тепловая карта



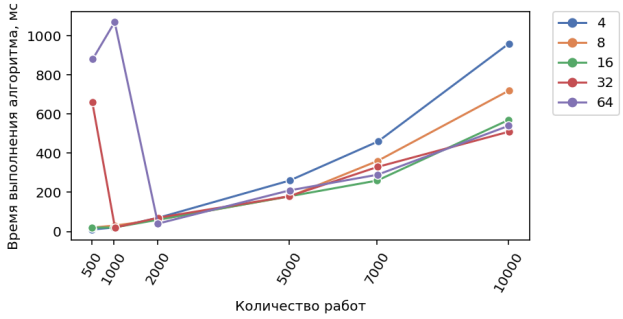
(b) Сводный график

Рис. 25: Время выполнения жадного алгоритма с EDF эвристикой на данных, основанных на слоистых данных, на постановке CR, в миллисекундах

Алгоритм работает в несколько раз быстрее жадного алгоритма с выбором по числу потомков.



(a) Тепловая карта



(b) Сводный график

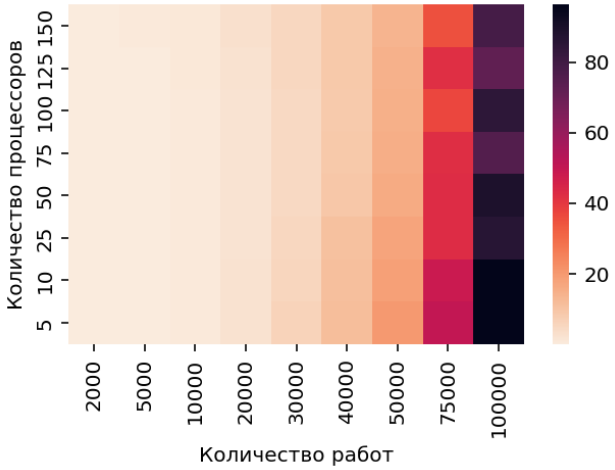
Рис. 26: Время выполнения жадного алгоритма с EDF эвристикой на данных, основанных на неоднородных процессорах, на постановке CR, в миллисекундах

На рисунках 26a и 26b показано время выполнения жадного алгоритма с EDF эвристикой, включая прогоны METIS, на слоистых данных с неоднородными процессорами. Как и для жадного алгоритма с выбором по числу потомков, на данных видно три 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 выбором по числу потомков.

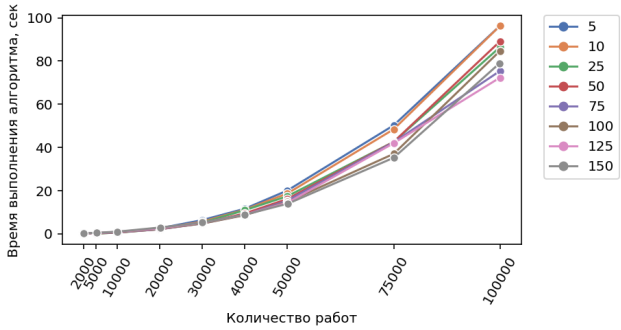
7.4.2.2 Постановка задачи без ограничений

На рисунках 27a и 27b показано время выполнения жадного алгоритма с EDF эвристикой, включая прогоны METIS, на данных с известным оптимумом.

На рисунках 28a и 28b показано время выполнения жадного алгоритма с EDF эвристикой, включая прогоны METIS, на данных, основанных на слоистых графах.

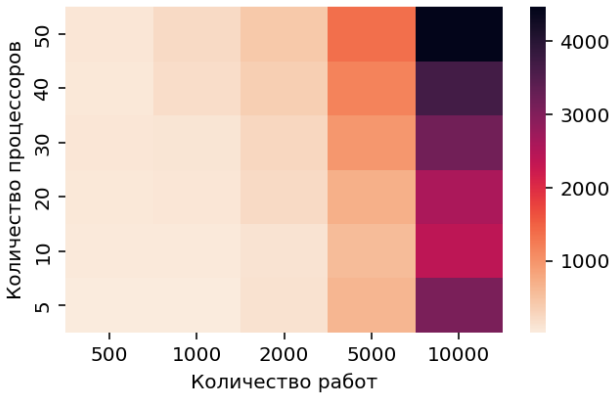


(a) Тепловая карта

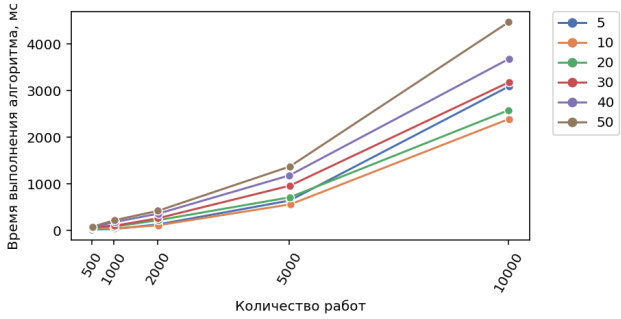


(b) Сводный график

Рис. 27: Время выполнения жадного алгоритма с EDF эвристикой на данных с известным оптимумом, на постановке NO, в секундах

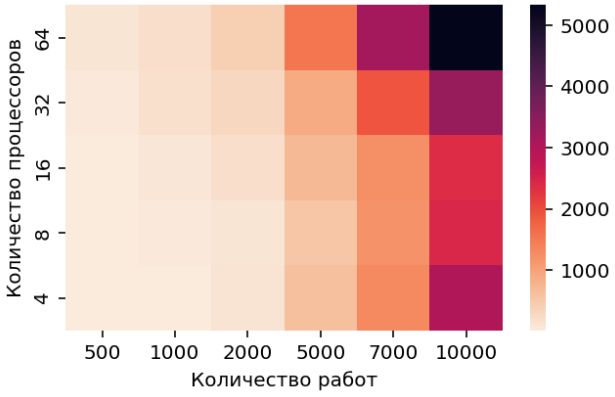


(a) Тепловая карта

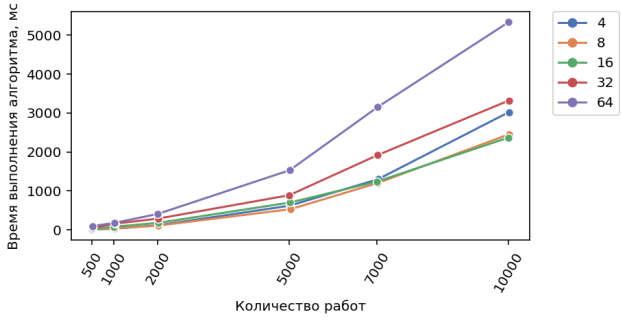


(b) Сводный график

Рис. 28: Время выполнения жадного алгоритма с EDF эвристикой на данных, основанных на слоистых данных, на постановке NO, в миллисекундах



(a) Тепловая карта



(b) Сводный график

Рис. 29: Время выполнения жадного алгоритма с EDF эвристикой на данных, основанных на неоднородных процессорах, на постановке NO, в миллисекундах

На рисунках 29a и 29b показано время выполнения жадного алгоритма с EDF эвристикой, включая прогоны METIS, на слоистых данных с неоднородными процессорами. Алгоритм выполняется быстрее жадного алгоритма с выбором по числу потомков, однако разница во времени выполнения незначительна. Время выполнения алгоритма увеличивается с увеличением количества работ и процессоров в системе.

7.5 Выводы из экспериментального исследования

На данных с известным оптимумом, для постановки с дополнительным ограничением CR жадный алгоритм с выбором по числу потомков достигает точности в 40%, а жадный алгоритм с EDF эвристикой - 5-10%. Однако, такое превосходство не наблюдается на наборах данных, основанных на слоистых данных и данных с неоднородными процессорами, где преимущество жадного алгоритма с EDF эвристикой над жадным алгоритмом с выбором по числу потомков не превышает 10-15%.

При постановке без дополнительных ограничений оба алгоритма быстро достигают точности в 5% на данных с известным оптимумом, а на данных, основанных на слоистых графах и данных, основанных на неоднородных процессорах жадный алгоритм с EDF эвристикой имеет преимущество над жадным алгоритмом с выбором по числу потомков, достигающее 15-20%.

Предпочтительным является жадный алгоритм с EDF эвристиками, т.к. при схожей или превосходящей точности он выполняется быстрее.

8 Заключение

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

1. Проведен аналитический обзор алгоритмов построения списочных расписаний с целью выявления детерминированных алгоритмов, которые возможно модифицировать под поставленную задачу и имеют хорошую возможность масштабирования, по результатам которого были выбраны жадные алгоритмы.

2. Разработаны и реализованы алгоритмы, основанные на различных жадных критериях.

3. Проведено исследование свойств алгоритмов, которое показало низкую вычислительную сложность и среднее отклонение от оптимума в 30% для жадного алгоритма с выбором по числу потомков и до 5-10% для жадного алгоритма с фиктивными директивными сроками.

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

9 Список литературы

1. Held M., Karp R. M. A dynamic programming approach to sequencing problems //Journal of the Society for Industrial and Applied mathematics. – 1962. – V. 10. – №. 1. – P. 196-210.

2. Coffman E. G. Computer and job-shop scheduling theory. — Nashville, TN : John Wiley & Sons, 02.1976. — ISBN 0471163198.

3. Шахбазян К. В., Тушкина Т. А. Обзор методов составления расписаний для многопроцессорных систем //Записки научных семинаров ПОМИ. – 1975. – Т. 54. – №. 0. – С. 229-258.

4. Magirou V. F., Milis J. Z. An algorithm for the multiprocessor assignment problem //Operations research letters. – 1989. – V. 8. – №. 6. – P. 351-356.

5. Cormen T. H. et al. Introduction to algorithms. – MIT press, 2022.

6. Калашников А. В. Алгоритмы оптимизации расписаний, основанные на исправлении неоптимальных фрагментов. — 2004.

7. Rahman M. Branch and Bound Algorithm for Multiprocessor Scheduling. – 2009 - URL: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A518609&dswid=-4391.

8. Karypis G. METIS and ParMETIS // Encyclopedia of Parallel Computing / edited by D. Padua. — Boston, MA : Springer US, 2011. — P. 1117—1124. — ISBN 978-0-387-09766-4. — DOI: 10.1007/978 − 0 − 387 − 09766 − 4_500.

9. 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.

10. Canon L. C., Sayah M. E., Héam P. C. A comparison of random task graph generation methods for scheduling problems //Euro-Par 2019: Parallel Processing: 25th International Conference on Parallel and Distributed Computing, Göttingen, Germany, August 26–30, 2019, Proceedings 25. – Springer International Publishing, 2019. – P. 61-73.

11. Boost C++ libraries. — URL: https://www.boost.org/ (дата обр. 02.04.2023).

12. JSON parsing library. — URL: https://github.com/ nlohmann / json (дата обр. 02.04.2023).

13. METIS library. — URL: https://github.com/KarypisLab/METIS (дата обр. 10.04.2023).

14. multiprocessor-scheduling. — URL: https://github.com/ipsavitsky/greedy-scheduling (дата обр. 23.04.2022).

15. TOML parsing library. — URL: https://github.com/ToruNiina/toml11 (дата обр. 02.04.2023).