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

Савицкий Илья Павлович

Построение многопроцессорного расписания с использованием жадных стратегий и ограниченного перебора

Курсовая 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 приводится один из возможных вариантов 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 [1]. 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 [7]):

- 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 алгоритма построения многопроцессорного расписания с дополниьельными ограничениями на основе комбинации жадных стратегий и ограниченного перебора.

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 алгоритм.

3. Реализовать алгоритм.

4. Провести исследование свойств алгоритма.

3 Постановка задачи

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.

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.

Способы представления расписаний

Пусть дан следующий граф потока данных:

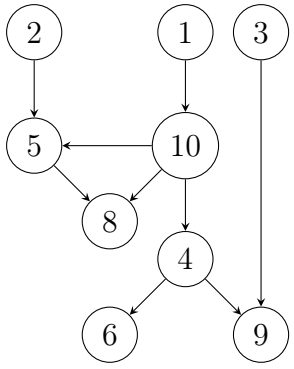


Рис. 1: Граф G потока данных

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

1. Графическая форма представления

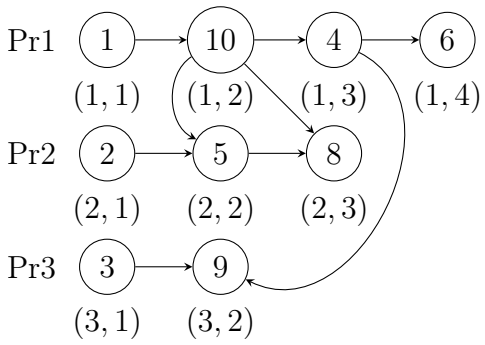


Рис. 2: Графическое представление расписания

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

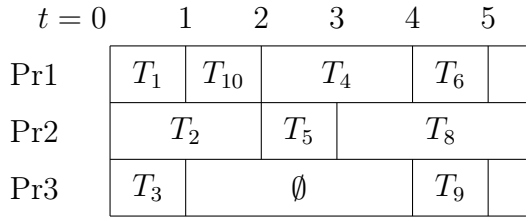


Рис. 3: Представление расписания в виде временной диаграммы

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.

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).

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
   - $CR2 = \frac{m_{2edg}}{m} < 0.05$, где m_{2edg} - количество дуг, начальный и конечный узлы которых назначены на процессоры, не соединенных напрямую

2. Задача с однородными процессорами и дополнительным ограничением сбалансированности распределения работ:

   - $BF = \lceil 100 \cdot \left(\frac{a_{max} \cdot p}{n} - 1\right) \rceil < 10$, где a_{max} - наибольшее, по всем процессорам, количество работ на процессоре

3. 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 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 [3]. 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.

4.3 Жадные алгоритмы с процедурой ограниченного перебора

Жадные алгоритмы в чистом виде применяются редко из-за того, что в результате их работы часто получаются решения очень далекие от оптимальных, так как они склонны застревать в локальных минимумах оптимизируемой функции. Для того чтобы исправить этот недостаток был предложен подход [7], сочетающий жадные стратегии и ограниченный перебор.

В основу предлагаемых в работе алгоритмов, сочетающих жадные стратегии и ограниченный перебор, положены следующие основные принципы:

- на каждом шаге работы алгоритма делается локально-оптимальный выбор в соответствии с используемыми жадными критериями,

- на каждом шаге производится проверка того, что «жадный выбор не закрывает пути к оптимальному решению»,
- вызов процедуры ограниченного перебора, если проверка условия «жадный выбор не закрывает пути к оптимальному решению» дала отрицательный результат.

Предложенный метод построения алгоритмов сочетающих жадные стратегии и ограниченный перебор дает возможность задавать требуемый баланс между точностью и сложностью алгоритма путем выбора значения максимально допустимой глубины перебора. Метод допускает настройку на частную задачу путем подбора жадных критериев.

В разобранном в статье [7] алгоритме рассматривается классический вариант задачи построения многопроцессорного расписания без учета дополнительных критериев, однако эти критерии легко учитываются как дополнительные условия при проверке оптимальности текущего расписания, построенного жадным алгоритмом.

Этот алгоритм детерминирован и имеет несколько параметров, один из которых – глубина перебора в процедуре ограниченного перебора. При установке малой глубины перебора можно создать неточное, но верное начальное приближение расписания для других алгоритмов, например для генетических алгоритмов, имитации отжига или дифференциальной эволюции. Притом, алгоритм конструктивен, нет потребности задания критерия останова алгоритма.

В краевом случае, когда все задачи распределяются в соответствии с жадными критериями и процедура ограниченного перебора не вызывается алгоритм имеет сложность $O(nlogn)$.

- Достоинства:
  1. Возможность балансировать между точностью и сложностью благодаря изменению глубины перебора.
  2. Не зависит от начальных условий, т.к. алгоритм детерминированный.
  3. Возможность получить расписание для поддерева работ.
- Недостатки:
  1. Необходимо подбирать жадные критерии в ходе эксперимента.
  2. Время работы алгоритма может быть большим, если процедура ограниченного перебора будет вызываться слишком часто.

4.4 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 [8], имитации отжига - в [4].

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 и ограниченный перебор	Рандомный	Конструктивный	+

Обзоры этих алгоритмов для схожих задач представлены в [1; 2; 6]

- 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. k - вектор длин критических путей от "головной" вершины до каждой вершины графа.

3. (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.

Жадные критерии

1. $GR1$ - критерий, используемый в выборе работы на постановку

2. $GR2$ - критерий, используемый в выборе места постановки работы в расписание

Процедуры ограниченного перебора

1. $H1$ - процедура перебора для создания места для постановки работы. Минимизируемый критерий - s_i

2. $H2$ - процедура перебора для приближения времени старта работы к длине критического пути до нее. Минимизируемый критерий - $\sum s_k$, где s_k - времена начала k последних добавленных работ.

5.2 Содержательное описание алгоритма

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 в множестве D одна вершина – обозначим ее за d, в противном случае – создадим фиктивную вершину с нулевой длительностью, у которой все потомки будут из множества D, и обозначим ее за d

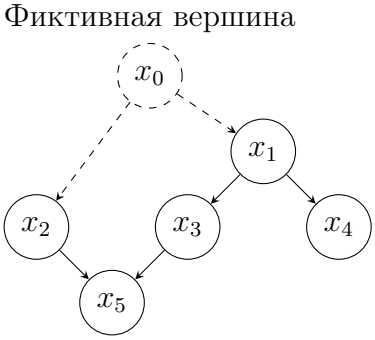


Рис. 4: Добавление фиктивной вершины

3. Зададим вектор k – вектор длин критических путей до вершин от d. При помощи алгоритма Дейкстры этот вектор заполняется значениями k_i, где i – номер вершины. Поскольку алгоритм Дейкстры работает со взвешенными графами, каждое ребро получает вес минимального времени работы на вычислительной системе вершины, из которой исходит

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

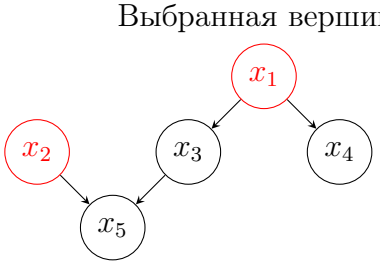


Рис. 5: Выбор вершины в соответствии с жадным критерием $GC1$

5. Производится пробное размещение работы d в расписании с учетом жадного критерия $GC2$ и дополнительных ограничений. В случае, если не получилось найти подходящее место для работы – запускается процедура ограниченного перебора $H1$. Становится известно s – время старта работы и p – процессор, на котором работа выполняется.

6. Если s_i больше длины критического пути (с точностью до Δ, где Δ – параметр алгоритма), то вызывается процедура ограниченного перебора $H2$. Если работу разместить не удалось – завершить алгоритм. Если s_i не превосходит длину критического пути (с точностью Δ), то работа размещается в расписании.

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

8. Обновляется множество D. Если D не пустое, то алгоритм переходит на пункт 4.

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.

Жадный критерий выбора места работы в расписании $GC2$

- Скорейшее завершение частично построенного расписания

Выбор места работы в расписании

- Взвешенная сумма.

$$crit = C_1 \cdot GC2 + C_2 \cdot CR + C_3 \cdot CR2$$

  где C_1, C_2 и C_3 - параметры алгоритма. Работа размещается на место с наибольшим значением параметра $crit$.

- Допускная система выбора

  1. Список мест размещения работ ранжируется по $GC2$, после чего отсекаются верхние $n\%$ работ, где n - параметр алгоритма
  2. Такие же действия повторяются для каждого дополнительного ограничения
  3. В конечном списке выбрать место по жадному критерию

При ранжировании задач по взвешенной сумме возможна ситуация, при которой будет выбрано место которое хорошо проходит по одному критерию, но очень плохо по другому и тогда расписание будет строиться в направлении ложного минимума. Поэтому в программной реализации было выбрано использование допускной системы выбора.

Ограниченный перебор

После неудачной пробной постановки работы в расписание алгоритм создает набор $K = (k_1, k_2, \ldots, k_t)$, состоящий из t последних добавленных работ (t – параметр алгоритма). Далее, процедурой полного перебора пробуются различные расписания до тех пор, пока не получится расписание, удовлетворяющее критерию критичности пути до последней поставленной работы и удовлетворяющее дополнительным ограничениям.

Расчет времени начала работы

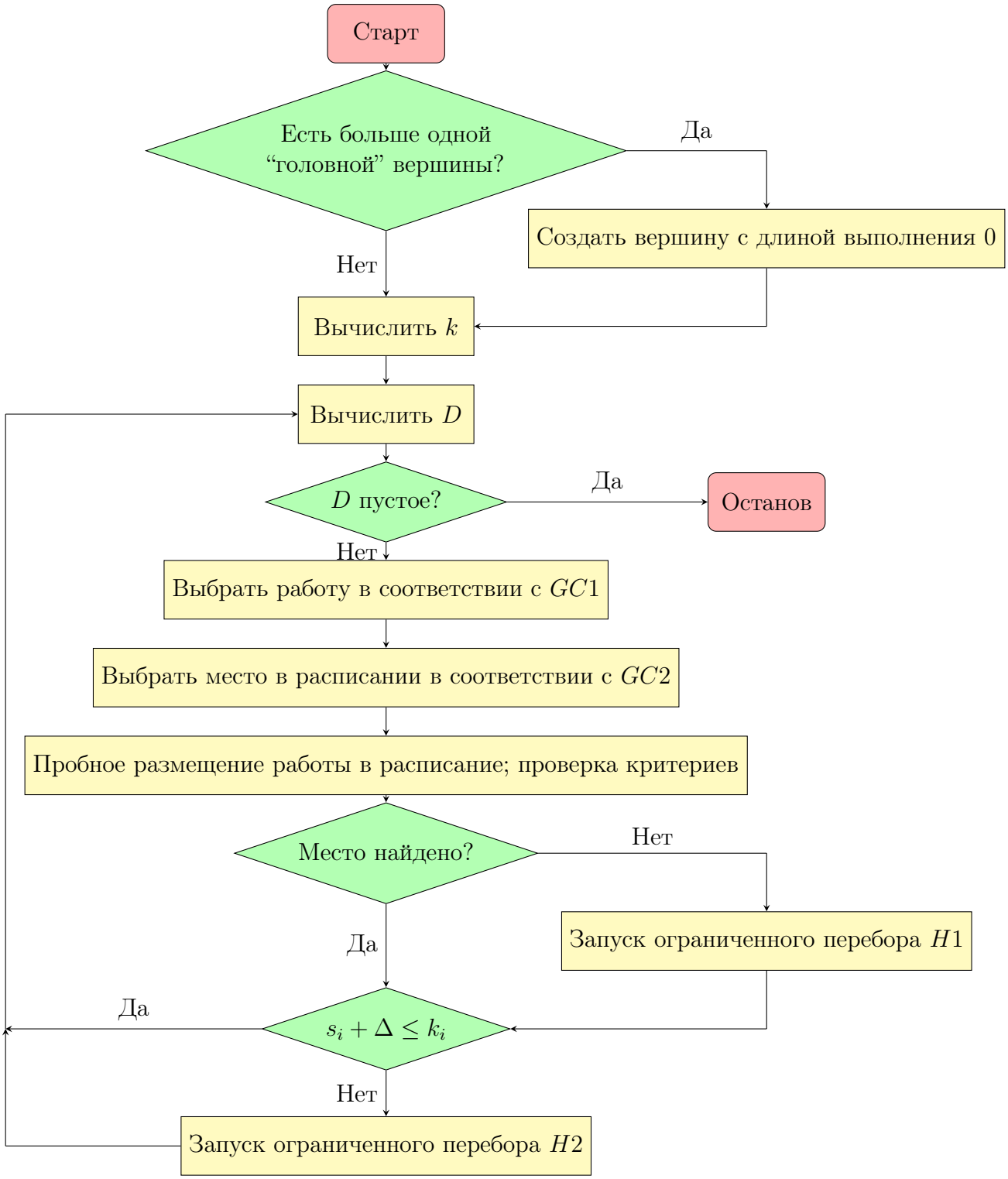
Для того, чтобы рассчитать время начала для конкретной работы на процессоре p требуется:

1. Вычислить вектор $PJ_{k=1}^{L}$, где L – количество предшественников у работы. Элементами этого вектора будут являться суммы вида $s_k + C_{kr} + D_{rj}$, где r – номер процессора, на котором размещен предшественник.

2. Максимумом этого вектора и будет являться первое доступное начало выполнения работы на данном процессоре. $n_j = \max PJ$

После расчета возможного времени начала n_j на каждом процессе требуется найти первое возможное время на процессоре. Для этого для каждой работе, которая следует после n_j берутся все окончания работ и расчитывается, есть ли последующая работа, которая начинается до возможного окончания работа. Если да, то работу возможно разместить в этом “слоте”. Иначе, рассматреть окончание следующей работы. Заметим, что всегда можно разместит работу в конец расписания на процессоре.

Рис. 6: Блок-схема алгоритма



6 Программная реализация алгоритма

Алгоритм реализован на языке C++ с помощюь фреймворка boost. Полный код выложен в репозитории [5]

Проект обладает следующей структурой:

1. logging - функции настройки логирования для проекта. Реализовано на основе Boost ::log
2. schedule - модуль для работы с графом входных данных и матрицами C и D, подаваемыми на вход. Реализован на основе Boost::graph и Boost::uBLAS.
3. time_schedule - модуль для работы с временной диаграммой.
4. main.cpp - main() программы. Основной алгоритм реализован тут. Разбор аргументов основан на Boost::program_options.
5. Doxyfile - файл с настройками Doxygen.

Для сборки проекта используется CMake.

Листинг 1: Сборка проекта

```
1  mkdir build
2  cd build
3  cmake ..
4  make
```

В зависимостях проекта стоит boost версии 1.74, хотя, в теории, более ранние версии могут поддерживаться.

Для сборки документации (на английском) используется Doxygen.

Листинг 2: Сборка документации

```
1  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

Для алгоритма с ограничением сбалансированности нагрузки процессоров были выбраны следующие параметры: $C_1 = 1,\ C_2 = 0.7$; для алгоритма с ограничением на число межпроцессорных передач - $C_1 = 1,\ C_2 = 0.5,\ C3 = 0.5$. Заметим, что в дальнейших исследованиях следует переписать алгоритм для работы с допускной системой выбора процессоров для размещения, а так же подбирать параметры не вручную, а при помощи библиотеки hyperopt. Параметр $\Delta = 10$ оставался константным на протяжении всего исследования. Подбор его оптимума - тема для дальнейших исследований.

7.3 Исследование эффективности алгоритма

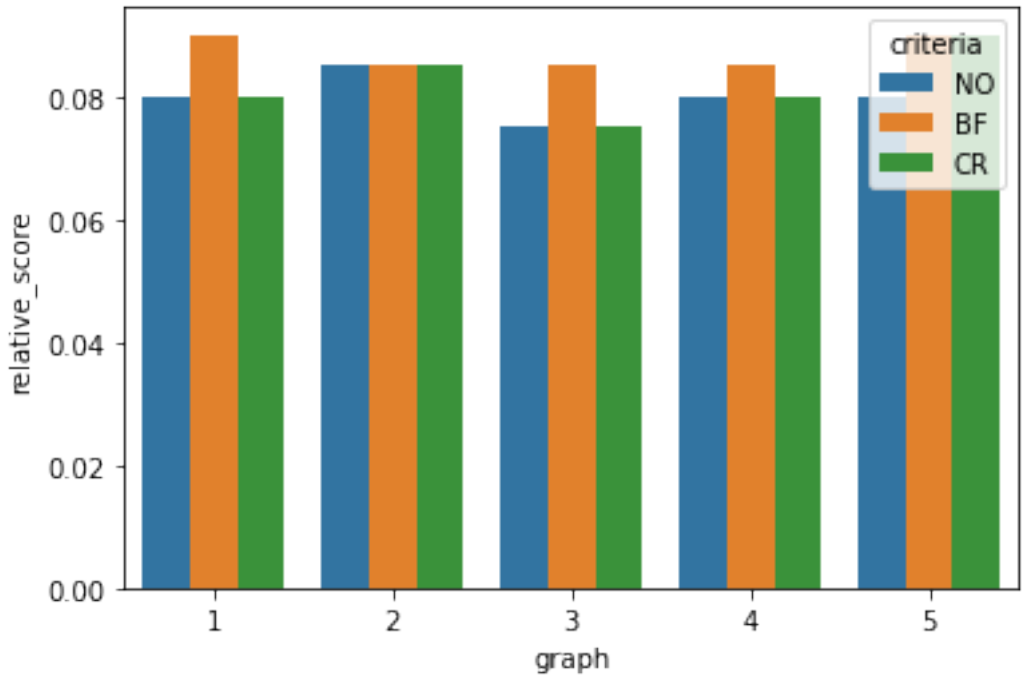


Рис. 7: Распределение точности полученного расписания относительно оптимума. Меньше - лучше.

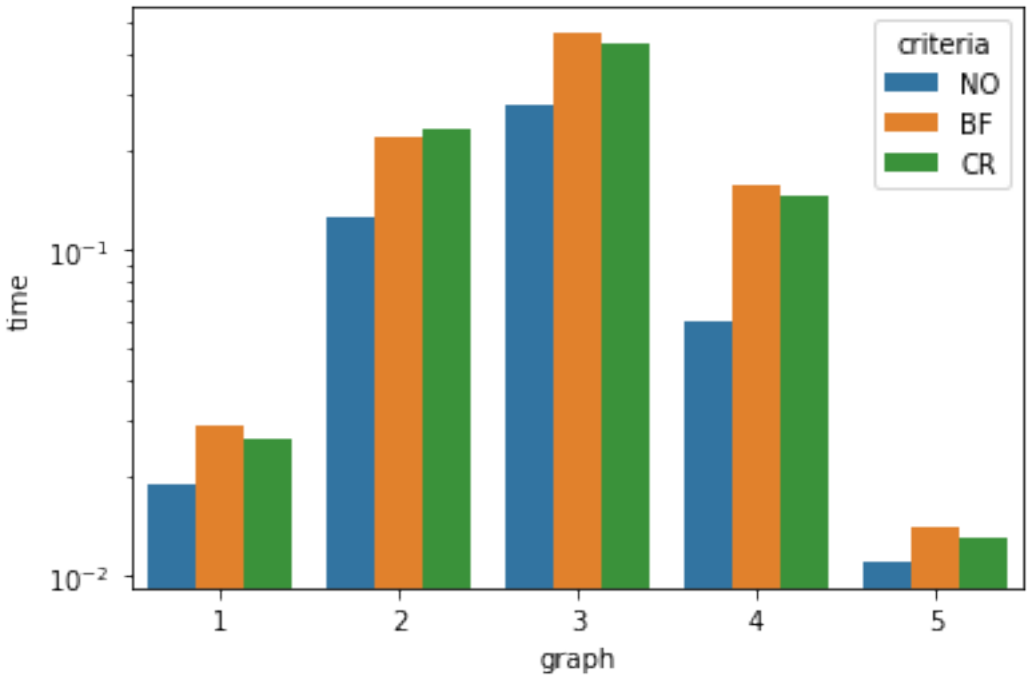


Рис. 8: Время выполнения алгоритма на каждом из тестовых графов. Меньше - лучше.

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

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

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

На рисунке 8 предоставлено время работы алгоритма для каждого из тестовых графов. Из графа видно, что время линейно зависит от количества передач данных между процессорами.

В ходе исследования подтвержден факт того, что более 90% работ размещаются в расписании при помощи жадного критерия, упомянутый в [7]

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.
2. Разработан алгоритм.
3. Реализован алгоритм.
4. Произведено исследование свойств алгоритма.

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. *Coffman E. G.* Computer and job-shop scheduling theory. — Nashville, TN : John Wiley & Sons, 02.1976. — ISBN 0471163198.
2. *Davis R. I.*, *Burns A.* A Survey of Hard Real-Time Scheduling for Multiprocessor Systems // ACM Computing Surveys. — 2011. — Окт. — Т. 43, № 4.
3. Introduction to Algorithms / T. H. Cormen [и др.]. — 2nd. — The MIT Press, 2001. — ISBN 0262032937.
4. *Rzadca K. S. F.* Heterogeneous Multiprocessor Scheduling with Differential Evolution // IEEE Congress on Evolutionary Computation. — 2005. — Т. 3.
5. *Savitsky I.* multiprocessor-scheduling. — URL: `https://github.com/ipsavitsky/multiprocessor-scheduling/tree/main/code` (дата обр. 23.05.2022).
6. *К.В. Шахбазян Т. Т.* Обзор методов составления расписаний для многопроцессорных систем // Журнал советской математики. — 1981. — Т. 15, № 5. — С. 651—669.
7. *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.
8. *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