Обзор по задаче 1

19 апреля 2022 г.

Оглавление

Содержание

1 Введение

Целями данного обзора являются: рассмотрение различных алгоритмов, решающих задачу построения многопроцессорного расписания минимальной длительности и выделение классов алгоритмов с учетом математических постановок задач из области исследования 1 и классов исходных данных для задач из области исследования 1.

Рассматриваемая задача построения многопроцессорного расписания минимальной длительности принадлежит к классу NP-hard, так как является частным случаем классической NP-hard задачи построения списочного расписания [Sha81]. Для задач из класса NP-hard не существует полиномиальных алгоритмов, оптимально решающих задачу. Однако, для некоторых задач из NP-hard удается разработать псевдополиномиальные алгоритмы. На практике часто используют приближенные полиномиальные алгоритмы либо алгоритмы, специализированные под конкретные классы входных данных.

Алгоритмы построения многопроцессорных расписаний можно разделить на два класса, различающихся по используемым методам построения расписаний:

1. *Конструктивные алгоритмы.* Алгоритмы этого класса начинают процесс построения расписания с пустого расписания, не содержащего работ, а затем на каждом шаге добавляют работы в расписание.

2. *Итерационные алгоритмы.* Такие алгоритмы работают с полным расписанием (содержащим все работы). На каждом шаге такие алгоритмы изменяют расписание (при этом расписание остаётся полным), последовательно приближаясь к оптимальному решению.

Можно выделить следующие классы конструктивных алгоритмов:

- алгоритмы, основанные на нахождении максимального потока в транспортной сети;
- алгоритмы, основанные на методе ветвей и границ;
- алгоритмы, основанные на методе динамического программирования;
- алгоритмы, построенные на жадной схеме.

Можно выделить следующие классы итерационных алгоритмов:

- генетический алгоритм
- дифференциальная эволюция
- алгоритм имитации отжига
- алгоритм муравьиных колоний

Далее будут сформулированы критерии обзора, и согласно этим критериям приведены описания всех классов алгоритмов, представленных выше.

2 Критерии обзора

Ниже приведены критерии, по которым будут рассматриваться и сравниваться алгоритмы.

1. Отличие решаемой алгоритмом задач из области исследований
2. Сложность адаптации алгоритма к решаемой задаче.
3. Алгоритм итерационный или конструктивный.
4. Алгоритм является рандомизированным или детерминированным.
5. Порядок сложности алгоритма, возможность масштабируемости алгоритма на данные большой размерности.
6. На данных какой размерности протестирован алгоритм. Качество полученного результата. Время работы на протестированных данных.

3 Конструктивные алгоритмы

3.1 Алгоритмы, основанные на нахождении максимального потока в сети

Алгоритмы построения многопроцессорных расписаний, основанные на нахождении максимального потока в транспортной сети [Gol88] предполагают сведение задачи построения расписания к задаче поиска максимального потока в сети. На основе набора рабочих интервалов, процессоров, и ограничений исходной задачи (например, директивных сроков) строится сеть. Затем, для построенной сети решается задача поиска максимального потока, после чего, по построенному потоку в сети восстанавливается многопроцессорное расписание. Данный метод построения хорошо описан в литературе [S77], однако общим свойством задач, для которого он может применяться, является допустимость прерывания работ. В данной работе рассматривается задача построение расписаний, не допускающая прерывания выполнения рабочих интервалов, поэтому алгоритмы, основанные на нахождении максимального потока в сети неприменимы.

3.2 Алгоритмы, основанные на методе динамического программирования

Алгоритмы, основанные на методе динамического программировании [Bel66], требуют введения двух операций:

1. вложение решаемой задачи в семейство более простых задач;
2. нахождение возвратного соотношения, связывающего оптимальные значения этих подзадач.

Данные алгоритмы позволяют получить глобальный оптимум, но при этом для большинства задач комбинаторной оптимизации имеют не полиномиальную сложность. В частности, для NP-трудных задач построения расписаний [Hel62] сложность алгоритмов – факториальная функция от размера входа (число процессов, число рабочих интервалов и мощность множества исходного отношения частичного порядка). Более того, нахождение возвратного соотношения, связывающего оптимальные значения подзадач требует наличия аналитической модели вычислительной системы. Таким образом, данный алгоритм оказывается неприменим по Критерию 4 *Порядок сложности алгоритма.*

3.3 Алгоритмы, основанные на методе ветвей и границ

Алгоритмы, основанные на методе ветвей и границ [Law66], используют разделение пространства возможных решений (представление в форме некоторого разветвления), верхнюю/нижнюю оценку значения целевой функции для выделенных областей и отсечение областей в которых нет оптимального

решения. Данные алгоритмы позволяют получить глобальный оптимум, но при этом для большинства задач комбинаторной оптимизации имеют не полиномиальную сложность (в частности, для NP-трудных задач составления расписаний) сложность алгоритмов – факториальная функция от размера входа. Метод ветвей и границ позволяет строить алгоритмы, получающие решения с заданной точностью, но при этом верхняя оценка сложности алгоритма зависит от значений входных данных. Возможно и обратное: построение алгоритмов для решения NP-трудных задач, верхняя оценка которых является полиномиальной функцией от размера входа, но точность получаемого решения при этом зависит от значений входных данных. Данные подходы позволяют строить более сложные эвристики, точность которых в ограниченном диапазоне входных данных может быть выше, чем у эвристик, основанных на жадных стратегиях.

Метод ветвей и границ применялся для решения рассматриваемой задачи в ряде исследований [Kas84; Krä97; Fuj11], однако из-за высокой сложности алгоритма рассматривались только небольшие графы задач с количеством вершин не более 1000.

3.4 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 [C01]. 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 построе-

ния временной диаграммы не может быть локализовано, а также проблемой выбора критериев оптимизации индивидуально для каждой подзадачи.

3.5 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 оптимизирующей функции. Для того чтобы исправить этот недостаток был предложен подход [A17], сочетающий жадные алгоритм и ограниченный перебор.

Для решения задачи построения многопроцессорного расписания был предложен алгоритм на основе этого подхода [A17], состоящий из следующих шагов:

1. Создание множества "доступных"работ (работы без "родителей"или работы, родители которых уже появились в расписании).
2. Выбор работы из пула согласно жадному критерию $C1$ – максимальное количество потомков.
3. Выбор и размещение работы в расписании с учетом жадного критерия $C2$: работа с ближайшим временем завершения.
4. Проверки критериев:
   (a) Путь не критический
   (b) В расписании нет простоев
5. В случае выполнения критериев a)-b), выбирается следующая работа – перейти к пункту 1 и обновить множество "доступных"работ.
6. Иначе обратить внимание на работы, которые образуют простой и/или время не равно времени критического пути и переставить их ближе к началу (родительские работы – в первую очередь). Это часть ограниченного перебора. В случае невыполнения условий, можно повышать "глубину"ограниченного перебора. В случае, если в какой-то конфигурации все условия будут выполнены - перейти к п.1

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 [A17] 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, пункт 4 алгоритма. Более подробно про учет дополнительных условий см. раздел "Схемы выбранных алгоритмов".

Этот алгоритм детерминирован и имеет единственный его параметр – глубину 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 $O(n \log n)$

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 Итерационные алгоритмы

4.1 Генетические алгоритмы

4.1.1 Описание подхода

Генетические алгоритмы (ГА), благодаря своей эффективности, применяются для решения таких задач, как построение оптимальных игровых стратегий, настройка и обучение нейронных сетей, оптимизация запросов в базах данных, построение расписаний [A13; Kos00], различные задачи на графах (раскраска, задача коммивояжера, нахождение паросочетаний), машинного обучения [Kos13] и многих других. Но при построении ГА для решения конкретной задачи требуется решить ряд проблем [A13]. Одной из таких проблем является выбор значений вероятностей операций мутации и скрещивания, которые оказывают сильное влияние на эффективность работы алгоритма. В настоящее время не существует никаких теоретических результатов, позволяющих решить эту проблему. В большинстве практических применений генетических алгоритмов значения параметров операций скрещивания и мутации подбираются экспериментально. Однако, для решения данной проблемы был предложен генетический алгоритм с самообучением [Kos15], основная идея которого заключается в изменении вероятностей скрещиваний и мутаций, в зависимости от того насколько удачным или неудачным оказалось применение конкретной операции (скрещивания или мутации) к элементу решения.

Генетические и эволюционные алгоритмы основаны на использовании механизмов естественной эволюции. Эволюция осуществляется в результате взаимодействия трех основных факторов: изменчивости, наследственности, естественного отбора. Изменчивость служит основой образования новых признаков и особенностей в строении и функциях организма. Наследственность закрепляет эти признаки. Естественный отбор устраняет организмы, плохо приспособленные к условиям существования. Генетические и эволюционные вычисления получили общее признание после выхода книги Джона Холланда “Адаптация в естественных и искусственных системах” [Hol75]

Приведем схему классического генетического алгоритма Джона Холланда:

1. Генерация случайным образом начальной популяции.
2. Вычисление функции выживаемости для каждой строки популяции.
3. Выполнение операции селекции.
4. Выполнение операции скрещивания:
   (a) Выбор пары для скрещивания.
   (b) Для каждой выбранной пары:
      i. с заданной вероятностью выполняется скрещивание – в результате создаются два потомков;

ii. замена в популяции родителей на их потомков.

5. Выполнение операции мутации.

6. Если критерий останова не достигнут, перейти к шагу 2, иначе завершить работу алгоритма.

Популяцией называется множество битовых строк, каждая из которых представляет в закодированном виде одно из возможных решений задачи. По битовой строке может быть вычислена функция выживаемости, которая характеризует качество решения. В качестве начальной популяции может быть использован произвольный набор битовых строк. Основные операции алгоритма - селекция, скрещивание и мутация - выполняются над элементами популяции. Результатом их выполнения является очередная популяция. Данный процесс продолжается итерационно до тех пор, пока не будет достигнут критерий останова.

4.1.2 Алгоритм построения многопроцессорного расписания

В качестве иллюстрации применения генетического алгоритма для решения задачи построения многопроцессорного расписания была рассмотрена работа [Akb15]. В ней рассматривается задача построения многопроцессорных расписаний в следующей формулировке. Дан ориентированный ациклический граф, задающих порядок выполнения работ. Для каждого ребра определено время передачи данных между двумя работами, если они назначены на различные процессоры. Для работ, находящихся на одном процессоре время передачи данных равно нулю. Так же задано время выполнения каждой работы на процессорах. Требуется минимизировать время работы расписания.

В данной постановке время передачи данных задано между работами в отличие от обобщенной задачи 1, в которой рассматривается время передачи между процессорами. Также в [Akb15] не рассматриваются процессоры, между которыми невозможна прямая передача данных. Отличие же от задачи 1 состоит еще и в том, что в статье не рассматриваются дополнительные ограничения на долю межпроцессорных передач и несбалансированность распределения по процессорам. Схема генетического алгоритма представлена выше.

Алгоритм:

1. Генерация начальной популяции случайным образом.

2. Вычисление функции выживаемости для каждой строки популяции.

3. Выполнение операции селекции.

4. Выполнение операции скрещивания.

5. Выполнение операции мутации.

6. Выполнение операции селекции.

7. Выполнение операции инверсии.

8. Выполнение операции скрещивания.

9. Пока число популяций не достигло предела, вернуться на шаг 2.

Граф зависимостей можно представить в виде ярусно-параллельной форме (ЯПФ)
Ярусно-параллельная форма графа - деление вершин ориентированного ациклического графа на пронумерованные подмножества V_i такие, что, если дуга e идет от вершины $v_1 \in V_j$ к вершине $v_2 \in V_k$, то обязательно $j < k$.

Каждое из множеств V_i называется ярусом ЯПФ, i - его номером, количество вершин в ярусе - его шириной. Количество ярусов в ЯПФ называется её высотой, а максимальная ширина её ярусов — шириной ЯПФ. Для ЯПФ графа алгоритма важным является тот факт, что операции, которым соответствуют вершины одного яруса, не зависят друг от друга, и поэтому заведомо могут быть выполнены параллельно на разных устройствах вычислительной системы. Минимальной высотой всех возможных ЯПФ графа является его критический путь. Построение ЯПФ с высотой, меньшей критического пути, невозможно.

Расписание представляется в виде хромосомы - двумерного массива, в первой строке которого расположены работы в порядке их выполнения, а во второй - процессоры, на которых выполняются соответствующие работы.

Вершины графа работ находятся в одном ярусе, если их высота в ЯПФ одинакова. Расписание в виде хромосомы представлен на Рис. №.., Фрагменты хромосом с разным цветом соответствуют разным ярусам в ЯПФ.

Начальная популяция генерируется таким образом, чтобы назначить работы из критического пути графа зависимостей на самый быстрый процессор.

Функция пригодности представителя популяции, которая используется в операции селекции - это функция, вычисляющая длительность расписания.

Во время селекции заранее заданный процент лучших представителей (который является параметром алгоритма, например, 10%) текущей популяции переходят в следующую популяцию. Остальные представители новой популяции выбираются из текущей случайно - чем выше их пригодность, тем выше вероятность выбора представителя.

При скрещивании выбираются два родителя и некоторый фрагмент хромосомы. В выбранном фрагменте у родителей меняются процессоры, на которые назначены работы, см. Рисунок №...

Мутация случайно меняет столбцы хромосомы внутри одного яруса ЯПФ. Инверсия обращает хромосому - переставляет работы внутри каждого яруса в обратном порядке. Обращение внутри яруса всегда возможно, так как работы не связаны друг с другом. В результате инверсии получается

новая популяция обращенных хромосом, которая скрещивается с изначальной.

Для того, чтобы алгоритм был применим к задаче 1, необходима коррекция функции пригодности с учетом связности процессоров и иного задания времени передачи данных. Одним из вариантов применения алгоритма к подзадачам с дополнительными ограничениями (BF, CR, CR_2) состоит в том, чтобы пересчитывать ограничения BF, CR, CR_2 после каждого применения операций (скрещивания и мутаций) и отменять изменения, если ограничения перестали выполняться. Другим вариантом может служить добавление штрафа в функцию пригодности при превышении критерия. При генерации начальной популяции также необходимо учитывать ограничения BF, CR, CR_2.

Экспериментальное исследование

В ходе экспериментального исследования размер популяции устанавливался в 20 и 30, наибольшее число поколений – 1000. Эффективность сравнивалась с такими эвристическими алгоритмами, как: Modified Critical Path (MCP) [Wu 90], Dynamic Critical Path (DCP) [Kwo96], Dominant Sequence Clustering (DSC) [Yan94], The Mobility Directed algorithm (MD) [Wu 90]. Размеры графов - 9-20 вершин, число процессоров - 2-6. Граф работ генерировался случайным образом. Результат алгоритма из [Akb15] лучше, чем у всех перечисленных алгоритмов.

Сложность алгоритма $O\left(G \cdot \left(n + n^2 + n^2 + n^2 \cdot l + e \cdot m\right)\right)$, где G - число положений, n - число работ во входном DAG, l - число ярусов в ЯПФ, построенной по графу работ, e - число ребер в графе работ, m - число процессоров. Для плотного графа, в котором число ребер сопоставимо с квадратом числа вершин, сложность составит $O\left(G \cdot n^2 \cdot m\right)$

Результаты экспериментального исследования говорят о том, что алгоритм плохо масштабируется на данные большой размерности, так как при $n \sim 10^6, m \sim 10^3$ число операций становится велико даже без учета числа поколений, которое с ростом размера входа также должно увеличиваться. Это является основным недостатком генетического алгоритма.

число операций становится велико даже без учета числа поколений, которое с ростом размера входа также должно увеличиваться. Это является основным недостатком генетического алгоритма.

4.2 Дифференциальная эволюция

4.2.1 Описание подхода

Рассмотрим конкретную реализацию алгоритма дифференциальной эволюции из работы [Rza05a]. Расписание представляется вектором приоритетов: для каждой работы вычисляется приоритетность ее выполнения. При подсчете времени выполнения расписания из списка не назначенных работ берется та, у которой наибольший приоритет, и устанавливается на процессор так, чтобы время выполнения было наименьшим из возможных.

1. Инициализация начальной популяции - *pop*
2. Для каждого представителя популяции вычислить длительность расписания.
3. $newPopulation = \emptyset$
4. Для каждого представителя pop текущей популяции i:
   (a) Случайно выбрать 3 различных представителя r_1, r_2, r_3.
   (b) Создать потомка с приоритетами:
   $$\underline{pr_{child}} = \underline{pr_i} + K \cdot \left(\underline{pr_{r_3}} - \underline{pr_i}\right) + F \cdot \left(\underline{pr_{r_1}} - \underline{pr_{r_2}}\right)$$
   (c) Для потомка применить корректировку приоритетов и нормализовать их.
   (d) Посчитать длину расписания потомка $makespan_{child}$
   (e) Если $makespan_{child} < makespan_i$, то включить потомка в новую $newPopulation$.
   Иначе включить представителя i в $newPopulation$.
5. $pop = newPopulation$.
6. Пока число популяций не достигло заранее заданного предела, вернутся на шаг 3.

Для первой части начального поколения списки приоритетов вычисляются с помощью различных приближений. Оставшаяся часть начального поколения генерируется случайно.

Для каждой работы производится коррекция приоритетов в соответствии с графом зависимостей - увеличиваются приоритеты у работ, предшественники которых имеют более высокий приоритет.

В работе [Rza05a] вычислительная система представлена в виде графа, вершинами которого являются процессоры. Дан ациклический граф зависимостей работ. Для каждой пары работ, соединенных ребром, задано время передачи данных, если они назначены на различные процессоры. Работы выполняемые на одном процессоре имеют нулевое время передачи данных. Полное время передачи данных вычисляется как произведение

времени передачи данных между работами и кратчайшего пути между соответствующими процессорами.

Задача, рассматриваемая в [Rza05a] близка к рассматриваемой задаче 1. Одно из отличий заключается в задании времени передачи данных между работами. Для сведения задачи из [Rza05a] к рассматриваемой задаче 1 необходимо изменить соответствующим образом функцию вычисления длительности расписания.

В [Rza05a] не рассматриваются дополнительные ограничения задачи 1. Однако, можно поступить аналогичным образом как и в генетическом алгоритме. Либо добавить штраф при вычислении длительности расписания, либо не добавлять потомков, которые не удовлетворяют ограничениям BF, CR, CR_2.

Сложность алгоритма зависит от размера популяции и количества поколений. Также зависит от функции вычисления длительности расписания, Для нее в статье указана сложность $O(N \cdot \log N)$, где N - число вершин графа.

Экспериментальное исследование В работе [Rza05a] было проведено два экспериментальных исследования. Первое заключалось в сравнении качества получаемого решения алгоритмом дифференциальной эволюции с генетическим алгоритмом при одинаковом числе поколений и размере популяции. Методика проведения сравнения алгоритмов заключалась в следующем. Генерировалось начальная популяция - начальное приближение расписания. Далее запускался генетический алгоритм и алгоритм дифференциальной эволюции и после завершения их работы сравнивались полученные ими расписания по длительности. По результатам сравнения генетический алгоритм улучшал расписание на $1 - 2\%$, а алгоритм из [Rza05a] давал улучшение в $4 - 5\%$.

Было проведено второе экспериментальное исследование, которое оценивало способность сходимости алгоритма к глобальному оптимуму. Для фиксированного набора входных данных – графа зависимостей работ в виде DAG и графа процессоров – генерировалось оптимальное расписание. Предложенный алгоритм строил расписание на 3% длиннее, чем оптимальное.

Первое экспериментальное исследованием проводилось на графах различной плотности с $50 - 500$ вершинами. Второе экспериментальное исследование проводилось на графах с $50 - 200$ вершинами. При количестве поколений 50 и размере популяции 24 время работы алгоритма в среднем составляло 11 секунд.

Выводом из экспериментальных исследований из [Rza05a] является тот факт, что дифференциальная эволюция может работать быстрее генетического алгоритма и давать лучшие результаты. Однако оценки сложности алгоритма и время его работы на небольших данных могут говорить о плохой масштабируемости алгоритма на графы большой размерности, что важно в исследуемой задаче 1.

4.3 Алгоритм имитации отжига

4.3.1 Описание подхода

Алгоритм имитации отжига – это стохастический метод, предложенный в 1983 [Rza05b]. Алгоритм имитации отжига хорошо себя зарекомендовал в решении задачи о назначении работ на многопроцессорную вычислительную систему [Ysk00], задачах управления [Yan21], задаче распределения работ по процессорам для обеспечения максимальной надежности вычислительной системы [Gam06], задаче построения расписания спортивных игр [A21], также в задачах машинного обучения [Y08]

В тоже время использование алгоритма имитации отжига позволяет получать хорошие приближенные решения для целого ряда NP-трудных задач комбинаторной оптимизации. Принципы работы алгоритма имитации отжига описаны в [Was88].

4.3.2 Алгоритм построения многопроцессорного расписания

В работе [Kal08] рассматривается задача наиболее близкая к задаче 1: задача минимизации времени выполнения расписания. На входе дан ориентированный граф зависимостей работ. Для каждой работы известна ее вычислительная сложность на процессорах системы.

Первое отличие от задачи 1 - отсутствие затрат на передачи данных, вследствие чего необходима коррекция целевой функции в соответствии с этим. Второе отличие состоит в том, что не рассматриваются ограничения на межпроцессорную передачу и балансировку. Учесть это в задаче 1 можно аналогичными способами, предложенными в генетическом алгоритме.

Схема алгоритма имитации отжига выглядит следующим образом:

1. Генерация корректного начального расписания.
2. Установление начальной температуры T_0.
3. Применение операции преобразования расписания.
4. Если изменение целевой функции $\Delta f \leq 0$ (т. е. расписание улучшилось), то новый вариант расписания считать текущим. Иначе принять новое расписание в качестве текущего с вероятностью $p = e^{\frac{-\Delta}{T}}$, где T - текущая температура.
5. Повторить заданное число раз шаги 3 и 4 без изменения текущей температуры.
6. Повысить текущую температуру в соответствии с выбранным законом.
7. Если не выполнен критерий останова, то перейти к шагу 3.

Расписание представляется в виде ярусной формы максимальной высоты. В ней на ярусе расположена ровно одна работа, причем все работы-предшественники расположены на более высоких ярусах.

Допускается любая генерация начального расписания. В [Kal08] в качестве начального расписания взято расписание, генерируемое жадным алгоритмом. Режим понижения температуры задан законом Больцмана $T = \frac{T_0}{\ln 1+i}$, где T_0 - начальная температура, i - номер текущей итерации. Критерием останова может служить ограничение на число итераций или отсутствие улучшение расписания заданное число шагов.

Применяется две операции преобразования: перенос работы на другой процессор и смена порядка работ на одном процессоре. Сложности операций $O(1)$ и $O(N)$ соответственно, где N - число работ.

Основной недостаток алгоритма имитации отжига заключается в его высокой вычислительной сложности, так как для нахождения хорошего решения требуется медленное понижение температуры, что приводит к увеличению итераций. Для решения этой проблемы в статье был предложен оригинальный метод распараллеливания алгоритма, основанный на разбиении на непересекающиеся области.

В области поиска (изначально исходное пространство расписаний) выбираются две работы таким образом, чтобы они не были связаны транзитивным отношением порядка. Из этого графа получается 3 области поиска:

- работы распределены на разные процессоры;
- работы распределены на один процессор, и вторая работа выполняется после первой;
- работы распределены на один процессор, и первая работа выполняется после второй.

Работы выбираются так, чтобы критические пути графов в получившихся областях сильно отличались. Это может позволить отбросить области с заведомо долгими расписаниями.

Области распределяются между узлами вычислительной системы, на которой запускается алгоритм. На узлах отдельно выполняются последовательные алгоритмы имитации отжига для областей и отбрасываются те, время работы которых сильно отличается от остальных. Раз в определенный промежуток времени между узлами происходит обмен, чтобы отсечь области, которые имеют сильно отличающиеся времена выполнения расписаний.

Дополнительным преимуществом данного параллельного подхода может служить то, что он не требует высокого трафика обмена между вычислительными узлами.

Экспериментальное исследование

Экспериментальное исследование проводилось на следующих входных данных: ориентированные графы работ с $100 - 250$ вершинами, рассматривались графы различной плотности, количество процессоров - $40 - 100$.

Предложенные в работе [Kal08] последовательный и параллельный алгоритмы имитации отжига, использующие разбиение пространства решений на области, показали высокую эффективность при решении задач построения многопроцессорных расписаний. Последовательный алгоритм позволяет уменьшить время решения задачи до 3 раз по сравнению с классическим алгоритмом имитации отжига, сохраняя, а во многих случаях даже улучшая качество получаемых решений. Параллельный алгоритм может быть эффективно реализован на многопроцессорном вычислителе или вычислительном кластере, состоящем из нескольких вычислительных узлов, поскольку не требует высокого трафика обмена между узлами сети.

Построение алгоритма имитации отжига, основанного на этом подходе, требует решения следующих проблем:

1. Разбиение исходного пространства корректных решений на несколько областей, дающих в объединении все пространство.

2. Выбор начального корректного решения в каждой из областей.

3. Введение операций преобразования решения таким образом, чтобы они были замкнуты в каждой из областей.

4. Выбор способа распределения областей по узлам вычислительной системы и схемы отсечения "неперспективных"областей в ходе работы алгоритма.

5 Алгоритм муравьиных колоний

5.1 Описание подхода

Муравьиные алгоритмы, или алгоритмы муравьиных колоний основаны на принципах взаимодействия муравьев в естественной среде, а именно на их роевом интеллекте. Каждый отдельный муравей обладает информацией только о локальной обстановке, ни один из них не имеет представления обо всей картине в целом, только то, что узнал от другим муравьев явно или неявно. На неявных взаимодействиях муравьёв основаны механизмы поиска кратчайшего пути от муравейника до источника пищи. Каждый раз, проходя по такому пути, муравей оставляет за собой след из феромонов. Другие муравьи, почувствовав след из феромонов, будут инстинктивно устремляться к нему. Поскольку все муравьи оставляют за собой дорожки феромонов, то чем больше муравьев проходит по определенному пути, тем более привлекательным данный путь становится для всех муравьев. При этом, чем короче путь до источника пищи, тем меньше времени требуется муравьям на его преодоление, а следовательно, тем быстрее оставленные на нем следы становятся заметными.

Чтобы применить алгоритм муравьиных колоний к рассматриваемой задаче, необходимо представить задачу в терминах поиска пути в графе. Найденный путь и будет представлять решение задачи. Различные пути в графе – это пути, по которым перемещаются муравьи и оставляют свои феромоны.

Стандартная схема муравьиного алгоритма выглядит следующим образом:

1. Задание начального количества феромонов на ребрах графа;
2. Задание начального количества муравьев;
3. Задание начального положения муравьев;
4. Построение муравьями путей;
5. Вычисление целевой функции для каждого пути;
6. Обновление количества феромона на каждом ребре;
7. Повторять шаги 1) – 7) пока не будет выполнено условие останова (например, расписание построено – все работы размещены, критерии выполнены).

Муравьиные алгоритмы нашли широкое применение в приближенном решении NP-трудных задач. Так в работах [C00] рассматривается задача минимизации суммарного запаздывания на одном процессоре. В Smitha_2011 приведено описание сбалансированного алгоритма муравьиных колоний для планирования вычислений в GRID-системах. В [al15] рассматривается гибридный муравьиный алгоритм для проектного планирования при условии

ограниченности ресурсов. Часто алгоритм муравьиных колоний показывает хорошие результаты, которые можно улучшить при помощи различных модификаций (например, локального поиска или правил исключения. В работе [M02] приводится сравнение алгоритма муравьиных колоний с различными метаэвристическими алгоритмами, в числе которых алгоритм имитации отжига и генетический алгоритм, и установлено, что для задач больших размерностей алгоритм муравьиных колоний показывает более лучшие результаты.

В работе [D05] предложен алгоритм муравьиных колоний, решающий задачу коммивояжера. В ней показано, что предложенный алгоритм можно легко адаптировать для решения задачи построения многопроцессорного расписания. Для решения задачи 1 достаточно для каждого муравья отрезать пути, которые не удовлетворяют дополнительным ограничениям BF, CR, CR_2.

Таким образом, можно сделать вывод, что алгоритм муравьиных колоний является рандомизированным алгоритмом, на каждой вершине муравей выбирает какой-либо путь с определенной вероятностью.

Алгоритм итерационный: на каждой итерации строится новый путь каждого муравья и новое расписание. Муравьиный алгоритм получает ответ за большее время, однако может получать более качественное решение.

5.1.1 Алгоритм построения многопроцессорного расписания

В качестве конкретной реализации муравьиного алгоритма рассмотрим работу [V17] Задача, первоначальная рассмотренная в статье сильно отличается от задачи 1, однако так же как и задача 1 сводится к задаче построения в графе работ пути минимальной длины. В [V17] решается задача о размещении виртуальных машин (V) и виртуальных хранилищ данных (S) на вычислительные узлы таким образом, чтобы количество размещенных работ было максимальным.

Алгоритм, решающий данную задачу, выглядит следующим образом:

1. Пусть C - это граф, вершины которого представляют собой работы и вычислительные узлы.

2. Все муравьи начинают путь из специальной начальной вершины O.

3. Находясь в вершине O первый раз, муравей выбирает одну вершину из $V_0^1, \ldots, V_0^R$ - множество виртуальных машин. После того как муравей прошел по всем вершинам V_0^K, он возвращается в вершину O и выбирает одну вершину из $S_0^1, \ldots, S_0^J$ - множество виртуальных хранилищ данных. Путь считается построенным когда муравей пройдет по всем вершинам S_0^K

4. Выбирать вершины $V_1, \ldots, V_N$ и $S_1, \ldots, S_Q$ муравьям можно больше одного раза.

5. Выбрав вершину, соответствующую V_0^K или S_0^K, муравей обязан выбрать следующей вершину, соответствующую вычислительному узлу.

6. Выбрав вершину, соответствующую вычислительному узлу, муравей обязан вернуться в вершину, из которой приш

В контексте задачи, рассматриваемой в [V17] виртуальные машины и виртуальный хранилища данных - это аналоги работ в задаче 1. А вычислительный узлы – это аналоги процессоров в задаче 1.

Экспериментальное исследование

В [V17] было проведено экспериментальное исследование разработанного алгоритма на различных случайно сгенерированных наборах данных, разделенных на два класса. Результат работы алгоритма сравнивался с результатом работы генетического алгоритма.

Достоинства:

1. Предложенный в [V17] сравнительно эффективен при маленьком количестве вершин.

2. Меньше подвержен начальному состоянию чем генетические алгоритмы.

3. Может адаптироваться к изменению начальных условий, в отличии от генетических алгоритмов.

Недостатки:

1. Сходимость гарантируется, но время сходимости неопределено

2. Все еще сильно зависит от начального состояния

Список литературы

[Hel62] Karp R. M Held M. «A Dynamic Programming Approach to Sequencing Problems». B: *Journal of the Society for Industrial and Applied Mathematics* 10 (1962).

[Bel66] R. Bellman. «Dynamic programming». B: *Science* 153 (1966).

[Law66] Wood D. E. Lawler E. L. «Branch-and-Bound Methods: A Survey». B: *Operations Research* 14 (1966).

[Hol75] J. N. Holland. «Adaptation in Natural and Artificial Systems». B: *Ann Arbor, Universitry of Michigan Press* (1975).

[S77] Stone H. S. «Multiprocessor Scheduling with the Aid of Network Flow Algorithms». B: *IEEE Transactions on Software Engineering* SE-3.1 (1977).

[Sha81] Tushkina T. A. Shakhbazyan K. V. «A survey of scheduling methods for multiprocessor systems». B: 1981.

[Kas84] Narita Kasahara. «Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing». B: *IEEE Transactions on Computers* 33 (1984).

[Gol88] Tarjan R. E. Goldberg A. V. «A new approach to the maximum-flow problem». B: (1988).

[Was88] Wasserman. *Neural Computing: Theory and Practice*. 1988.

[Wu 90] Gajski D. D. Wu M. Y. «Hypertool: A programming aid for message-passing systems». B: *IEEE Transactions on Parallel and Distributed Systems* 1 (1990).

[Yan94] Gerasoulis A. Yang T. «DSC: Scheduling parallel tasks on an unbounded number of processors». B: *IEEE Transactions on Parallel and Distributed Systems* 5 (1994).

[Kwo96] Ahmad I. Kwok Y. K. «Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors». B: *IEEE Transactions on Parallel and Distributed Systems* 7 (1996).

[Krä97] A. Krämer. «Branch and bound methods for scheduling problems with multiprocessor tasks on dedicated processors». B: *Operation Research Spektrum* 19 (1997).

[C00] Bauer A. Bullnheimer B. Hartl R. F. Strauss C. «Minimizing Total Tardiness on a Single Machine Using Ant Colony Optimization». B: (2000).

[Kos00] Trekin A. G. Kostenko V. A. Smeliansky R. L. «Synthesizing structures of real-time computer systems using genetic algorithms». B: *Programming and Computer Software* 26.5 (2000).

[Ysk00] Khalil S. H. Yskandar H. «Assignment of program modules to processors: A simulated annealing approach». B: *European Journal of Operational Research* 122 (2000).

[C01] Cormen T. H. Leiserson C. E. Rivest R. L Stein C. «Introduction to Algorithms». B: *The MIT Press* (2001).

[M02] Gagné C. Price W. Gravel M. «Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times». B: *Journal of the Operational Research Society* 53 (2002).

[D05] Shtovba S. D. «Ant Algorithms: Theory and Applications». B: *Programming and Computer Software* 31.4 (2005).

[Rza05a] Seredynski F. Rzadca K. «Heterogeneous Multiprocessor Scheduling with Differential Evolution». B: *IEEE Congress on Evolutionary Computation* 3 (2005).

[Rza05b] Seredynski F. Rzadca K. «Heterogeneous Multiprocessor Scheduling with Differential Evolution». B: *IEEE Congress on Evolutionary Computation* 3 (2005).

[Gam06] Yskandar H. Gamal A. «Task allocation for maximizing reliability of distributed systems: A simulated annealing approach». B: *Journal of Parallel and Distributed Computing* 66 (2006).

[Kal08] Kostenko V. A. Kalashnikov A. V. «A Parallel Algorithm of Simulated Annealing for Multiprocessor Scheduling». B: *Journal of Computer and Systems Sciences International* 47.3 (2008).

[Y08] Lin S. W. Lee Z. J. Chen S. C. Tseng T. Y. «Parameter determination of support vector machine and feature selection using simulated annealing approach». B: *Applied Software Computing* 8 (2008).

[Fuj11] S. Fujita. «A Branch-and-Bound Algorithm for Solving the Multiprocessor Scheduling Problem with Improved Lower Bounding Techniques». B: *IEEE Transactions on Computers* (2011).

[A13] Kostenko V. A. «Scheduling algorithms for real-time computing systems admitting simulation models». B: *Programming and Computer Software* 39.5 (2013).

[Kos13] Shcherbinin V. V. Kostenko V. A. «Training Methods and Algorithms for Recognition of Nonlinearly Distorted Phase Trajectories of Dynamic Systems». B: *Optical Memory and Neural Networks* 22.1 (2013).

[Akb15] Rashidi H. Akbari M. «An Efficient Algorithm for Compile-Time task scheduling problem on heterogeneous computing systems». B: *International Journal of Academic Research* 7.Part A, 2nd (2015).

[al15] Myszkowski P. B. Skowroński M. E Olech Ł. P. et al. «Hybrid ant colony optimization in solving multi-skill resource-constrained project scheduling problem». B: *Soft Comput* 19 (2015).

[Kos15] Frolov A. V. Kostenko V. A. «Self-learning genetic algorithm». B: *Journal of Computer and Systems Sciences International* 54.4 (2015).

[A17] Kostenko V. A. «Combinatorial Optimization Algorithms Combining Greedy Strategies with A Limited Search Procedure». B: *Journal of Computer and Systems Sciences International* 56.2 (2017).

[V17] Kostenko V. A. Plakunov A. V. «Ant algorithms for scheduling computations in data centers». B: *Moscow University Computational Mathematics and Cybernetics* 41.1 (2017).

[A21] Rosati R. M. Petris M. Gaspero L. Schaerf A. «Multi-Neighborhood Simulated Annealing for the Sport Timetabling Competition ITC2021». B: *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling - PATAT 2021* II (2021).

[Yan21] Vigneron A. Yang H. «A Simulated Annealing Approach to Coordinated Motion Planning». B: *37th International Symposium on Computational Geometry* 189 (2021).