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

Савицкий Илья Павлович

Жадные алгоритмы для построения многопроцессорного списочного расписания

Дипломная работа

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

Москва, 2022

Аннотация

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

Содержание

1 Введение

Классическая задача построения расписания хорошо изучена и досканально описана в [3]. Поскольку данная задача принадлежит к классу NP-трудных, не существует алгоритма, который за полиномиальное время даст точный ответ, но существуют алгоритмы, которые дают приближенные результаты. Большинство таких алгоритмов резделяются на две категории: *конструктивные* и *итерационные*. Из основных примеров можно выделить (большинство из них упомянуты в [12]):

- Конструктивные алгоритмы
    1. Алгоритмы, основанные на поиске максимального потока в сети
    2. Алгоритмы, основанные на методах динамического программирования
    3. Алгоитмы, основанные на методе ветвей и границ
    4. Жадные алгоритмы
    5. Жадные алгоритмы с процедурой ограниченного перебора
- Итерационные алгоритмы
    1. Генетические алгоритмы
    2. Дифференциальная эволюция
    3. Алгоритм имитации отжига
    4. Алгоритм муравьиных колоний

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

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

2 Цели и задачи курсовой работы

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

Для достижения указанной цели требуется:

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

3 Постановка задачи

3.1 Дано

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

2. Вычислительная система, состоящая из p различных процессоров

3. Матрица C_{ij} длительности выполнения работ на процессорах, $i = 1 \dots n, j = 1 \dots p$. Каждая строка этой матрицы - длины выполнения n-й задачи на p процессорах.

4. Матрица D_{kl} передач данных между процессорами, $k = 1 \dots p, l = 1 \dots p, D_{kk} = 0$. D_{ij}-й элемент этой матрицы - время пеердачи данных между процессорами i и j.

3.2 Определение расписания

Расписание программы определено, если заданы:

1. Множества процессоров и работ

2. Привязка - всюду определенная на множестве работ функция, которая задает распределение работ по процессорам

3. Порядок - заданные ограничения на последовательность выполнения работ и является отношением частичного порядка, удовлетворяющим условиям ацикличности и транзитивности. Отношение порядка на множестве работ, распределенных на один процессор, является отношением полного порядка.

3.2.1 Способы представления расписаний

Пусть дан следующий граф потока данных:

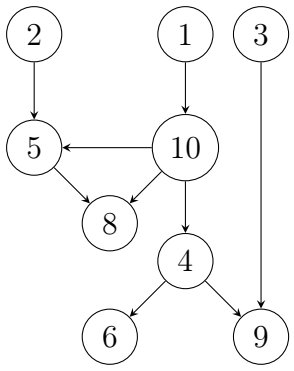

Рис. 1: Граф G потока данных

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

1. Графическая форма представления

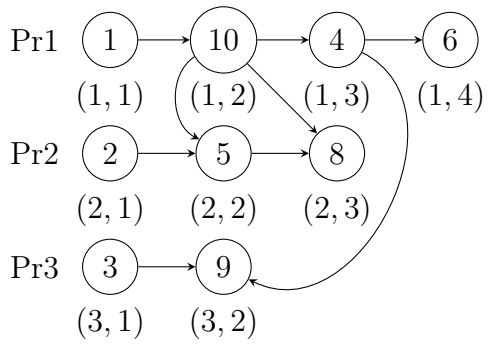

Рис. 2: Графическое представление расписания

В такой форме представления 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.

2. Временная диаграмма

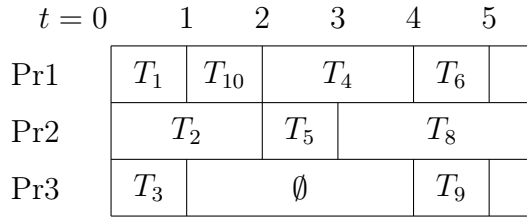

Рис. 3: Представление расписания в виде временной диаграммы

В такой форме представления расписания каждой задаче сопоставляется пара из номера процессора и временис старта задачи на процессоре.

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

3.3 Требуется

1. Построить расписание HP, то есть для i-й работы определить время начала ее выполнения s_i и процессор p_i на 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 сиартом работы-получателя должен быть интервал времени, не меньший чем задержка на передачу данных между ними (с учетом привязки работ к процессорам и маршрута передачи данных).

3.4 Различные постановки задачи

1. Задача с однородными процессорами (длительность выполнения работы не зависит от того, на каком процессоре она выполняется) и дополнительными ограничениями на количество передач:

   - $CR = \frac{m_{ip}}{m} < 0.4$, где m_{ip} - количество передач данных между работами на каждый процессор. (Cut ratio)

2. Задача с неоднородными процессорами, но без дополнительных ограничений на расписание

4 Обзор предметной области

4.1 Критерии обзора

Ниже приведены критерии, по которым будут рассматриваться и сравниваться алгоритмы

1. Насколько сильно рассматриваемая в статье задача отличается от решаемой. Можно ли взять алгоритм, описанный в статье за базу для алгоритма для решения данной задачи.

2. Порядок сложности алгоритма.

3. На данных какой размерности протестирован алгоритм.

4.2 Конструктивные алгоритмы

4.2.1 Жадные алгоритмы

Жадные алгоритмы подразумевают декомпозицию задачи на ряд более простых подзадач. На каждом шаге решение принимается исходя из принципа получения оптимального решения для очередной подзадачи. То есть, на каждом шаге алгоритм делает выбор, оптимальный с точки зрения получения решения очередной подзадачи, предполагая, что эти локально-оптимальные решения приведут к приемлемому решению задачи. Какие-либо жадные стратегии, гарантированно получающие оптимальное расписание, на настоящий момент времени неизвестны, за исключением небольшого числа вариантов задач составления расписаний не принадлежащих к классу NP-полных. Например, известен жадный алгоритм, получающий точное решение для задачи обслуживания одним процессором максимального числа работ из заданного набора работ с фиксированными сроками начала и окончания [5]. Набор локальных критериев оптимизации сильно зависит от класса архитектуры. Для архитектур, в которых возможно 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.

4.2.2 Структкра жадных алгоритмов построения многопроцессорного расписания.

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 [11]) единственным образом определяется из порядка работ на процессоре. Следовательно, имеет роль порядок, в котором работы добавляются в расписание.

Таким образом, для построения расписания достаточно определить:

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

4.2.3 Другие конструктивные алгоритмы

(Написать про [1])

4.3 Итерационные алгоритмы

Итерационные алгоритмы работают путем создания аппроксимированного варианта решения и последующего его улучшения. Однако, большинство из них рандомизированные и, следовательно, из нескольких различных запусков теоретически возможно получить различные расписания (несмотря на то то они все сходятся). Более того, многие из таких алгоритмов плохо масштабируемы. Муравьиные алгоритмы разобраны в [13], имитации отжига - в [7].

Таблица 1: Существующие алгоритмы

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

Обзоры этих алгоритмов для схожих задач представлены в [3; 4; 10]

- Достоинства:
  1. Поскольку большинство из итерационных методов - рандомизированные алгоритмы, они меньше зависят от подбора параметров.
  2. Некоторые из итерационных алгоритмов, например, муравьиные алгоритмы, могут адаптироваться к изменению начальных условий, что не релевантно в рассматриваемой постановке задачи.
- Недостатки:
  1. Некоторые из итерационных алгоритмов могут быть плохо масштабируемы.
  2. Многие из итерационных алгоритмов требуют генерации начальных состояний расписаний, от которых они могут сильно зависеть.

5 Алгоритм построения расписания

5.1 Дополнительные обозначения

1. $D = (d_1, d_2, \ldots, d_l)$, где l - количество вершин, доступных для добавления(т.е. у которых нет предшественников в графе G) - множество вершин, доступных для добавления в расписание.

2. (s_i, p_i) - пара, состоящая из номера процессора p_i и время старта задачи s_i, то есть достаточное количество информации для размещения работы в расписании.

5.1.1 Жадные критерии

1. $GC1$ - критерий, используемый в выборе работы на постановку

2. $GC2$ - критерий, используемый в выборе места постановки работы в расписание

5.2 Жадный алгоритм

1. Сформировать множество вершин, у которых нет предшественников. Множество $D = (d_1, d_2 \ldots d_i)$ где d_i – номер работы, доступной для добавления в расписание (т.е. у которой нет предшественников в исходном графе).

2. В случае, если дополнительная дополнительное ограничение CR - запустить алгоритм разбиения графа METIS.

3. По жадному критерию $GC1$ выбирается работа из множества D для размещения в расписании. Пусть выбранная работа – d_i.

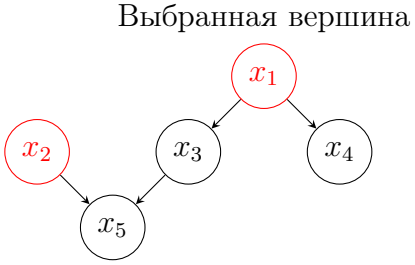

Рис. 4: Выбор вершины в соответствии с жадным критерием $GC1$

4. Производится пробное размещение работы d_i. В случае задачи с дополнительным ограничение CR, задача ставится на процессор, которое определяет разбиение из пункта 1. В случае постановки задачи без дополнительных ограничений, задача ставится в соответствии с жадным критерием $GC2$.

5. d_i удаляется из списка размещенных работ и в графе G удаляется соответствующая вершина и все дуги, исходящие из нее.

6. Обновляется множество D. Если D не пустое, то алгоритм переходит на пункт 3.

5.2.1 Жадный критерий выбора работы $GC1$

- Максимальное количество потомков у работы

Такой выбор работы позволяет открыть максимально возможное количество кандидатов на следующую постановку задачи в расписание, а значит с минимальной вероятностью закрывает путь к оптимальному решению.

5.2.2 Жадный критерий выбора места работы в расписании $GC2$

- Среди доступных процессоров выбирается тот, при постановки на который время завершения работы на процессоре будет минимальным.

5.2.3 Расчет времени начала работы

При постановке задачи d_i на процессор p_i:

1. Рассматривается множество всех ее предшественников $Pr = (pr_1, pr_2, \ldots, pr_i)$
2. Вычисляются времена завершения pr_i на распределенный процессор.
3. Пусть pr_i распределена на процессор p_k. Если p_k не равна p_i, к времени завершения работы pr_i добавляется время передачи между p_k и p_i.
4. После чего из получившегося набора берется максимум - "зависимость по задачам"
5. После чего рассматриваются все пробелы (участки временной диаграммы между завршением i-й работы и началом $i + 1$-й) на p_i, находящиеся после "зависимости по задачам и, если длительность какого-либо пробела больше или равна времени выполнения d_i на p_i - задача помещается в пробел.
6. Если данная процедура не нашла пробел - задача помещается в конец.

5.3 Жадный алгоритм, опирающийся на фиктивные директивные сроки

1. В случае дополнительного ограничения CR - строится взвешенное разбиение при помощи METIS.
2. Листовые вершины добавляются в множество D.
3. Для каждой вершины из DL вычисляется ее фиктивный директивный срок. Первые листовые вершины, добавленные в пункте 2 получают директивные сроки 0. Остальные вершины получают директивные сроки, равные разности минимального директивного срока среди потомков задачи и времени выполнения задачи. В случае дополнительного ограничения CR также вычитается время передачи с процессора, на который распределен потом задачи и на который распределена задача.
4. Задачи ставятся в расписание в порядке возрастания директивных сроков. В случае дополнительного ограничения CR процессор выбирается из распределения, полученного в пункте 1. Для постановки задачи NO Процессор выбирается с учетом дополнительного ограничения $GC2$ (пункт 5.2.2). Постановка происходит алгоритмом, описанным в 5.2.3

6 Программная реализация алгоритма

Код реализации выложен в репозитории [8] В программной реализации были использованы следующие библиотеки:

1. `boost` 1.81 [2]
2. `json` 3.11.2 [6]
3. `toml11` 3.7.1 [9]

Проект обладает следующей структурой:

1. `greedy/greedy_algo.cpp`
2. `greedy/schedule.cpp`
3. `greedy/time_schedule.cpp`
4. `graph_part.cpp`
5. `input_class.cpp`
6. `json_dumper.cpp`
7. `logger_config.cpp`
8. `options.cpp`
9. `parser.cpp`

Для сборки проекта используется `CMake`.

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

Для сборки документации (на английском) используется `Doxygen`.

```
doxygen Doxyfile
```

7 Экспериментальное исследование алгоритма

7.1 Цели и методика эксперименального исследования

Цель экспериментального исследования - проверить эффективность разработаного алгоритма. Для этого алгоритм будет протестирован на нескольких наборах входных данных, в которых число работ будет варьироваться от 120 до 420, число процессоров - от 4 до 8. Классы графов коррелируют с классами, данными в задаче Хуавей (Приложение 1).

7.2 Подбор параметоров алгоритма

7.3 Исследование эффективности алгоритма

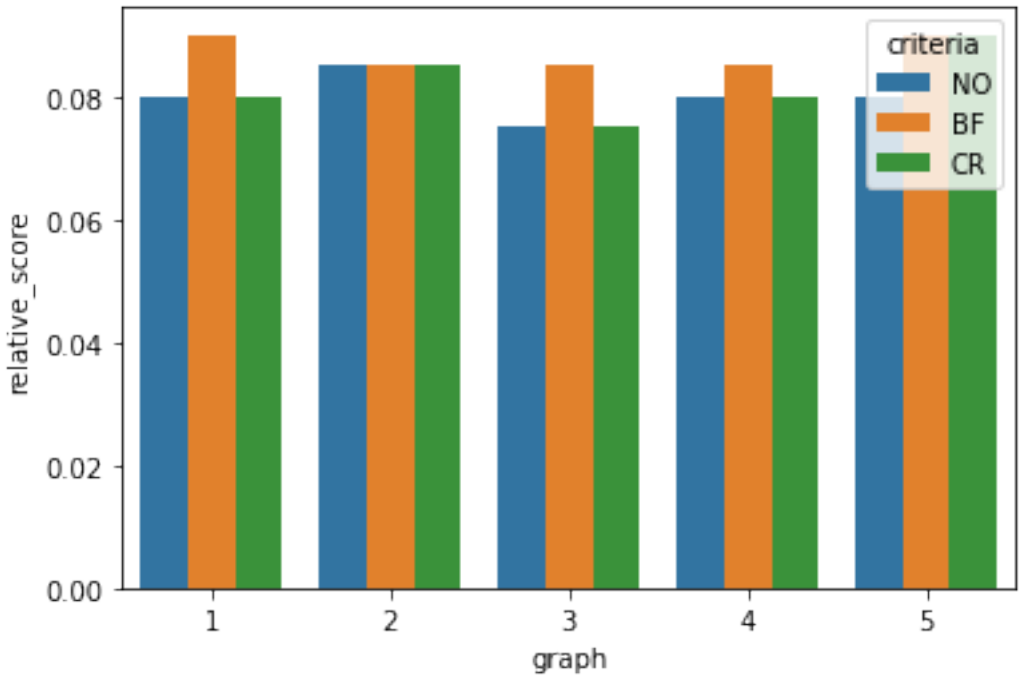

Рис. 5: Распределение точности полученного расписания относительно оптимума. Меньше - лучше.

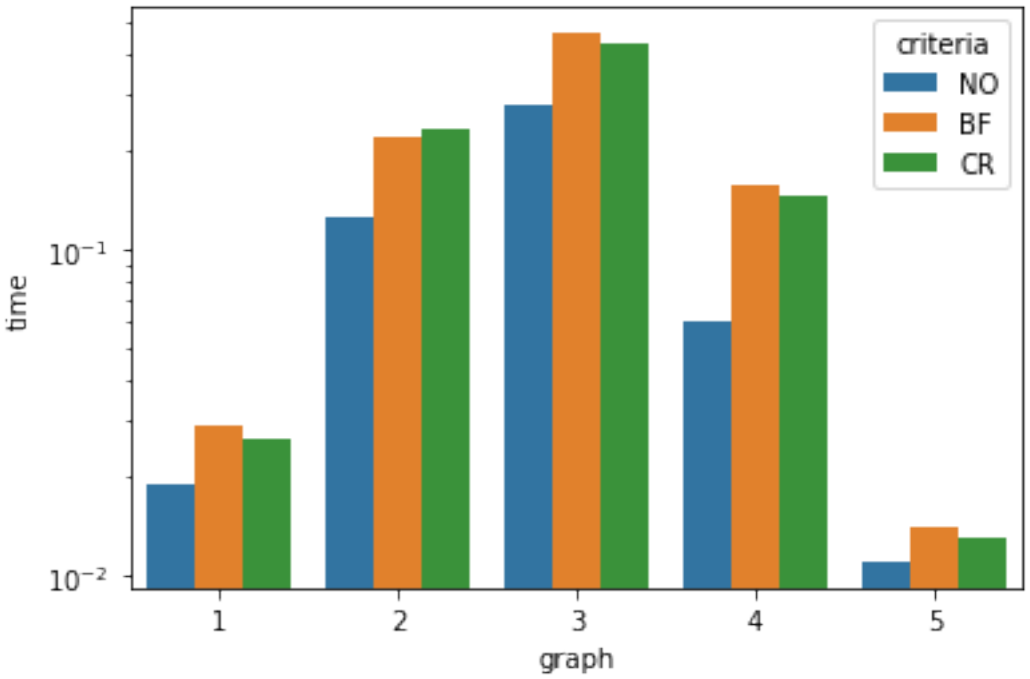

Рис. 6: Время выполнения алгоритма на каждом из тестовых графов. Меньше - лучше.

Алгортм бы испытан на пяти наборах входных данных:

1. 126 вершин, 4 процессора, 716 передач данных
2. 417 вершин, 8 процессоров, 2367 передач данных
3. 408 вершин, 8 процессоров, 8763 передач данных
4. 396 вершин, 8 процессоров, 395 передачи данных
5. 93 вершины, 4 процессора, 92 передачи данных

На рисунке 5 представлена относительная точность *score* алгоритма при расчете на каждом из тестовых графов, расчитанная по формуле:

$$score = \frac{\text{время полученного расписания}}{\text{время оптимального расписания}} - 1$$

Дополнительные критерии, упомянутые в 3.4, предствлены на графиках разными цветами.

Из графика видно, что с различными дополнительными критериями алгоритм дает сравнимые времена расписаний. Причина этого - тема для дальнейшего исследования.

На рисунке 6 предоставлено время работы алгоритма для каждого из тестовых графов. Из графа видно, что время работы линейно зависит от количества передач данных между процессорами и количества работ.

В ходе исследования подтвержден факт того, что более 90% работ размещаются в расписании при помощи жадного критерия, упомянутый в [12]

8 Заключение

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

1. Проведен обзор существующих решений задач. Произведено сравнений других стратегий с жадными критериями и ограниченным перебором. На основе обзора произведен выбор подхода, основанного на комбинации жадных стратегий и ограниченного перебора.

2. Разработан и реализован алгоритм.

3. Произведено исследование свойств алгоритма, которое показало что алгоритм генерирует расписание, превосходящее оптимальное на 8%. Более 90% задач размещены при помощи жадной стратегии.

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

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

Список литературы

1. *Akbari M.*, *Rashidi H.* AN EFFICIENT ALGORITHM FOR COMPILE-TIME TASK SCHEDULING PROBLEM ON HETEROGENEOUS COMPUTING SYSTEMS. — 2015. — Янв. — DOI: 10.7813/2075-4124.2015/7-1/A.45.
2. Boost C++ libraries. — URL: https://www.boost.org/ (дата обр. 02.04.2023).
3. *Coffman E. G.* Computer and job-shop scheduling theory. — Nashville, TN : John Wiley & Sons, 02.1976. — ISBN 0471163198.
4. *Davis R. I.*, *Burns A.* A Survey of Hard Real-Time Scheduling for Multiprocessor Systems // ACM Computing Surveys. — 2011. — Окт. — Т. 43, № 4.
5. Introduction to Algorithms / T. H. Cormen [и др.]. — 2nd. — The MIT Press, 2001. — ISBN 0262032937.
6. JSON parsing library. — URL: https://github.com/nlohmann/json (дата обр. 02.04.2023).
7. *Rzadca K. S. F.* Heterogeneous Multiprocessor Scheduling with Differential Evolution // IEEE Congress on Evolutionary Computation. — 2005. — Т. 3.
8. *Savitsky I.* multiprocessor-scheduling. — URL: https://github.com/ipsavitsky/multiprocessor-scheduling/tree/main/code (дата обр. 23.05.2022).
9. TOML parsing library. — URL: https://github.com/ToruNiina/toml11 (дата обр. 02.04.2023).
10. *К.В. Шахбазян Т. Т.* Обзор методов составления расписаний для многопроцессорных систем // Журнал советской математики. — 1981. — Т. 15, № 5. — С. 651—669.
11. *Калашников А. В.* Алгоритмы оптимизации расписаний, основанные на исправлении неоптимальных фрагментов. — 2004.
12. *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.
13. *Штовба С. Д.* Муравьиные алгоритмы // Exponenta Pro. Математика в приложениях. — 2003. — № 4. — С. 70—75.

ПРИЛОЖЕНИЕ 1

Классы входных данных

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

1. Первый класс (примеры DAG_A и DAG_B) характеризуется относительно небольшим масштабом графа работ, небольшим числом процессоров, полнотой графа связности процессоров и одинаковыми задержками между любыми двумя процессорами.

2. Второй класс (примеры DAG_C и DAG_D) характеризуется относительно большим масштабом графа работ, большим числом процессоров

Таблица 2: Сравнение примеров из классов данных

	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