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, 2022

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

Классическая задача построения расписания хорошо изучена и досканально описана в [1]. Поскольку данная задача принадлежит к классу NP-трудных, неизвестен алгоритм, который за полиномиальное время даст точный ответ, но существуют алгоритмы, которые дают приближенные результаты. Большинство таких алгоритмов резделяются на два класса: *конструктивные* и *итерационные*. Из основных примеров можно выделить:

- Конструктивные алгоритмы
    1. Алгоритмы, основанные на поиске максимального потока в сети
    2. Алгоритмы, основанные на методах динамического программирования
    3. Алгоритмы, 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 этой работы

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

Для достижения указанной цели требуется:

1. Математически сформулировать поставленную задачу.
2. Провести обзор алгоритмов построения списочных расписаний с целью выявления жадных критериев которые могут быть модифицированы для решения данной задачи.
3. Разработать и реализовать алгоритмы.
4. Провести экспериментальное исследование свойств алгоритмов.

3 Постановка задачи

1. Ориентированный граф потока данных G без циклов, в котором дуги - зависимости по данным, а вершины - задания. Вершин n, дуг m

2. Вычислительная система, состоящая из p однородных, или неоднородных процессоров процессоров. Неоднородные процессоры имеют кратную производительность, то есть для любой работы k длительности l на процессоре A, на процессоре B будет иметь длительность tl, где t константно для B по отношению к A.

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 пеердачи данных между процессорами i и j.

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

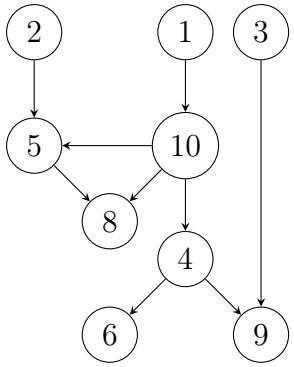


Рис. 1: Граф G потока данных

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

Пусть в оптимальном расписании работы 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 представлениях:

1. Графическое представление.

   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.

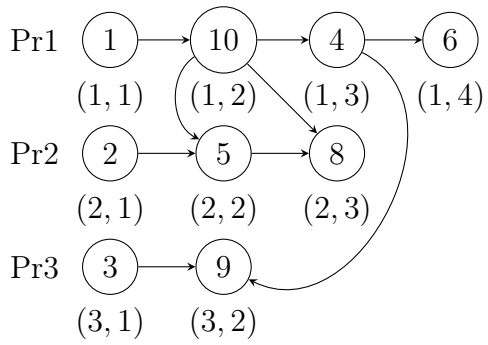


Рис. 2: Графическое представление расписания

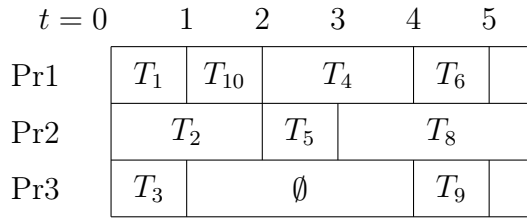


Рис. 3: Представление расписания в виде временной диаграммы

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.

Доказано, что эти представления полностью эквивалентны и, имея одно, возможно построить другое. В предложенном решении расписание строится в виде временной диаграммы.

1. Построить расписание HP, то есть для i-й работы определить время начала ее выполнения s_i и процессор p_i на которм 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 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 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. Далее эта постановка будет упоминаться как **постановка** CR

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

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. Точность решений, строимых рассмотренными алгоритмов

3. Порядок сложности алгоритма, насколько он масштабируемый.

4. На данных какой размерности протестирован алгоритм. Время его работы.

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 [2]. 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 заадчи 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 [3]) 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 [4].

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 неадекватны для решения проблемы.

Алгоритм, обсуждаемый в [5], 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 [6]. Этот конкретный алгоритм подходит только для задач, допускающих прерывания в вычислительной системе, поэтому в данной задаче его нельзя использовать.

4.3 Итерационные алгоритмы

4.3.1 Генетические алгоритмы

Такие алгоритмы используют селекцию, кроссинговер и мутацию, чтобы оптимизировать набор решений, называющийся популяцией, при этом сохраняя среди них достаточное разнообразие чтобы не находить решения в локальных минимумах.

Недостатком этого типа алгоритмов является отсутствие масштабируемости. По мере увеличения использования становится все труднее управлять системой или масштабировать ее для различных задач. Результаты исследования в статье демонстрируют, что алгоритм не имеет высокой производительности при использовании больших объемов данных. Разобранный в [7] алгоритм смог обработать задачу, содержащую 1000 заданий, за 1,5 часа на процессоре с частотой 2 ГГц. Хотя это может показаться длительным временем выполнения, это в значительной степени связано с тем, что генетические и эволюционные алгоритмы требуют наличия множества решений, доступных в популяциях для реализации.

4.3.2 Алгоритм имитации отжига

В 1983 году был представлен алгоритм имитации отжига, который является отличным способом нахождения приближенных решений для сложных NP-сложных задач комбинаторной оптимизации. Этот алгоритм позволяет решать проблемы с помощью стохастических шагов, гарантируя получение наилучшего решения [3].

Имитация отжига может эффективно обрабатывать большие наборы данных, так как этот алгоритм работает с одним решением. В отличие от генетических и эволюционных алгоритмов, это делает его более масштабируемым.

4.3.3 Алгоритм муравьиных колоний

Муравьиные алгоритмы - это методы оптимизации, которые используют положительную и отрицательную обратную связь для поиска оптимального пути [8]. Они находят широкое применение в различных задачах оптимизации, но могут столкнуться с проблемой преждевременной сходимости, что может привести к неоптимальным результатам. Важным аспектом применения муравьиных алгоритмов является тщательная настройка и учет особенностей конкретной задачи, чтобы достичь наилучшего результата.

4.4 Выводы из обзора предметной области

Название алгоритма	Рандомизированность	Класс алгоритма	Возможность масштабирования
Генетические алгоритмы	Рандомный	Итерационный	+/-
Алгоритм имитации отжига	Рандомный	Итерационный	+
Муравьиные алгоритмы	Рандомный	Итерационный	-
Жадные стратегии	Детерминированный	Конструктивный	+
Динамическое программирование	Детерминированный	Конструктивный	-
Ветви и границы	Детерминированный	Конструктивный	-
Максимальный поток	Детерминированный	Конструктивный	-

Таблица 1: Существующие алгоритмы

В результате обзора предметной области, под критерии масштабируемости и соответствия задаче подходит жадный алгоритм. По результатам не было найлено работ, точно

соответствующим постановке задачи, поэтому предложенный подход требуется модифицировать.

5 Алгоритм построения расписания

5.1 Алгоритмы построения расписания

5.1.1 Общая схема жадных алгоритмов

Жадные алгортмы, 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

5.1.2 Жадный алгоритм

Жадный 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 постановки задачи в раписание, в случае невозможности постановки без нарушения дополнительного ограничения. В таком случае, алгоритм завершается.

Задача d_i наывается **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 выбирает задачу для постановки по следующему критерию: пусть $Succ(d)$ - функция, определяющая количество непосредственных последователей работы в графе. Тогда $\forall d_j \in D, d_j \neq d_i : Succ(d_j) < Succ(d_i)$.

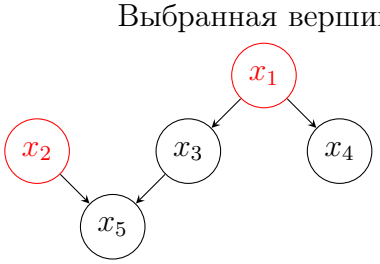


Рис. 4

На рисунке 4 $D = (x_1, x_2)$, 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 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 детерминирован. 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 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 "саммый ранний директивный срок первый" (earliest deadline first, или **EDF**) упорядочивает работы по возрастанию директивных сроков и выбирает работу с нименьшим 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. $d_A \coloneqq d - p$

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 изветсного 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 рассчета 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 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 разбиении METIS, самые медленные процессору будут самыми загруженными. Задача данного алгоритма - "разгрузить"самые загруженные процессоры путем перемещания раот с него на менее загруженные процессоры.

Алгоритм имет 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 следующее условие: 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 подграфы METIS [9].

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 проуессоры равный, 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 вреднему количеству работ в подграфе. Другими словами, `ufactor` позволяет контролировать дизбаланс в количестве вершин в кластере. 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 кждое 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 интревал времени (простой процессора или после завершения последней поставленной задачи), начинающийся в t и длительностью, больший или равный времени выполнения работы, в который не выолняется ни одназ работа. В некоторых случаях, задача будет поставлена до начала другой, не свзяанной 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 5.

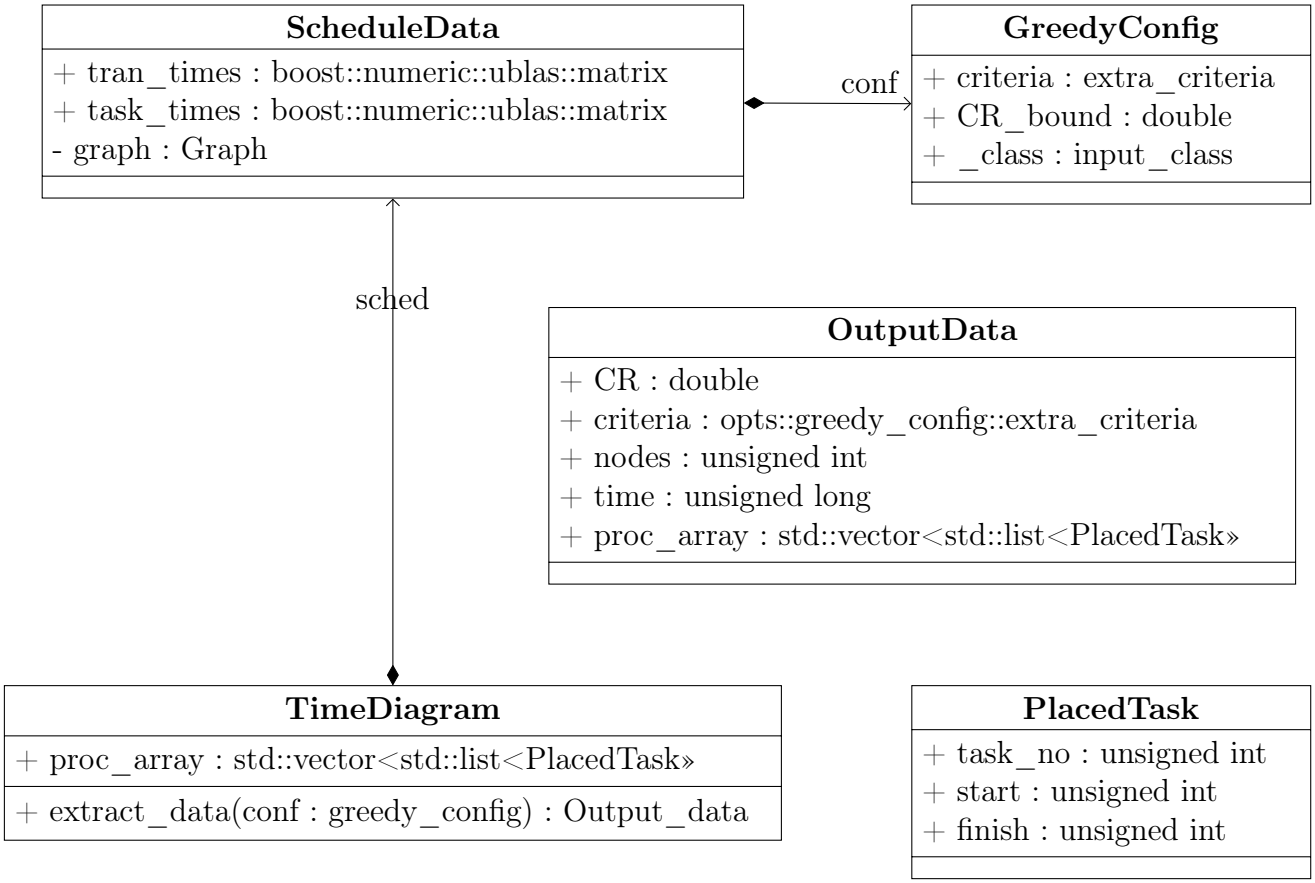


Рис. 5: UML-диаграмма реализации

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.
- `greedy_EDF_heuristic()` - Жадный алгоритм с EDF эвристикой.

В репозиторий включены следубщие библиотеки:

1. `METIS` 5.1.0 [11] - библиотека для разбиения графов.
2. `json` 3.11.2 [12] - библиотека для работы с форматом JSON. Используется для составления выходных файлов.

3. `toml11` 3.7.1 [13] - библиотека для работы с форматом TOML. Используется для чтения конфигурационных файлов.

Также, у реализации есть зависимость, не включенная в репозиторий - `boost` 1.80 [14]. 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 собираетяс 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

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.

```
[general]
criteria = "BF"
CR_bound = 0.4
inp_class = "class_1"

[greedy]
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 5 и таблице 4. Конфигурационный 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 постанленных на соответствующий процессор работами. Каждая поставленная работа состоит из: • `task_dur` - время выполнения работы на поставленный процессор. • `task_no` - идентификатор работы. • `task_start` - время начала выполнения работы на процессоре.

Таблица 4: 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 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 добавленно 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 исзвестного оптимума, но с неоднородными процессорами. Используется для постановки задачи без лополнительных ограничений

Схема генерации слоистых графов описана в [15].

7.2 Экспериментальный стенд

Эксперименты были проведены на машине, обладающей следующими характеристиками:

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

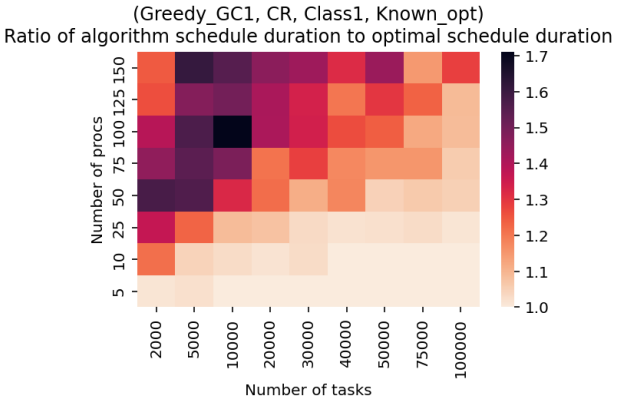
7.3 Исследование качества решений

7.3.1 Жадный алгоритм

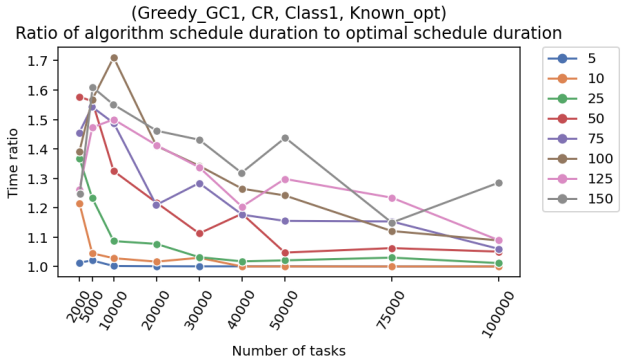
7.3.1.1 Постановка задачи с ограничением на количество передач

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

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

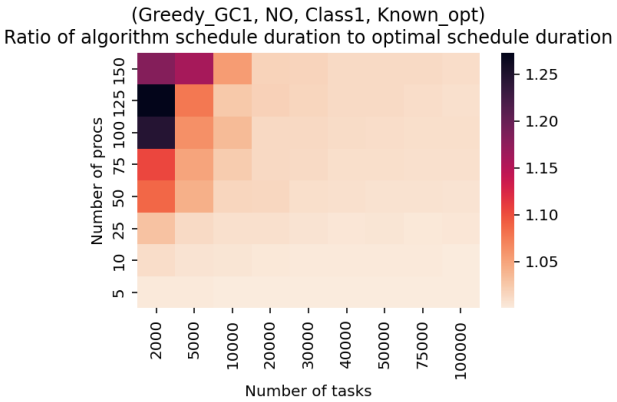


(a) Тепловая карта

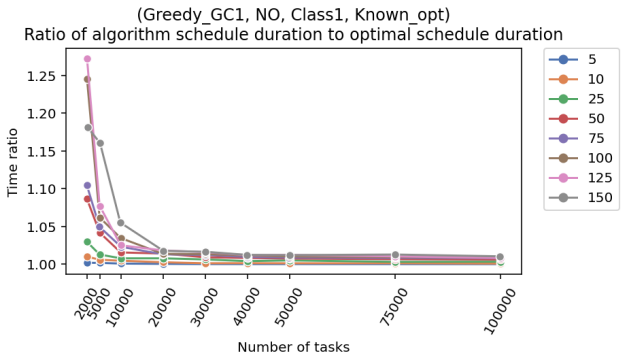


(b) Сводный график

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



(a) Тепловая карта



(b) Сводный график

Рис. 7: Отношение времени выполнения расписания к оптимальному времени выполнения

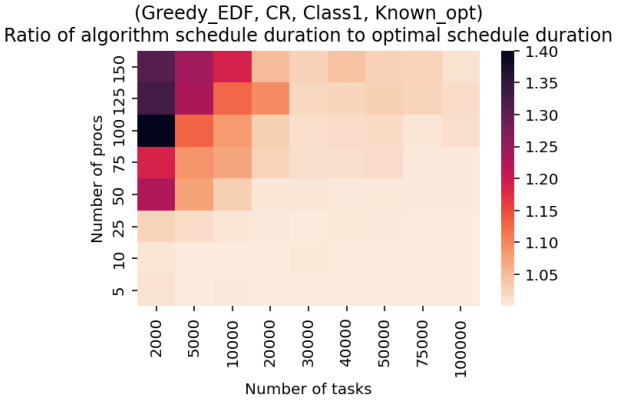
7.3.1.2 Постановка задачи без ограничений

На рисунках 7a и 7b показано качество решенией, генерируемых жадных алгоритмом с жадными критерием на данных с известным оптимумом. Цветом на рисунке 7a и значением на оси Oy на рисунке 7b показано отношение длительности расписания, построенного алгоритмом к длительности оптимального расписания. Значения всегда больше 1, чем меньше, тем лучше.

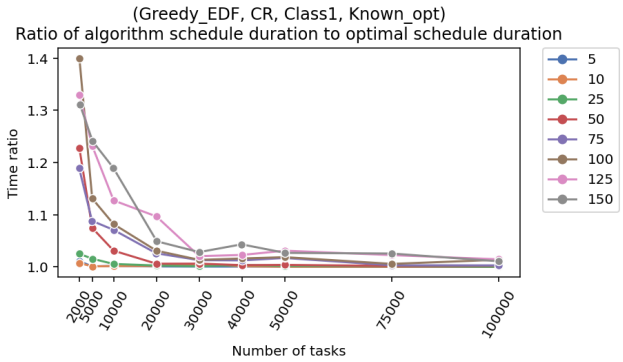
Точность алгоритма возрастает с увеличением количества работ и понижается с увеличением количества процессоров.

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. 8: Отношение времени выполнения расписания к оптимальному времени выполнения

На рисунках 8a и 8b показано качество решенией, генерируемых жадным алгоритмом с жадными критерием на данных с известным оптимумом. Цветом на рисунке 8a и значением на оси Oy на рисунке 8b показано отношение длительности расписания, построенного алгоритмом к длительности оптимального расписания. Значения всегда больше 1, чем меньше, тем лучше.

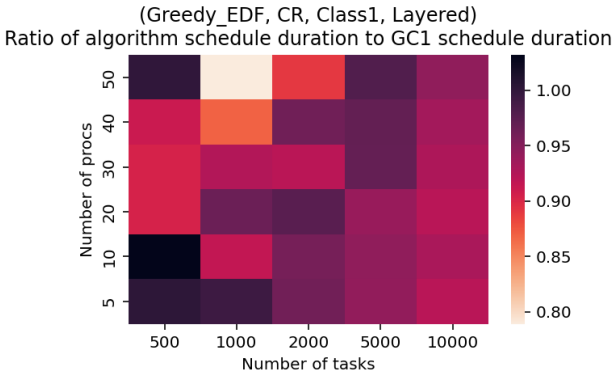
Точность алгоритма повышается с увеличением количества работ, однако ухудшение решения с увеличением количества процессоров в системе менее значительно, чем в жадном алгоритме.

На рисунках 9a и 9b показано качество решений, генерируемых жадным алгоритмом с EDF эвристикой на данных, построенных на слоистых графах. Цветом на рисунке 9a и значением на оси Oy на рисунке 9b показано отношение длительности расписании расписании, построенного жадным алгоритмом с жадным кртерием.

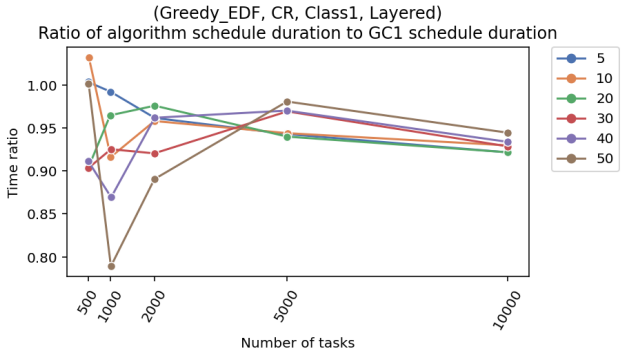
В большинстве случаев, качество решений жадного алгоритма с EDF эвристикой лучше качества решений жадного лагоритма, однако преимущество остается в пределах 10%, кроме двух выбросов в районе 1000 работ.

На рисунках 10a и 10b показано качество решений, генерируемых жадным алгоритмом с EDF эвристикой на данных, построенных на слоистых графах. Цветом на рисунке 10a и значением на оси Oy на рисунке 10b показано отношение длительности расписании расписании, построенного жадным алгоритмом с жадным кртерием.

Алгоритм не дает значимых улучшений решения по сравнению с жадным алгоритмом, кроме случая с 500 работами на 64 процессорах, в котром решение, строимое жадным

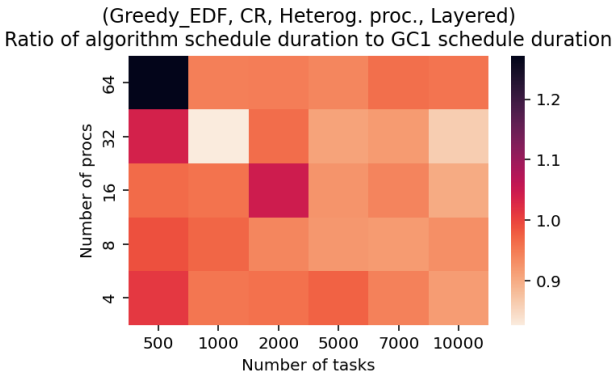


(a) Тепловая карта

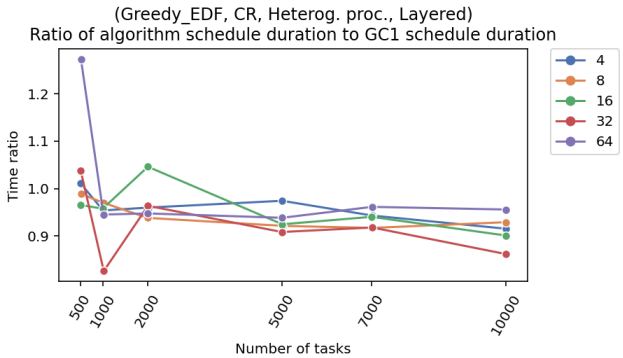


(b) Сводный график

Рис. 9: Отношение времени выполнения расписания к времени выполнения расписания, построенного при помощи жадного алгоритма



(a) Тепловая карта

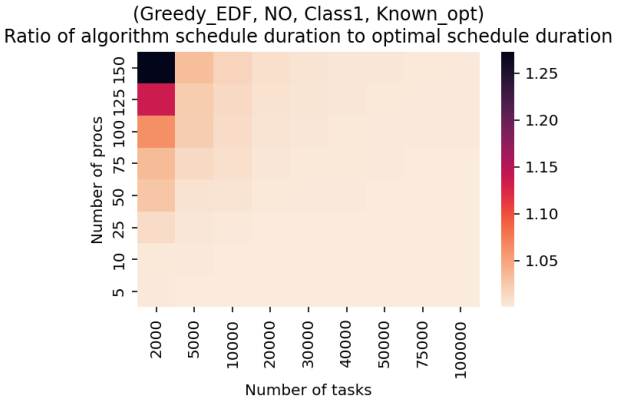


(b) Сводный график

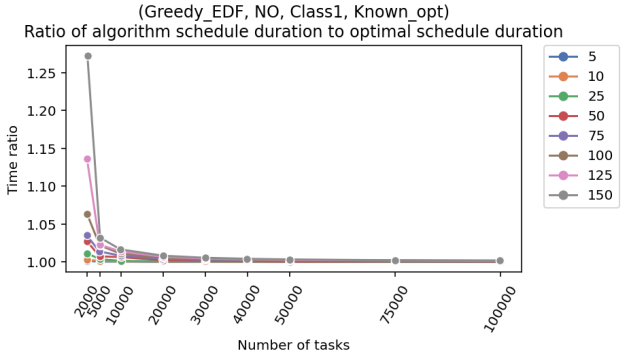
Рис. 10: Отношение времени выполнения расписания к времени выполнения расписания, построенного при помощи жадного алгоритма

алгоритмом с EDF эвристикой значительно хуже решения, построенного жадным алгоритмом.

7.3.2.2 Постановка задачи без ограничений



(a) Тепловая карта

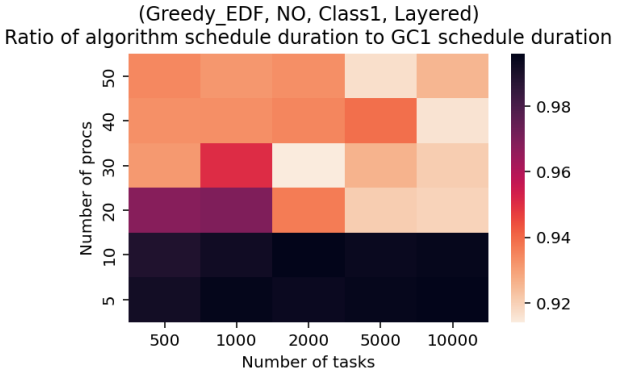


(b) Сводный график

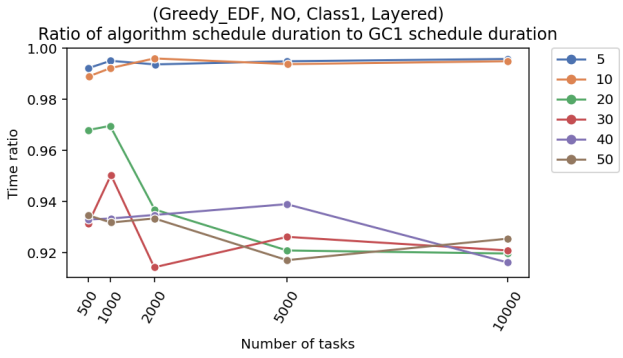
Рис. 11: Отношение времени выполнения расписания к оптимальному времени выполнения

На рисунках 11a и 11b показано качество решений, генерируемых жадным алгоритмом с EDF эвристикой на данных с известным оптимумом. Цветом на рисунке 11a и значением на оси Oy на рисунке 11b показано отношение длительности расписания, построенного жадным алгоритмом с жадным кртерием. 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.



(a) Тепловая карта



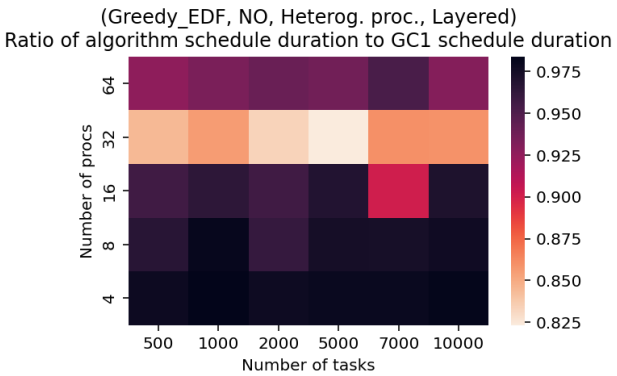
(b) Сводный график

Рис. 12: Отношение времени выполнения расписания к оптимальному времени выполнения

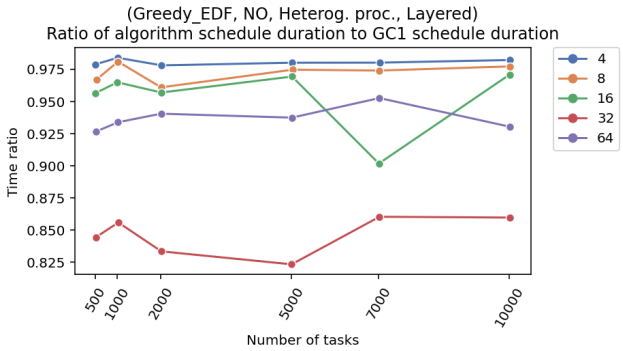
На рисунках 12a и 12b показано качество решений, генерируемых жадным алгоритмом с EDF эвристикой на данных, построенных на слоистых графах. Цветом на рисунке 12a и значением на оси Oy на рисунке 12b показано отношение длительности расписания, построенного жадным алгоритмом с жадным кртерием.

Во всех случаях, качество решений выше качетва решений жадного алгоритма, но это улучшение не превышает 10%.

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



(a) Тепловая карта



(b) Сводный график

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

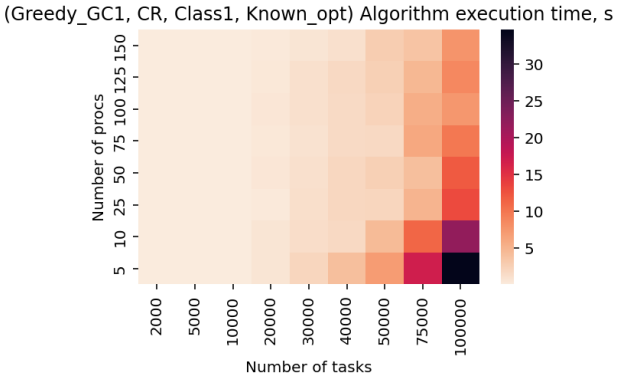
и значением на оси Oy на рисунке 13b показано отношение длительности расписании расписании, построенного жадным алгоритмом с жадным кртерием. Значения всегда больше 1, чем меньше, тем лучше.

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

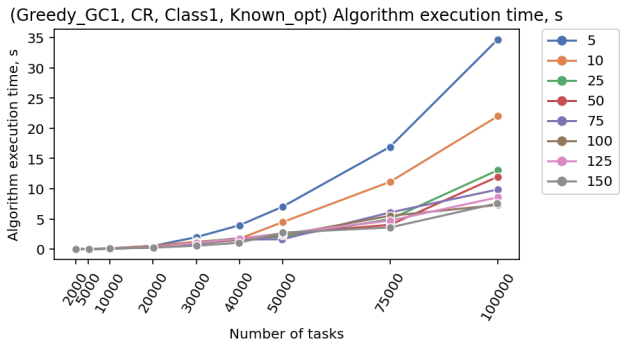
7.4 Исследование временной сложности алгоритма

7.4.1 Жадный алгоритм

7.4.1.1 Постановка задачи с ограничением на количество передач



(a) Тепловая карта

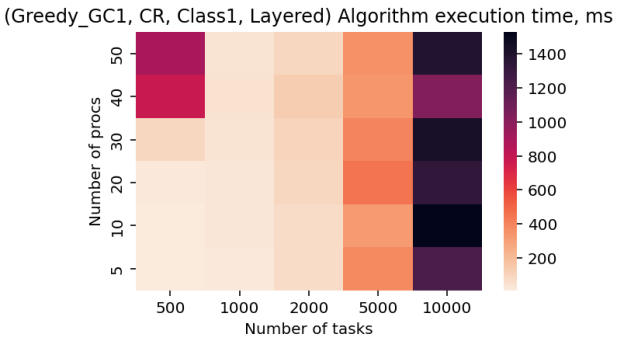


(b) Сводный график

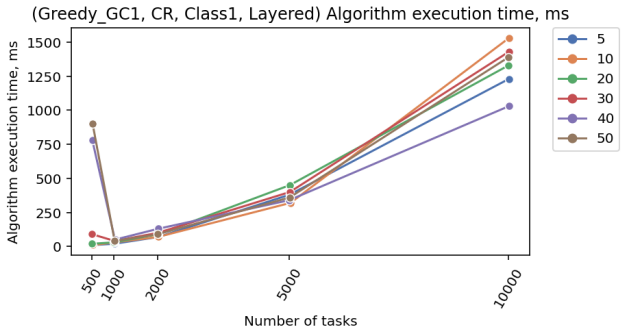
Рис. 14: Время выполнения алгоритма, в секундах

На рисунках 14a и 14b показано время выполнения жадного алгоритма, 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 15a и 15b показано время выполнения алгоритма на наборе данных, основанных на слоистых графах.



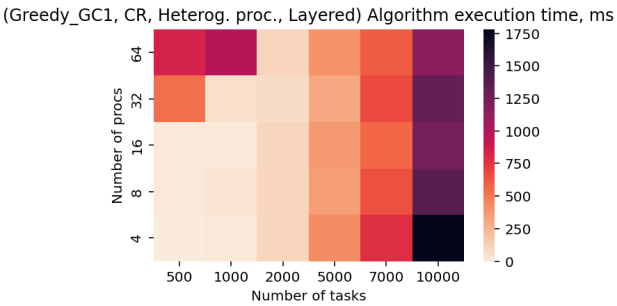
(a) Тепловая карта



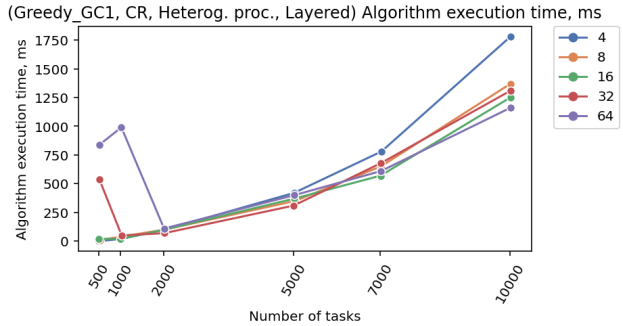
(b) Сводный график

Рис. 15: Время выполнения алгоритма, в миллисекундах

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. 16: Время выполнения алгоритма, в миллисекундах

На рисунках 16a и 16b 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.

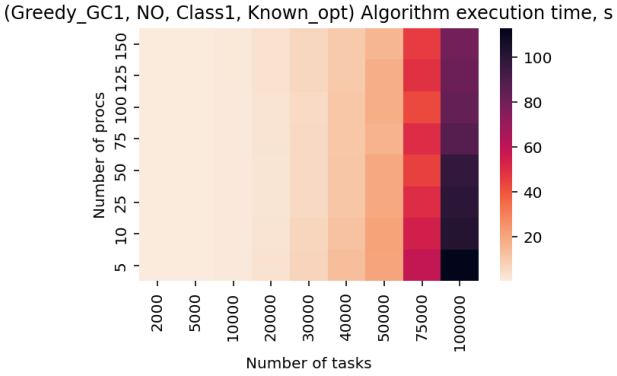
7.4.1.2 Постановка задачи без ограничений

На рисунках 17a и 17b показано время выполнения алгоритма на наборе данных с известным оптимумом. Время выполнения растет с увеличением количества задач.

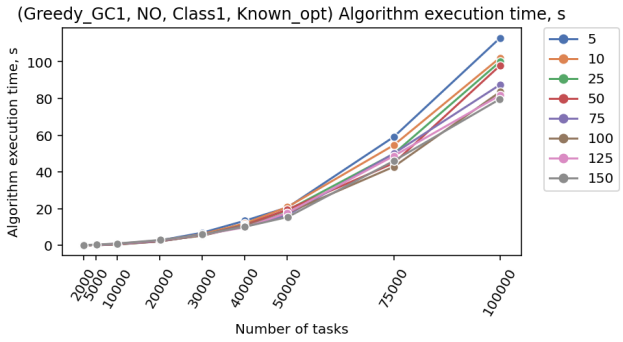
При одинаковом количестве работ, быстрее выполняется алгоритм с большим количеством процессоров, по тем же причинами, что и в постановке с дополнительным ограничением CR.

На рисунках 17a и 17b показано время выполнения алгоритма на наборе данных, основанных на слоистых графах. Время выполнения растет с увеличением количества работ и количества процессоров.

На рисунках 19a и 19b показано время выполнения алгоритма на наборе данных с неоднородными процессорами. Время выполнения растет с увеличением количества работ и количества процессоров.

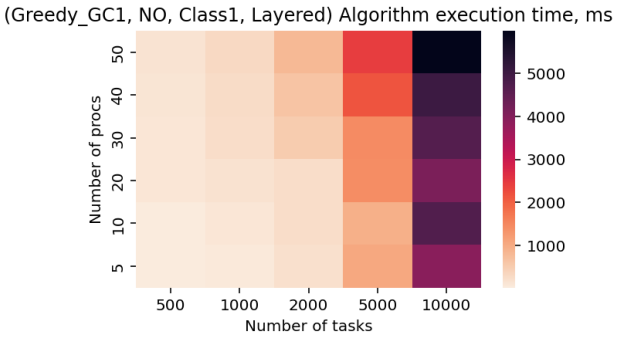


(a) Тепловая карта

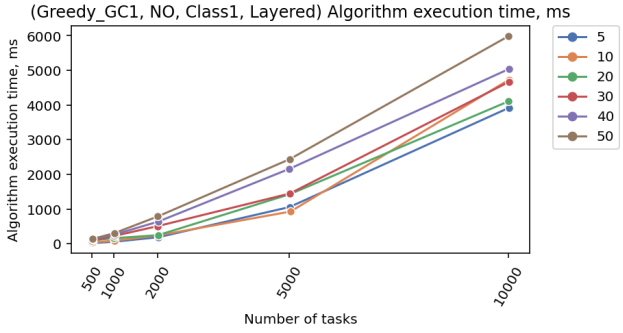


(b) Сводный график

Рис. 17: Время выполнения алгоритма, в секундах

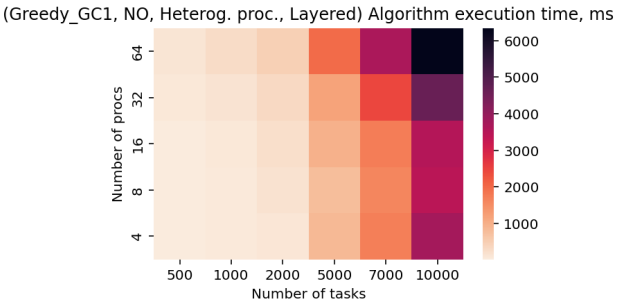


(a) Тепловая карта

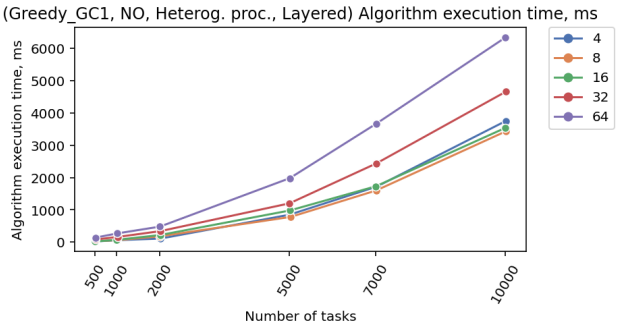


(b) Сводный график

Рис. 18: Время выполнения алгоритма, в миллисекундах



(a) Тепловая карта

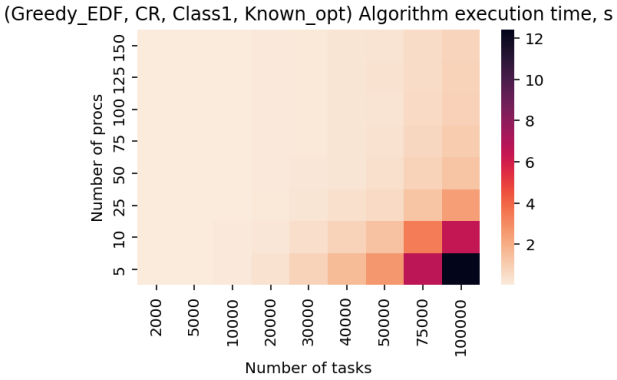


(b) Сводный график

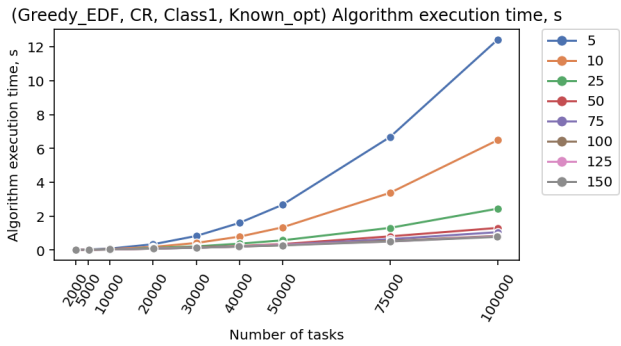
Рис. 19: Время выполнения алгоритма, в миллисекундах

7.4.2 Жадный алгоритм с EDF эвристикой

7.4.2.1 Постановка задачи с ограничением на количество передач



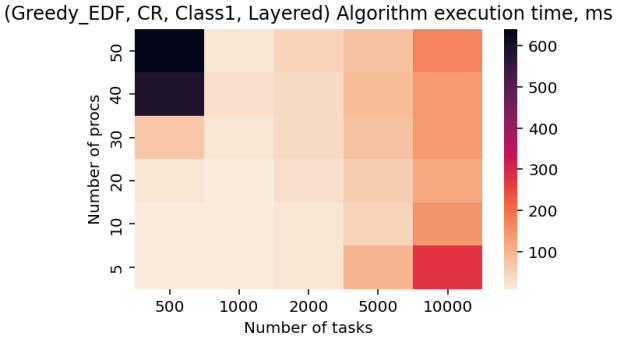
(a) Тепловая карта



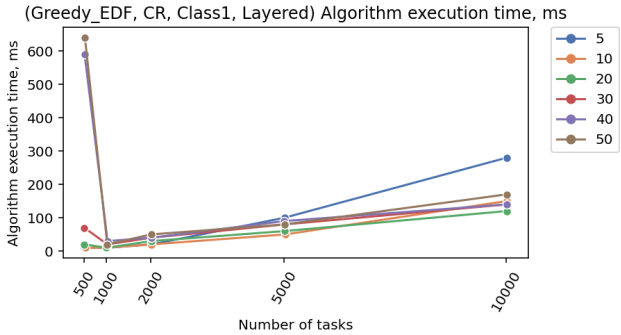
(b) Сводный график

Рис. 20: Время выполнения алгоритма, в секундах

На рисунках 20a и 20b 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. При равном количестве работ, выше время выполнения при меньшем количестве процессоров. Причина схожа с причинами подобного явления для жадного алгоритма, поскольку они разделяют одну процедуру поиска нового места в расписании.



(a) Тепловая карта

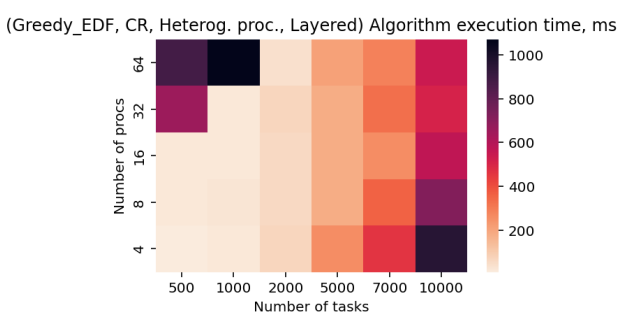


(b) Сводный график

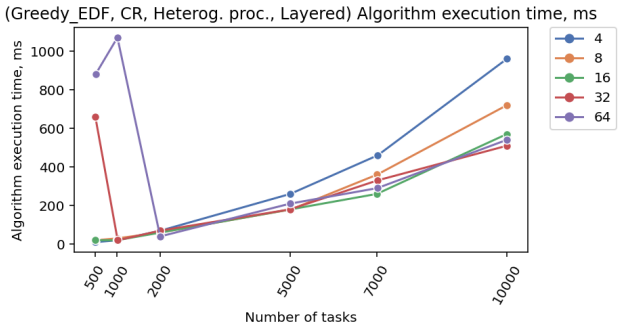
Рис. 21: Время выполнения алгоритма, в миллисекундах

На рисунках 21a и 21b 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.

На рисунках 22a и 22b показано время выполнения жадного алгоритма с EDF эвристикой, включая прогоны METIS, на слоистых данных с неоднородными процессорами. Как и для жадного алгоритма с жадной эвристикой, 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



(a) Тепловая карта

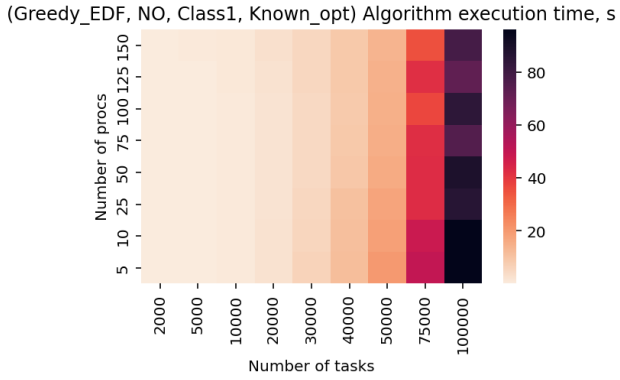


(b) Сводный график

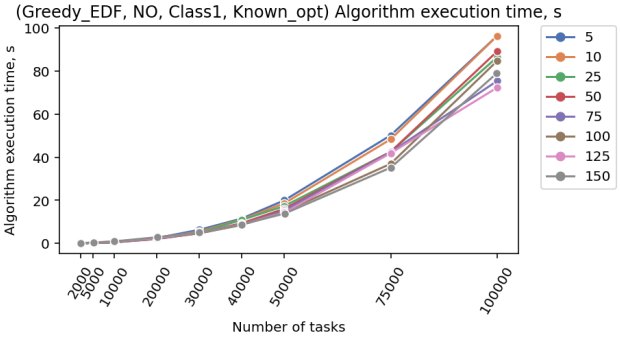
Рис. 22: Время выполнения алгоритма, в миллисекундах

количеством процессоров в системе и временем, затраченным на построение расписания. Алгоритм работает в несколько раз быстрее жадного алгоритма.

7.4.2.2 Постановка задачи без ограничений



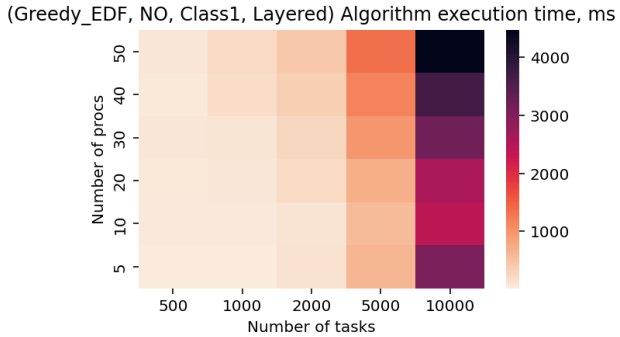
(a) Тепловая карта



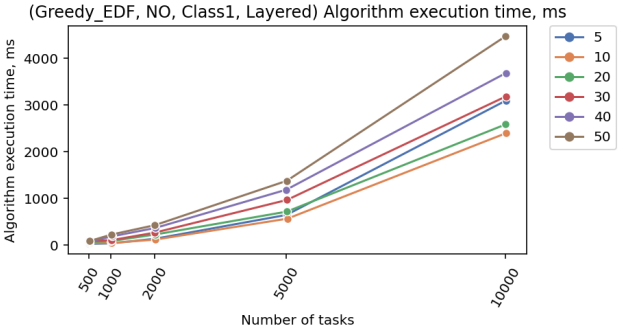
(b) Сводный график

Рис. 23: Время выполнения алгоритма, в секундах

На рисунках 23a и 23b показано время выполнения жадного алгоритма с EDF эвристикой, включая прогоны METIS, на данных с известным оптимумом.



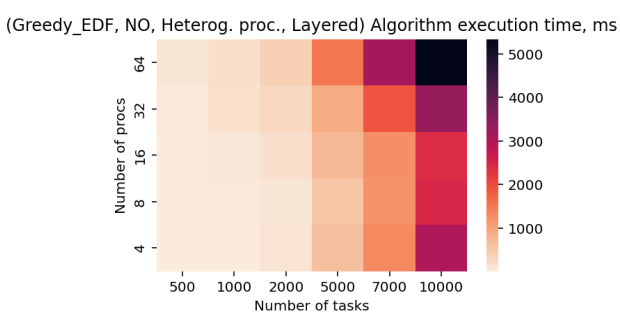
(a) Тепловая карта



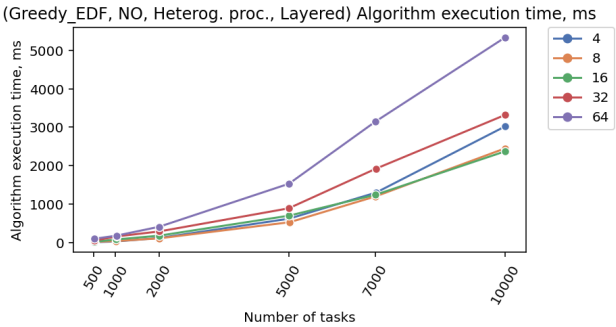
(b) Сводный график

Рис. 24: Время выполнения алгоритма, в миллисекундах

На рисунках 24a и 24b показано время выполнения жадного алгоритма с EDF эвристикой, включая прогоны METIS, на данных, основанных на слоистых графах.



(a) Тепловая карта



(b) Сводный график

Рис. 25: Время выполнения алгоритма, в миллисекундах

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

7.5 Выводы из экспериментального исследования

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.

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

Список литературы

1. *Coffman E. G.* Computer and job-shop scheduling theory. — Nashville, TN : John Wiley & Sons, 02.1976. — ISBN 0471163198.
2. Introduction to Algorithms / T. H. Cormen [и др.]. — 2nd. — The MIT Press, 2001. — ISBN 0262032937.
3. *Калашников А. В.* Алгоритмы оптимизации расписаний, основанные на исправлении неоптимальных фрагментов. — 2004.
4. *Held M., Karp R. M.* A Dynamic Programming Approach to Sequencing Problems // Journal of the Society for Industrial and Applied Mathematics. — 1962. — Т. 10, № 1. — С. 196—210. — DOI: 10.1137/0110015. — eprint: https://doi.org/10.1137/0110015. — URL: https://doi.org/10.1137/0110015.
5. *Rahman M.* Branch and Bound Algorithm for Multiprocessor Scheduling //. — 2009.
6. *Magirou V., Milis J.* An algorithm for the multiprocessor assignment problem // Operations Research Letters. — 1989. — Т. 8, № 6. — С. 351—356. — ISSN 0167-6377. — DOI: https://doi.org/10.1016/0167-6377(89)90022-9. — URL: https://www.sciencedirect.com/science/article/pii/0167637789900229.
7. *Sheikh H. F., Ahmad I., Fan D.* An Evolutionary Technique for Performance-Energy-Temperature Optimized Scheduling of Parallel Tasks on Multi-Core Processors // IEEE Transactions on Parallel and Distributed Systems. — 2016. — Т. 27. — С. 668—681.
8. *Штовба С. Д.* Муравьиные алгоритмы // Exponenta Pro. Математика в приложениях. — 2003. — № 4. — С. 70—75.
9. *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.
10. *Savitsky I.* multiprocessor-scheduling. — URL: https://github.com/ipsavitsky/greedy-scheduling (дата обр. 23.05.2022).
11. METIS library. — URL: https://github.com/KarypisLab/METIS (дата обр. 10.04.2023).
12. JSON parsing library. — URL: https://github.com/nlohmann/json (дата обр. 02.04.2023).
13. TOML parsing library. — URL: https://github.com/ToruNiina/toml11 (дата обр. 02.04.2023).
14. Boost C++ libraries. — URL: https://www.boost.org/ (дата обр. 02.04.2023).
15. *Canon L.-C., Sayah M., Héam P.-C.* A Comparison of Random Task Graph Generation Methods for Scheduling Problems. — 02.2019.

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

	Примеры входных данных			
Критерии	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