Омский филиал Федерального государственного бюджетного
учреждения науки
Института математики им. С. Л. Соболева
Сибирского отделения Российской академии наук

На правах рукописи

Захарова Юлия Викторовна

Задачи теории расписаний с альтернативами выполнения работ: вычислительная сложность и методы решения

Специальность 1.2.3 –
Теоретическая информатика, кибернетика

Диссертация на соискание ученой степени
доктора физико–математических наук

Омск 2025

Содержание

Введение **5**

1 Задачи составления энергетически эффективных расписаний для распараллеливаемых работ **28**

1.1 Постановка задачи и предыдущие исследования 28
1.2 NP-трудность задач 35
1.3 От минимизации длины расписания к минимизации потребления энергии 39
1.3.1 Нижняя оценка 40
1.3.2 Задача с согласованными моментами поступления и директивными сроками 42
1.3.3 Задача с прерываниями без миграции при согласованных моментах поступления и директивных сроках 50
1.3.4 Задача с прерываниями и миграцией 52
1.4 От однопроцессорных работ к многопроцессорным работам 55
1.5 Задача линейного программирования с конфигурациями работ 60
1.5.1 Дискретизация скоростей 60
1.5.2 Модель линейного программирования с конфигурациями работ 62
1.5.3 Отделяющий оракул 64
1.5.4 Одномодальные двухпроцессорные работы 66
1.6 Вычисление нижней оценки и частные случаи 68
1.6.1 Алгоритм для вычисления нижней оценки 69
1.6.2 Общий момент поступления и директивный срок . . 72
1.6.3 Общий момент поступления или директивный срок 74

2 Задачи составления расписаний с ограничением на потребление энергии **77**

2.1 Длина расписания 78
2.1.1 Предыдущие исследования 80
2.1.2 От настраиваемых скоростей к фиксированным скоростям 87

2.1.3 Неравномерное распределение объема работы между процессорами . . . 101
2.1.4 Задача линейного программирования с конфигурациями работ . . . 103
2.1.5 Нижняя оценка на целевую функцию . . . 109
2.2 Сумма моментов завершения . . . 114
2.2.1 Предыдущие исследования . . . 115
2.2.2 Жесткие работы . . . 117
2.2.3 Двухпроцессорный случай . . . 127
2.2.4 Гибкие работы . . . 129
2.2.5 Одномодальные многопроцессорные работы . . . 131
2.2.6 Полиномиально разрешимые случаи . . . 138
2.3 Распараллеливаемые работы . . . 141
2.3.1 Предыдущие исследования . . . 143
2.3.2 Мягкие работы . . . 144
2.3.3 Гибкие работы . . . 146
2.3.4 Гибкие работы с частичным порядком . . . 150
2.3.5 Нижняя оценка на целевую функцию . . . 156
2.4 Задача open-shop . . . 160
2.4.1 Однопроцессорные операции . . . 162
2.4.2 Многопроцессорные операции и ограниченное число стадий . . . 169

3 Задачи составления расписаний и маршрутизации с запретами работ: одностадийные и многостадийные системы 172
3.1 Одностадийные системы . . . 172
3.1.1 Задачи на одной машине с запретами работ . . . 176
3.1.2 Подход к решению . . . 191
3.1.3 Модели целочисленного программирования . . . 196
3.1.4 Задачи на нескольких машинах с запретами работ . . . 200
3.2 Многостадийные системы . . . 201
3.2.1 Задачи job-shop и flow-shop с запретами работ . . . 201
3.2.2 Задача open-shop с запретами работ . . . 219
3.2.3 Подход к решению . . . 221
3.2.4 Модели целочисленного программирования . . . 223

4 Оптимальная рекомбинация в эволюционных алгоритмах для задач на перестановках 228

4.1 Эволюционный алгоритм с оптимальной рекомбинацией . 228

4.2 Задача минимизации суммарного взвешенного временного запаздывания на одной машине 238

4.2.1 Операторы 240

4.2.2 Вычислительный эксперимент 241

4.3 Задача коммивояжера 246

4.3.1 Операторы 247

4.3.2 Вычислительный эксперимент 258

4.4 Задача составления расписаний выполнения заказов клиентов 271

4.4.1 Операторы 273

4.4.2 Вычислительный эксперимент 275

4.5 Задача составления расписаний с учетом ограничения на расход энергии 278

4.5.1 Операторы 278

4.5.2 Вычислительный эксперимент 284

4.6 Задача составления энергетически эффективных расписаний 287

4.6.1 Операторы 289

4.6.2 Вычислительный эксперимент 293

Заключение 296

Литература 299

Введение

Актуальность и степень разработанности темы исследования. В настоящей диссертации исследуются задачи составления расписаний с альтернативами выполнения работ при учете ресурсных и структурных ограничений. Это широкий круг, как правило, NP-трудных задач комбинаторной оптимизации, которые возникают в компьютерных и производственных системах.

Современные компьютерные технологии позволяют не только решать сложные вычислительные задачи, но и порождают новые проблемы оптимального использования вычислительных ресурсов. Например, для выполнения трудоемких вычислений компьютеры могут как использовать несколько процессоров (ядер) одновременно, так и менять скорость вычислений. В многопроцессорных компьютерных системах возникают задачи теории расписаний, характеризующиеся зависимостью длительностей операций-работ от потребления ресурсов, возможностью параллельных вычислений и отношениями предшествования специального вида. Здесь учитываются максимальная допустимая температура, кэш память, потребление энергии, пропускная способность шины данных и др. Выделяют системы с общей памятью, где допускаются прерывания и миграция операций между процессорами, а также системы с распределенной памятью, где имеются более значительные временные затраты на взаимодействие с памятью и ограничения на передачу данных, которые приводят к запретам (возможно частичным) на прерывания. Если процессоры параллельные, то работы могут выполняться на любых из них и задается количество требуемых процессоров. Такие работы называются распараллеливаемыми. В случае специализированных процессоров для работ указываются подмножества процессоров, которые могут использоваться. Такие работы называются многопроцессорными. Составление расписаний для многопроцессорных и распараллеливаемых работ имеет место на практике при тестировании и надежных вычислениях; в параллельных приложениях на видеокартах; компьютерных системах управления; приложениях в компьютерных устройствах, срок службы которых зависит от ограниченной эффективности батареи.

Задачи с энергетическим критерием и с ограничением на расход энергии являются труднорешаемыми даже в простых практически значимых частных случаях [90,115]. При исследовании вычислительной сложности таких задач актуальными являются подходы к построению полиномиальных алгоритмов с гарантированными оценками точности, методы доказательства гарантированных оценок точности и NP-трудности классов примеров, представленные в работах Брукера П. [60,61], Гимади Э.Х. [3], Дроздовского М. [87–90], Кононова А.В. [32,33,177], Лазарева А.А. [28], Севастьянова С.В. [68], Серваха В.В. [1], Хачая М.Ю. [144] и других.

Ряд работ авторов из различных стран посвящен построению оригинальных приближенных алгоритмов для составления расписаний однопроцессорных работ, где учитываются выпуклость функции расчета мгновенной мощности и загрузка процессоров, среди которых можно выделить работы Алберс С. [22], Бамписа Е. и Кононова А.В. [32], Герардса М.Е.Т. [115], Струсевича В.А. и Шакхлевич Н.В. [220]. При доказательствах гарантированных оценок точности предлагаются и выявляются сложные комбинаторные схемы и структурные свойства формируемых расписаний [22, 115]. В большинстве случаев рассматриваются оффлайн и онлайн задачи, где составляются расписания для независимых однопроцессорных работ с заданными моментами поступления и директивными сроками при минимизации расхода энергии, и с заданным общим доступным объемом энергии при минимизации временных критериев. Однако реальная практика предполагает также наличие связей по последовательности выполнения операций, совокупного эффекта от выполнения нескольких операций и учет характеристик системы (см., примеры в [90]). Специфика задач обуславливает необходимость разработки приближенных алгоритмов и метаэвристик, позволяющих быстро находить решения хорошего качества (см., например, [32, 38, 144, 200]).

Представляет интерес выявить те классы задач, где при подходящей выпуклой модели для нижней оценки целевой функции удастся сохранить те же гарантированные оценки точности алгоритмов жадного типа при учете потребления энергии работами, что имеют место в задачах без ресурсных ограничений.

В литературе также представлены результаты по исследованию задач

составления расписаний в производственных системах с оптимизацией расхода энергии на различных уровнях [109,188,216]. Предложены модели частично целочисленного программирования, конструктивные эвристики и метаэвристики. При исследовании и решении структурно сложных задач важно учитывать полиэдральные характеристики этих задач и свойства решений. Такие исследования представлены в работах Бондаренко В.А. [2], Золотых Н.Ю. и Шевченко В.Н. [15], Колоколова А.А. [10], Николаева А.В. [160], Симанчева Р.Ю. [12], Хамисова О.В. [14].

В задачах составления расписаний с выпуклой функцией потребления ресурсов, характерных для промышленного производства [216], длительности работ зависят от потребляемого ресурса по правилу $p_j(R_j) = (W_j/R_j)^{\kappa}$, где $0 < \kappa \leq 1$ — константа; W_j — объём работы j; R_j — объём потребляемого ресурса работой j. Случай $\kappa = 1$ соответствует многим реальным государственным и промышленным операциям, а случай $\kappa = 0{,}5$ – проектированию схем СБИС, где для отдельной задачи произведение площади кремния (ресурса) на квадрат затраченного времени равно константе (рабочей нагрузке).

В диссертации автором исследуются задачи теории расписаний с многопроцессорными и распараллеливаемыми работами как с критерием минимизации расхода энергии, так и с классическими критериями при бюджетных ограничениях на расход энергии. Предлагаются методы доказательства труднорешаемости задач, оригинальные конструктивные алгоритмы с гарантированными оценками точности и подходы к построению оценок целевой функции, выявляются комбинаторные свойства допустимых расписаний.

Еще одно актуальное направление исследований представляют задачи составления расписаний с предписаниями и/или запретами, которые имеют приложения в компьютерных системах и производственных предприятиях, где запреты на выполнение работ в позициях машин возникают из-за технологических ограничений, фиксированных маршрутов и рабочих смен, что представлено в работах Ернста А.Т. [104], Фишера М. [105], Хачая М.Ю. [144] и Черных И.Д. [69]. Например, рассмотрим задачу составления расписаний с маршрутизацией [69]. Задан неориентированный взвешенный граф. Вершины соответствуют узлам транспорт-

ной сети. Работы размещены в узлах сети. Веса рёбер соответствуют временам перемещения между узлами. Машина посещает узел не более одного раза. Работы выполняются машинами в узлах, где они размещены. Исследуются модели, в которых работы связаны отношениями предшествования, имеют несколько стадий и другие отношения/связи. Целью является минимизировать некоторый регулярный критерий. Такие задачи могут решаться в два этапа. Сначала отыскиваются маршруты машин, а затем составляется расписание выполнения работ в позициях маршрутов [68, 69]. Однако, когда маршруты машин фиксированы, то имеет место задача составления расписаний с запретами работ, определяемыми узлами, посещаемыми машинами в позициях маршрутов [68, 69, 105].

В то же время задачи составления расписаний с предписаниями работ играют важную роль в эволюционных алгоритмах (ЭА) [204], где задача оптимальной рекомбинации решается в операторе кроссинговера (рекомбинации). Родительские решения обмениваются своими признаками при построении потомков в операторах кроссинговера [29, 204, 244]. Для произвольного набора родительских решений задача оптимальной рекомбинации состоит в нахождении наилучшего возможного потомка, который может возникнуть в результате кроссинговера, где значение каждого компонента решения-потомка копируется как одно из значений этого компонента в родительских решениях. Такие операторы рекомбинации основаны на свойстве передачи генов [204]. Исследование задачи оптимальной рекомбинации в эволюционных алгоритмах представлено в работах Балаша Е. и Ниехауса В. [29], Еремеева А.В. [95–97], Чикано Ф., Очоа Г. и Витлей Д. [70, 71, 212, 228, 229], Ягиура М. и Ибараки Т. [244] и других.

Ранее задачи составления расписаний с предписаниями или запретами элементов исследовались Пяткиным А.В. в контексте задачи о раскраске графа со списками [203], Балашем Е. для задач булева программирования [29] и Сердюковым А.И. для задачи коммивояжера [11]. В настоящей диссертации проводится исследование алгоритмической сложности задач составления расписаний с предписаниями или запретами работ. Для каждой позиции в очередности выполнения на машине задано под-

множество работ, которые здесь могут выполняться. Предлагается метод доказательства NP-трудности для указанных задач с предписаниями работ, основанный на сводимости специального случая задачи Разбиение. В результате доказывается NP-трудность при различных регулярных критериях и дополнительных условиях на связи между работами в одностадийных и многостадийных системах. Строятся точные алгоритмы для случая мощности предписаний не превосходящей двух, показывается, что «почти все» индивидуальные задачи полиномиально разрешимы. Формулируются модели целочисленного линейного программирования и предлагается алгоритм переборгого типа для общего случая, когда предписания работ могут иметь произвольную структуру и мощность.

Метаэвристики, в частности эволюционные алгоритмы, находят широкое применение при решении практических задач, позволяя получать решения требуемого качества за приемлемое время в случаях, когда классические методы оптимизации оказываются чрезмерно трудоемкими. десь применяются проблемно-ориентированные операторы кроссинговера и мутации (см., например, работы Блума К. [49, 181], Еремеева А.В. [51, 52, 54], Картака В.М. [8]), методы локального поиска с различными типами окрестностей (см., например, результаты исследований Кочетова Ю.А. [146, 157] и Младеновича Н. [186]), и различные варианты настройки параметров эволюционных алгоритмов (см., подходы Еремеева А.В. [49, 52, 54] и Сопова Е.А. [233]). В контексте рассматриваемых в диссертации задач составления расписаний предлагаются адаптивные ЭА с оптимизированными операторами кроссинговера. Здесь необходимо отметить, что одним из перспективных направлений исследований в эволюционных вычислениях является анализ сложности и разработка эффективных алгоритмов решения задачи оптимальной рекомбинации в различных постановках [19, 29, 70, 71, 78, 234]. Важную роль играет учет основных принципов построения операторов кроссинговера и способ представления решений. Экспериментальные результаты показывают перспективность использования оптимизированных операторов в ЭА [19, 29, 70, 71, 78, 234]. В настоящей работе особое внимание уделяется построению оптимизированных операторов скрещивания для задач на перестановках и проводится систематический анализ их вычислитель-

ной сложности. Правильный выбор и реализация указанных операторов позволяют разработать высокоэффективные эвристики для исследуемых задач. Также экспериментальные исследования показывают перспективность использования нескольких операторов, действия которых настраиваются и адаптируются в процессе вычислений. Для обеспечения разнообразия в популяции и настройки параметров используются различные методы адаптации [114, 233].

Таким образом, исследование вычислительной сложности, комбинаторных свойств и разработка новых алгоритмов с установленными свойствами для задач составления расписаний с альтернативами выполнения работ при учете ресурсных ограничений и структурных связей, характерных для современных производственных и компьютерных систем, являются важными для практики задачами, требующими новых подходов в комбинаторной оптимизации.

Целью диссертации является построение новых математических моделей и точных полиномиальных алгоритмов оптимизации для актуальных задач составления расписаний с альтернативами выполнения работ, сложными технологическими связями и ресурсными ограничениями, возникающими на производстве и в компьютерных системах, либо доказательство труднорешаемости таких задач и разработка для NP-трудных задач приближенных алгоритмов с гарантированной оценкой точности.

Задачи исследования. Для достижения цели поставлены следующие задачи:

– анализ вычислительной сложности рассматриваемых задач и выделение практически значимых полиномиально разрешимых классов примеров;

– построение оценок целевых функций и их анализ;

– разработка приближенных алгоритмов с гарантированной оценкой точности, построение серий примеров, где полученные оценки достигаются;

– разработка эффективных алгоритмов приближенного решения, основанных на эволюционных вычислениях и оптимизированных операторах;

– реализация и экспериментальный анализ разработанных эволюци-

онных алгоритмов.

Объектом исследования являются задачи составления расписаний с альтернативами выполнения работ при наличии ресурсных и структурных ограничений.

Предмет исследования – вычислительная сложность и точные и приближенные алгоритмы решения указанных задач.

Методология и методы исследования. В диссертации для исследования оптимизационных моделей использованы методы математического программирования, теории вычислительной сложности, дискретной оптимизации и теории графов. При построении и исследовании алгоритмов использованы методы исследования операций, выпуклого анализа, линейной алгебры и аналитической геометрии, теории вычислительной сложности; методы локального поиска и эволюционных вычислений. Также в диссертации применяются методы построения приближенных алгоритмов и анализа их точности, гибридные методы локального поиска и эволюционных вычислений, разработанные автором.

Научная новизна. Все представленные в диссертации результаты являются новыми:

– Метод доказательства NP-трудности для задач составления расписаний с учетом расхода энергии.

– Двухэтапные подходы к решению задач с учетом расхода энергии и распараллеливания работ, позволяющие построить новые полиномиальные приближенные алгоритмы с гарантированными оценками точности.

– Подходы к построению нижних оценок и отделяющих оракулов для задач с учетом расхода энергии и распараллеливания работ.

– Метод доказательства NP-трудности для задач составления расписаний с предписаниями работ малой мощности.

– Метод решения задач составления расписаний с предписаниями работ произвольной структуры.

– Адаптивные эволюционные алгоритмы с оптимизированными операторами скрещивания.

– Доказательство полиномиальной разрешимости индивидуальных задач оптимальной рекомбинации для «почти всех» пар родительских решений.

Теоретическая и практическая значимость работы. Работа носит в основном теоретический характер. Полученные в ней результаты позволяют понять структуру решений рассматриваемых задач, вычислительную сложность и границы возможностей полиномиальных алгоритмов. Для большинства задач впервые установлена их вычислительная сложность. Для всех рассмотренных NP-трудных задач построены новые приближенные алгоритмы с гарантированными оценками точности.

Отметим, что все задачи, рассматриваемые в диссертации, имеют приложения в области компьютерных технологий и производственных систем. Построенные алгоритмы, за исключением алгоритмов где используется метод эллипсоидов, имеют низкую трудоемкость и могут быть использованы для эффективного построения приближенных решений хорошего качества.

Разработанные алгоритмы для задач составления расписаний в многопроцессорных системах с учетом ресурсов и возможностью распараллеливания операций могут быть использованы при настройке и планировании работы приложений в вычислительных (компьютерных) онлайн системах и при организации работы на серверах в системах организаций для повышения эффективности.

Математические модели и методы, предложенные для задач составления расписаний с запретами и предписаниями работ, могут использоваться в производственных и логистических системах для повышения качества обслуживания и эффективного использования ресурсов при учете важных технологических ограничений и маршрутов транспортировки.

Апробация результатов. Результаты диссертации докладывались и обсуждались на следующих научных мероприятиях:

• Байкальская международная школа-семинар «Методы оптимизации и их приложения» (2014, 2017);

• Всероссийская конференция «Проблемы оптимизации и экономические приложения» (2015);

• Международная азиатская школа-семинар «Проблемы оптимизации сложных систем» (2020);

• Международная конференция «Математика в современном мире» (2017);

• Международная школа-семинар «Проблемы оптимизации и их приложения (OPTA)» (2018);

• Международная конференция «Mathematical Optimization Theory and Operations Research (MOTOR)» (2019-2025);

• Международная конференция «Проблемы теоретической кибернетики» (2023);

• Международная конференция «Optimization and Applications» (2018, 2022-2024);

• Международная конференция «Прикладная математика и фундаментальная информатика» (2023, 2025);

• Международная конференция «Математическое и компьютерное моделирование» (2020, 2022-2025);

• Международная конференция «Дискретные модели в теории управляющих систем» (2024);

• Международная конференция «Numerical Computations: Theory and Algorithms» (2017, 2023);

• Международная конференция «Mathematics of Artificial Intelligence» (2025, 2026);

• Всероссийская конференция с международным участием «Распределённые информационно-вычислительные ресурсы» (2022)

• Всероссийская конференция с международным участием «Параллельные вычислительные технологии» (2024-2025);

• Научный семинар «Математическое моделирование и дискретная оптимизация» (ОФ ИМ СО РАН);

• Научный семинар «Модели и алгоритмы для задач составления расписаний» (ОФ ИМ СО РАН);

• Научный семинар «Математические модели принятия решений» (ИМ СО РАН);

• Научный семинар «Дискретные экстремальные задачи» (ИМ СО РАН).

Личный вклад. Диссертационная работа представляет собой единый цикл исследований автора, объединенных не только предметом, но и методами исследований. В совместных работах соискателю принадлежат либо основные идеи анализа приближенных алгоритмов, включая

доказательства основных лемм и теорем, либо идеи новых приближенных алгоритмов. Некоторые доказательства лемм и теорем выполнены в соавторстве при непосредственном участии соискателя. Конфликт интересов с соавторами отсутствует.

Основные положения, выносимые на защиту, включают совокупность результатов по разработке и анализу точных алгоритмов и приближенных алгоритмов с гарантированными оценками точности, эволюционных алгоритмов с оптимальной рекомбинацией для задач теории расписаний с альтернативами выполнения работ, где учитываются ограничения на ресурсы и возможность распараллеливания операций, а также предписания или запреты работ, результаты по вычислительной сложности для рассматриваемых задач, а именно с указанием пунктов паспорта специальности 1.2.3, которым они соответствуют:

– методы доказательства NP-трудности для задач составления расписаний с учетом распараллеливания работ и расхода энергии и задач теории расписаний с запретами работ (п. 3 – Теория сложности алгоритмов и вычислений);

– двухэтапный метод решения задач составления расписаний с учетом распараллеливания работ и расхода энергии, на основе которого построены точные алгоритмы и приближенные алгоритмы с гарантированными оценками точности для различных критериев и типов работ (п. 3 – Теория сложности алгоритмов и вычислений, п. 9 – Математическая теория исследования операций);

– комбинаторные свойства расписаний для задач с учетом распараллеливания работ и расхода энергии, которые позволили получить гарантированные оценки точности для конструктивных алгоритмов списочного и жадного типов и построить серии примеров, где эти оценки достигаются (п. 3 – Теория сложности алгоритмов и вычислений, п. 9 – Математическая теория исследования операций);

– методы построения нижних оценок на целевую функцию для задач составления расписаний с учетом расхода энергии (п. 8 – Математическое программирование);

– точный метод решения задач составления расписаний с запретами работ произвольной мощности (п. 3 – Теория сложности алгоритмов и

вычислений, п. 9 – Математическая теория исследования операций);

– доказательство NP-трудности задачи оптимальной рекомбинации для ряда задач составления расписаний на перестановках (п. 3 – Теория сложности алгоритмов и вычислений);

– эволюционный алгоритм с оптимальной рекомбинацией для задач на перестановках и его конкурентоспособность, которая показана экспериментально на ряде задач составления расписаний с регулярными временными критериями (п. 3 – Теория сложности алгоритмов и вычислений, п. 9 – Математическая теория исследования операций).

Достоверность и публикации. Все выносимые на защиту результаты достоверны и теоретически обоснованы. По теме диссертации опубликовано 48 работ, в том числе 26 статей: 16 [4–7, 9, 49, 98, 100, 103, 150, 152, 154–156, 249, 252] – в журналах из списка ВАК или включенных в базы данных Scopus, Web of Science, ZBMath и 10 [99, 101, 148, 149, 151, 153, 247, 248, 250, 254] – в трудах конференций (Lecture Notes in Computer Science, Communications in Computer and Information Science).

Структура работы. Диссертация состоит из введения, четырех глав, заключения и списка литературы.

Глава 1 посвящена построению энергетически эффективных расписаний для распараллеливаемых работ. Требуется составить расписание распараллеливаемых работ с учетом расхода энергии на m процессорах с возможностью настройки скоростей процессоров. Множество работ обозначается через $\mathcal{J} = \{1, \ldots, n\}$, а множество процессоров через $\mathcal{M} = \{1, \ldots, m\}$. Каждая работа $j \in \mathcal{J}$ характеризуется моментом поступления r_j, директивным сроком d_j. Кроме того, работа j имеет объем W_j, $j \in \mathcal{J}$. Рассматриваются жесткие работы (англ., rigid jobs, обозначение – $size_j$), гибкие работы (англ., moldable jobs, обозначение – any), мягкие работы (англ., malleable jobs, обозначение – var) и одномодальные многопроцессорные работы (англ., single mode jobs, обозначение – fix_j). Задача исследуется в различных постановках, когда прерывания и миграция запрещаются (системы с распределенной памятью) и когда допускаются (системы с общей памятью). Миграция предполагает возможность перемещения работы с одного процессора на другой.

Если процессор работает со скоростью s, то мгновенная мощность

(потребление энергии в единицу времени) равна s^α. Минимизируется суммарное потребление энергии всеми процессорами. При построении расписания для работ необходимо определить интервалы выполнения, используемые процессоры и скорости. Обозначение задачи – $P|\beta_1, \beta_2, r_j, d_j|E$, где $\beta_1 \in \{size_j,\ any,\ var,\ fix_j\}$ задает тип работ, $\beta_2 \in \{\circ, pmtn, pmtn*\}$ определяет запрет прерываний выполнения работ, наличие прерываний с миграцией и наличие прерываний без миграции, соответственно. Обозначение «$r_j, d_j \uparrow\uparrow$» вместо «$r_j, d_j$» во втором поле будет указывать на согласованность моментов поступления и директивных сроков.

Показано, что большинство доказательств NP-трудности для задач составления расписаний с критерием минимизации максимального временного смещения $L_{\max}$ можно преобразовать в их аналоги с учетом расхода энергии и возможности варьирования скоростей процессоров.

В результате доказаны следующие теоремы.

Теорема 1 Задача $P|size_j, pmtn, r_j, d_j|E$ является NP-трудной в обычном смысле, даже если все работы имеют общий момент поступления, общий директивный срок и идентичные объемы.

Теорема 2 Задача $P|fix_j, pmtn, r_j, d_j|E$ с $|fix_j| \leq 3$ является NP-трудной в сильном смысле, даже если все работы имеют общий момент поступления, общий директивный срок и идентичные объемы.

Классическим критерием в теории расписаний является минимизация длины расписания, и для широкого круга задач с этим критерием было предложено множество алгоритмов (см., например, [90]). Для задачи составления расписаний с минимизацией расхода энергии предлагается подход к вычислению нижней оценки на потребление энергии и предполагаемых длительностей работ. Это позволяет свести задачу с возможностью варьирования скоростей к классической задаче минимизации длины расписания с фиксированными скоростями работ. В результате предложены новые сильно полиномиальные алгоритмы с гарантированными константными оценками точности для следующих случаев:

- случай без прерываний с согласованными директивными сроками, где каждая работа требует не более $\lceil \frac{m}{2} \rceil$ процессоров:

  Теорема 3 Для задач $P|size_j \leq \lceil \frac{m}{2} \rceil, pmtn*, r_j, d_j \uparrow\uparrow |E$ и

$P|size_j \leq \lceil \frac{m}{2} \rceil, r_j, d_j \uparrow\uparrow |E$ можно построить $(3-\varphi_m)^{\alpha-1}$-приближенное решение за время $O(n^4 + n \max\limits_{j \in \mathcal{J}} size_j)$.

- случай без прерываний с общим моментом поступления и/или директивным сроком:

  Теорема 10 Для задач $P|size_j, pmtn*, r_j = r, d_j = d|E$ и $P|size_j, r_j = r, d_j = d|E$ можно построить $\left(2 - \frac{1}{m}\right)^{\alpha-1}$-приближенное расписание за время $O(n^2 + n \max\limits_{j \in \mathcal{J}} size_j)$.

  Теорема 11 Для задач $P|size_j, pmtn*, r_j = r, d_j|E$ и $P|size_j, pmtn*, r_j, d_j = d|E$ можно построить $(3 - \varphi_m)^{\alpha-1}$-приближенное расписание за время $O(n^4 + n \max\limits_{j \in \mathcal{J}} size_j)$.

- случай с прерываниями, согласованными директивными сроками, но без миграции:

  Теорема 4 Для задачи с прерываниями $P|size_j, pmtn*, r_j, d_j \uparrow\uparrow |E$ можно построить $(4 - \varphi_m)^{\alpha-1}$-приближенное расписание за время $O(n^4 + n \max\limits_{j \in \mathcal{J}} size_j)$.

- случай с миграцией:

  Теорема 5 Для задачи с миграцией $P|size_j, pmtn, r_j, d_j|E$ можно построить $\left(2 - \frac{1}{m}\right)^{\alpha-1}$-приближенное расписание за время $O(n^4 + n \max\limits_{j \in \mathcal{J}} size_j)$.

Также предлагается подход на основе идеи преобразования оптимального расписания для однопроцессорных работ в приближенное расписание для распараллеливаемых работ. Используя эту идею, предлагается приближенный алгоритм для задач без прерываний с согласованными директивными сроками и работами идентичного объема, для которых случай соответствующей задачи с однопроцессорными работами полиномиально разрешим.

Теорема 6 Для задач $P|size_j, pmtn*, r_j, d_j \uparrow\uparrow, W_j = W|E$ и $P|size_j, r_j, d_j \uparrow\uparrow, W_j = W|E$ можно построить $3^{\alpha-1}$-приближенное расписание за время $O(\min\{m, n\}n^2)$.

Более того, формулируется задача линейного программирования с конфигурациями работ и предлагается «почти точный» алгоритм для

задачи с миграцией и прерываниями. Алгоритм основан на методе эллипсоидов. Он возвращает решение с аддитивной ошибкой $\varepsilon > 0$ и работает за время, полиномиальное от $1/\varepsilon$ и размера входа. Этот результат обобщает результат [32] для составления расписания однопроцессорных работ на распараллеливаемые и многопроцессорные работы в случае однородной компьютерной системы.

Теорема 7 Для задачи $Pm|size_j, pmtn, r_j, d_j|E$ допустимое расписание с потреблением энергии, не превосходящим $OPT + \varepsilon$, можно построить за время, полиномиальное относительно размера входа индивидуальной задачи и $1/\varepsilon$.

Теорема 8 Для задачи $P|fix_j, pmtn, r_j, d_j|E$ с $|fix_j| = 2$ допустимое расписание с потреблением энергии, не превосходящим $OPT + \varepsilon$, можно построить за время, полиномиальное относительно размера входа индивидуальной задачи и $1/\varepsilon$.

Вторая глава посвящена исследованию задач составления расписаний с ограничением на потребление энергии.

Имеется множество работ $\mathcal{J} = \{1, \ldots, n\}$, которые должны быть выполнены на m процессорах, имеющих возможность работать с различной скоростью. Рассматриваются жесткие работы, гибкие работы, мягкие работы и одномодальные многопроцессорные работы, а также многостадийная система open-shop. Прерывания, миграция между процессорами и частичный порядок могут быть разрешены или запрещены.

Исследуется стандартная однородная модель в контексте настройки скоростей процессоров. Если процессор работает со скоростью s, то потребление энергии в единицу времени равно s^α, где $\alpha > 1$ – заданная константа. Предполагается, что доступно непрерывное множество скоростей процессоров из заданного диапазона. Общий объем работы W_j, $j \in \mathcal{J}$, может быть произвольным образом распределен между используемыми для ее выполнения процессорами. Требуется построить расписание (указав моменты начала и окончания выполнения работ и используемые ими процессоры), при котором минимизируется длина расписания или сумма моментов завершения работ, а суммарное потребление энергии не превосходит заданной величины на бюджет энергии E. Обозначение задачи – $P|\beta_1,\ \beta_2,\ \beta_3,\ energy|\gamma$, где $\beta_1 \in \{size_j,\ any(\delta_j),\ var(\delta_j),\ fix_j\}$ задает

тип работ (величина δ_j указывает верхнюю границу на степень распараллеливания), $\beta_2 \in \{\circ, pmtn, pmtn*\}$ отвечает за отсутствие или наличие прерываний; $\beta_3 \in \{\circ,\ prec\}$ определяет отсутствие или наличие частичного порядка между работами. Критерий оптимизации $\gamma \in \{C_{\max},\ \sum_j C_j\}$.

Строятся приближенные алгоритмы, основанные на двух этапах. На первом этапе вычисляется нижняя оценка на целевую функцию и находятся длительности работ с использованием вспомогательной задачи выпуклого программирования, метода эллипсоидов или условий Каруша-Куна-Таккера. Затем на втором этапе исходная задача преобразуется в классическую задачу без учета расхода энергии и используются алгоритмы списочного типа с назначением числа процессоров для работ с целью построения расписаний с гарантированной константной оценкой точности. Для задачи минимизации длины расписания получены следующие результаты:

- жёсткие работы без прерываний:

  Теорема 12 Для задачи $P|size_j, energy|C_{\max}$ можно построить $\left(2 - \frac{1}{m}\right)$-приближенное расписание за время, полиномиальное от размера входа индивидуальной задачи.

- жёсткие работы с отношениями предшествования и без прерываний:

  Теорема 13 Для задачи $P|size_j, prec, energy|C_{\max}$ с $size_j \leq qm$, $0 < q < 1$, можно построить $\left(\frac{2-q}{1-q}\right) C^*_{max}$-приближенное расписание за время, полиномиальное от размера входа индивидуальной задачи.

- жёсткие работы с прерываниями и произвольными моментами поступления:

  Теорема 14 Для задачи $P|r_j, size_j, pmtn, energy|C_{\max}$ можно построить $\left(2 - \frac{1}{m}\right) C^*_{max}$-приближенное расписание за время, полиномиальное от размера входа индивидуальной задачи.

- одномодальные двухпроцессорные работы без прерываний:

  Теорема 15 Для задачи $P|fix_j, energy|C_{\max}$ с $|fix_j| = 2$ можно

построить 2-приближенное расписание за время, полиномиальное от размера входа индивидуальной задачи.

- мягкие работы:

  Теорема 30 Для задачи $P|var, \delta_j, energy|C_{\max}$ оптимальное расписание можно построить за полиномиальное время.

- гибкие работы:

  Теорема 32 Для задачи $P|any, \delta_j, energy|C_{\max}$ можно построить $\left(2 - \frac{2}{m+1}\right)$-приближенное расписание за полиномиальное время.

- гибкие работы с частичным порядком:

  Теорема 33 Для задачи $P|any, \delta_j, prec, energy|C_{\max}$ можно построить $\left(\frac{3+\sqrt{5}}{2}\right)$-приближенное расписание за полиномиальное время.

Также отдельно рассматривается задача с прерываниями и миграцией для жестких работ и одномодальных многопроцессорных работ. Сначала для случая дискретного набора скоростей процессоров строятся «почти точные» алгоритмы на основе моделей линейного программирования с конфигурациями работ. Приближенный алгоритм для непрерывного множества скоростей процессоров и произвольных интенсивностей потребления энергии строится с помощью алгоритма для дискретного набора скоростей.

Теорема 16 Для задачи $P|size_j, pmtn, energy|C_{\max}$ расписание с потреблением энергии не более $E + \theta$ и длительностью не более $OPT + \varepsilon$ можно построить за время, полиномиальное от m, $\log 1/\varepsilon$, $1/\theta$ и размера входа индивидуальной задачи.

Теорема 17 Для задачи $P|fix_j, pmtn, energy|C_{\max}$ с $|fix_j| = 2$ расписание с потреблением энергии не более $E+\theta$ и длительностью не более $OPT + \varepsilon$ можно построить за время, полиномиальное от $\log 1/\varepsilon$, $1/\theta$ и размера входа индивидуальной задачи.

Разрабатываются двухэтапные приближенные или точные алгоритмы для критерия минимизации суммы моментов завершения. На первом этапе вычисляется нижняя оценка на суммарное время завершения выполнения работ, отыскивается последовательность и длительности работ

с помощью решения вспомогательной задачи выпуклого программирования. Затем, на втором этапе, задача переформулируется в виде классической задачи составления расписаний без возможности варьирования скоростей процессоров и для ее решения применяются алгоритмы списочного типа и более сложные подходы. Общая идея алгоритма списочного типа – всякий раз, когда подмножество процессоров освобождается, алгоритм добавляет в расписание первую работу из списка, которая требует не больше процессоров, чем доступно. В результате доказаны следующие теоремы:

- жесткие работы:

  Теорема 19 Для задачи $P|size_j \leq \frac{m}{2}, energy|\sum C_j$ можно построить 2-приближенное расписание за полиномиальное время при условии, что неубывающему порядку объемов W_j соответствует неубывающий порядок размеров $size_j$.

  Теорема 20 Для задачи $P|size_j, energy|\sum C_j$ можно построить 2-приближенное расписание за полиномиальное время при условии, в что неубывающему порядку объемов $V_j = \frac{W_j}{size_j}$ соответствует неубывающий порядок размеров $size_j$.

  Теорема 21 Для задачи $P2|size_j, energy|\sum C_j$ можно построить 2-приближенное расписание за время $O(n\log n)$.

- гибкие работы:

  Теорема 22 Для задачи $P|any, \delta_j, energy|\sum C_j$ можно построить 2-приближенное расписание за полиномиальное время в случае, когда упорядочение работ по неубыванию объемов W_j соответствует их упорядочению по неубыванию границ на степень распараллеливания δ_j.

- ограниченное число специализированных процессоров:

  Теорема 24 Для задачи $P2|fix_j, energy|\sum C_j$ можно построить $2^{2\alpha-1/\alpha-1}$-приближенное расписание за полиномиальное время.

  Теорема 25 Для задачи $P2|fix_j, pmtn, energy|\sum C_j$ можно построить $2^{\alpha/\alpha-1}$-приближенное расписание за полиномиальное время.

- однопроцессорный случай и полностью распараллеливаемые работы:

  Теорема 27 Оптимальное расписание для задач $1|energy|\sum C_j$ и $P|var, \delta_j = m, energy|\sum C_j$ можно найти за полиномиальное время.

- несовместные работы:

  Теорема 28 Задача $P|fix_j, energy|\sum C_j$ полиномиально разрешима в случае, когда $fix_j \cap fix_i \neq \emptyset$ для всех $i \neq j,\ i, j \in \mathcal{J}$.

  Теорема 29 Задача $P|size_j, energy|\sum C_j$ полиномиально разрешима в случае, когда $size_j > \frac{m}{2}$ для всех $j \in \mathcal{J}$.

Установлена вычислительная сложность рассматриваемых задач, где либо показана полиномиальная разрешимость, либо доказана NP-трудность.

Также исследуется вычислительная сложность и строятся алгоритмы с гарантированной оценкой точности для различных вариантов задачи open-shop с учетом расхода энергии (при обозначении задачи в первом поле указывается буква «O»). Предлагаемые методы решения позволяют обобщить известные результаты для классической задачи open-shop без учета ресурсных ограничений с критерием минимизации длины расписания на случай учета расхода энергии и получить новые результаты.

Двухэтапный подход (вычисляются скорости работ с учетом ограничения на расход энергии и строится расписание для фиксированных длительностей, на котором достигается нижняя оценка на длину расписания) позволяет показать, что:

– задача $O3|energy|C_{\max}$ является NP-трудной;

– задача $O|pmtn, energy|C_{\max}$ является полиномиально разрешимой;

– задача $O2|energy|C_{\max}$ является полиномиально разрешимой.

На основе модели линейного программирования с конфигурациями работ выделяется полиномиально разрешимый случай задачи с прерываниями $O|discr,\ pmtn,\ energy|C_{\max}$ при дискретном множестве скоростей (используется обозначение «$discr$» во втором поле).

Доказана полиномиальная разрешимость задачи open-shop с не более чем тремя стадиями и многопроцессорными операциями. Здесь нижняя

оценка на длину расписания строится через модель выпуклого программирования специального вида. Далее осуществляется переход к полиномиально разрешимой задаче с фиксированными длительностями работ без учета расхода энергии.

В третьей главе рассматривается класс задач составления расписаний и маршрутизации с запретами выполнения работ на определенных этапах в одностадийных и многостадийных системах.

Исследуется задача составления расписаний на перестановках с запретами работ в следующей постановке. Пусть $\mathcal{J}$, $|\mathcal{J}| = n$, – множество работ, а $\mathcal{M}$, $|\mathcal{M}| = m$, – множество машин. Анализируются одностадийные системы, где каждая работа состоит из одной операции, и многостадийные системы типа flow-shop, job-shop и open-shop, где каждая работа состоит из нескольких операций. Обозначим через p_{ij} длительность (время обработки) операции i работы j. Для каждой машины $l \in \mathcal{M}$ задано множество позиций $N_l = \{1, \ldots, n_l\}$, в которые могут быть размещены работы (или их операции). Также задана система подмножеств работ $X = \{X^{i,l}\}_{i \in \mathcal{M},\ l \in N_l}$. $X^{i,l}$ – это подмножество работ (операций), которые могут выполняться в позиции $i \in N_l$ машины $l \in \mathcal{M}$. В результате имеем запреты на выполнение работ в некоторых позициях машин. Цель – назначить работы (или их операции) в позиции машин в соответствии с системой подмножеств работ X так, чтобы полиномиально вычислимый критерий принимал минимальное значение. Такие задачи составления расписаний с дополнительными ограничениями на выполнение работ будем называть задачами с запретами [11] или задачами с предписаниями. Задачи имеют приложения в компьютерных системах и производственных предприятиях, где запреты на выполнение работ в позициях машин возникают из-за технологических ограничений, фиксированных маршрутов [69, 105] и рабочих смен [104]. В то же время такие задачи играют важную роль в эволюционных алгоритмах, где в операторе кроссинговера решается задача оптимальной рекомбинации [29, 204, 244].

Результаты данной главы позволяют получить новые свойства задач составления расписаний с запретами работ для различных критериев и ограничений. Эти свойства гарантируют NP-трудность в худшем случае и полиномиальную разрешимость в «почти всех» случаях для

практически значимых примеров с системами предписаний работ, имеющими мощность не превосходящую двух. Предложен алгоритм решения для произвольных систем предписаний работ и новые модели частичного целочисленного линейного программирования, учитывающие специфику запретов выполнения работ в позициях перестановок машин и установленную связь с задачей перебора совершенных паросочетаний в двудольных графах.

Доказана NP-трудность следующих задач в случае наличия предписаний работ мощности не больше двух:
$1||\sum_j w_j U_j$;
$1|r_j|L_{\max}$;
$1|r_j|\sum_j U_j$;
$1|r_j|\sum_j T_j$;
$1|C_j \le d_j|\sum_j w_j C_j$;
$1|r_j|\sum_j C_j$;
$1|l_{ij}|C_{\max}$;
$1||\sum_j w_j T_j$;
$2||\gamma \in \{C_{\max},\ L_{\max},\ \sum_j U_j,\ \sum_j T_j\}$;
$J2||C_{\max}$;
$J2||\gamma \in \{L_{\max},\ \sum_j U_j,\ \sum_j T_j\}$;
$F3|Miss-Oper|C_{\max}$;
$F3|Miss-Oper|\gamma \in \{L_{\max},\ \sum_j U_j,\ \sum_j T_j\}$;
$F3||C_{\max}$;
$F2||\gamma \in \{L_{\max},\ \sum_j U_j,\ \sum_j T_j\}$;
$F2||\sum_j C_j$;
$O3||\gamma \in \{C_{\max},\ L_{\max},\ \sum_j U_j,\ \sum_j T_j\}$;
$O2|Miss-Oper|\gamma \in \{C_{\max},\ L_{\max}\}$;
$O2|Miss-Oper|\gamma \in \{\sum_j U_j,\ \sum_j T_j,\ \sum_j C_j\}$.

В четвертой главе исследуется теоретически и экспериментально задача оптимальной рекомбинации, возникающая в эволюционных алгоритмах для задач составления расписаний, и показывается целесообразность ее использования наряду с механизмами перезапуска и адаптации.

Задача оптимальной рекомбинации (ЗОР) для задачи на перестанов-

ках формулируется следующим образом. Даны два родительских решения, заданных перестановками $\pi^1 = (\pi^1_1, \ldots, \pi^1_n)$ и $\pi^2 = (\pi^2_1, \ldots, \pi^2_n)$. Требуется найти перестановку π' такую, что:

(I) $\pi'_i = \pi^1_i$ или $\pi'_i = \pi^2_i$ для всех $i = 1, \ldots, n$;

(II) π' имеет оптимальное значение целевой функции среди всех перестановок, удовлетворяющих условию (I).

Адаптивный эволюционный алгоритм с оптимальной рекомбинацией апробируется на классических NP-трудных задачах составления расписаний и маршрутизации (минимизация суммарного взвешенного временного запаздывания на одной машине, задача коммивояжера, составление расписаний выполнения заказов клиентов), где имеется ряд различных известных метаэвристик для сравнения, и на задачах первых двух глав с учетом расхода энергии в виде ограничения или критерия. Алгоритмическая сложность ЗОР исследуется на основе результатов или методов третьей главы для модификаций указанных задач составления расписаний с предписаниями работ в позициях машин (процессоров). Показана целесообразность использования оптимальной рекомбинации и конкурентоспособность эволюционных алгоритмов с ее применением в вычислительном эксперименте на тестовых примерах известных библиотек.

Также применяются механизмы локальной оптимизации при построении начальной популяции. Баланс между локальным поиском и эволюционным поиском настраивается с помощью механизмов перезапуска. ЗОР апробируется в составе популяционного алгоритма локального поиска, где окрестность определяется на базе произвольного подмножества решений как рекомбинация произвольного набора родительских решений. Результаты третьей главы свидетельствуют о труднорешаемости оптимальной рекомбинации для исследуемых задач на перестановках и приводят к необходимости использования алгоритмов приближенного решения, в том числе усеченных версий операторов скрещивания, и адаптивной настройки операторов.

В **заключении** представлены **основные результаты** диссертации.

1. Предложен двухэтапный метод решения задачи составления энергетически эффективных расписаний с распараллеливаемыми и мно-

гопроцессорными работами. На первом этапе вычисляется нижняя оценка целевой функции и оцениваются характеристики работ. На втором этапе реализуется переход к задаче с фиксированными длительностями работ и строится допустимое расписание с гарантированной оценкой точности. Построены серии примеров, на которых оценки точности достигаются. Предложен метод доказательства NP-трудности на основе тех же принципов, что и двухэтапный метод решения.

2. Предложены методы построения нижних оценок на потребление энергии в задачах составления расписаний с распараллеливаемыми и многопроцессорными работами, которые основаны:
   – на сведении релаксации исходной задачи к специального вида задаче о максимальном потоке минимальной стоимости;
   – на представлении исходной задачи как задачи линейного программирования с экспоненциальным числом переменных и полиномиальным числом ограничений и решении ее методом эллипсоидов с использованием полиномиального оракула отсечения.

3. Впервые исследованы вопросы труднорешаемости задачи теории расписаний с многопроцессорными и распараллеливаемыми работами с классическими критериями при бюджетных ограничениях на расход энергии, выделены полиномиально разрешимые и NP-трудные частные случаи.

4. Для задач с ограничением на расход энергии предложены подходы к построению нижних оценок на целевую функцию, которые основаны:
   – на представлении исходной задачи как задачи выпуклого программирования и решении ее с использованием условий Каруша-Куна-Таккера;
   – на представлении исходной задачи как задачи линейного программирования с экспоненциальным числом переменных и полиномиальным числом ограничений и решении ее методом эллипсоидов с использованием полиномиального оракула отсечения.

   При построении допустимых решений применяются либо жадные

алгоритмы, либо методы декомпозиции.

Указанные результаты являются первыми приближенными алгоритмами с гарантированными оценками точности для многопроцессорных и распараллеливаемых работ с учетом расхода энергии и обобщают известные результаты для однопроцессорных работ.

5. Предложен метод доказательства NP-трудности для задач составления расписаний с запретами работ.

6. Разработан точный метод решения для класса задач составления расписаний с предписаниями работ мощности не превосходящей 2, трудоемкость которого полиномиальна для «почти всех» систем предписаний. На основе этого подхода предложены модели целочисленного линейного программирования с меньшим числом переменных, чем дает классический подход. Метод основан на переборе совершенных паросочетаний в специальном двудольном графе и обобщен на случай предписаний произвольной мощности.

7. Предложены эволюционные алгоритмы с оптимизированными операторами, которые показывают конкурентоспособные результаты на тестовых примерах известных библиотек. Исследована вычислительная сложность задачи оптимальной рекомбинации.

8. Экспериментально показана перспективность использования оптимальной рекомбинации, а также ее статистически значимое превосходство над рандомизированными аналогами в составе эволюционных алгоритмов.

9. Разработаны стратегии управления популяцией при использовании оптимизированных операторов. Построены схемы алгоритма локального поиска с применением оптимальной рекомбинации.

Автор выражает благодарность Еремееву Антону Валентиновичу и Кононову Александру Вениаминовичу за научное сотрудничество и полезные советы при работе над материалом диссертации.

Глава 1

Задачи составления энергетически эффективных расписаний для распараллеливаемых работ

Глава 1 посвящена построению энергетически эффективных расписаний для распараллеливаемых работ.

1. Предлагается двухэтапный метод решения задачи составления энергетически эффективных расписаний с распараллеливаемыми работами. На первом этапе вычисляется нижняя оценка целевой функции. На втором этапе реализуется переход к задаче с фиксированными длительностями работ и строится допустимое расписание с гарантированной оценкой точности. Строятся серии примеров, на которых оценки точности достигаются. Предлагается метод доказательства NP-трудности на основе тех же принципов, что и двухэтапный метод решения.

2. Предлагаются методы построения нижних оценок на потребление энергии, которые основаны:
   – на сведении релаксации исходной задачи к задаче о максимальном потоке минимальной стоимости специального вида;
   – на представлении исходной задачи как задачи линейного программирования с экспоненциальным числом переменных и полиномиальным числом ограничений и решении ее методом эллипсоидов с использованием полиномиального оракула отсечения.

Результаты первой главы опубликованы в работах [148–150, 152].

1.1 Постановка задачи и предыдущие исследования

Представляет интерес и важность разработка высокопроизводительных и энергоэффективных вычислительных систем [21,24,46,75,115,220]. Существует два популярных подхода к снижению энергопотребления в

компьютерных системах. Первый подход – это проектирование оборудования с учетом различных режимов работы, где низкое энергопотребление и высокая надежность системы, доступность и удобство использования достигаются за счет проектирования системы. Второй подход – это разработка программного обеспечения с учетом энергопотребления при использованием механизма динамической настройки скорости. Многие современные процессоры позволяют динамически настраивать скорость, а алгоритм планирования может изменять напряжение и частоту во время выполнения операций для оптимизации производительности и энергопотребления. Составление расписаний с учетом энергопотребления на процессорах, допускающих работу с переменной скоростью, активно исследовалось ранее в оффлайн и онлайн постановках с однопроцессорными работами (операциями). Имеет место два основных направления исследований: минимизация энергопотребления при ограничениях на работу системы и оптимизация характиристик расписания при заданном ограничении на энергопотребление. В текущей главе рассматривается задача динамического управления скоростью процессоров для минимизации потребления энергии, обеспечивая при этом желаемое качество обслуживания.

Распараллеливаемые работы требуют более одного процессора в один и тот же момент времени. Параллельные вычисления получают все большую популярность в теории расписаний (см., например, [90]). Компьютерные системы обладают различными видами параллелизма. Энергоэффективное планирование распараллеливаемых работ возникает при тестировании и надежных вычислениях, параллельных приложениях на графических картах, системах управления компьютерами и других.

Работы, допускающие параллелизацию при выполнении, делятся на два основных вида - распараллеливаемые и многопроцессорные.

Типы распараллеливаемых работ [90]:

- фиксированное число требуемых процессоров ($size_j$ – размер работы) – *жесткие* работы (англ., rigid jobs);

- число используемых процессоров выбирается при составлении расписания и не меняется (δ_j – верхняя граница на степень распарал-

леливания) – *гибкие* работы (англ., moldable jobs);

- число используемых процессоров может меняться в процессе выполнения работы (δ_j – верхняя граница на степень распараллеливания) – *мягкие* работы (англ., malleable jobs).

Типы многопроцессорных работ [90]:

- фиксированный набор требуемых процессоров (fix_j) – *одномодальные* работы (англ., single mode jobs);
- альтернативные наборы используемых процессоров (set_j) – *многомодальные* работы (англ., multimode jobs).

Рассматривается задача составления расписаний распараллеливаемых (или многопроцессорных) работ с учетом расхода энергии на m процессорах с возможностью настройки скоростей процессоров. Множество работ обозначается через $\mathcal{J} = \{1, \ldots, n\}$, а множество процессоров через – $\mathcal{M} = \{1, \ldots, m\}$. Каждая работа $j \in \mathcal{J}$ характеризуется моментом поступления r_j, директивным сроком d_j. Кроме того, работа j характеризуется объемом W_j, $j \in \mathcal{J}$ (также рассматривается вариант, когда работа состоит из нескольких операций и для каждой из них задан объем). Задача рассматривается в различных постановках, когда прерывания и миграция запрещаются (системы с распределенной памятью) и когда допускаются (системы с общей памятью). Миграция предполагает возможность перемещения работы с одного процессора на другой.

Расписание является допустимым, если каждый процессор выполняет не более одной работы в любой момент времени, и каждая работа j выполняется на заданном наборе или количестве процессоров в требуемом объеме между моментом поступления и директивным сроком. Работа $j \in \mathcal{J}$ считается активной в интервале $[r_j, d_j)$. В большинстве случаев рассматривается однородная модель учета потребления энергии, где если любой из процессоров работает со скоростью s, то мощность (энергия, потребляемая за единицу времени) составляет s^α, где $\alpha > 1$ — константа [115]. Каждый из m процессоров может менять скорость при выполнении работ. Полагается, что если некоторые процессоры выполняют одну

и ту же работу одновременно, то все эти процессоры функционируют в течение одного и того же периода времени. Задача рассматривается как в случае непрерывного множества скоростей, так и в случае дискретного набора скоростей.

Требуется найти допустимое расписание σ, при котором минимизируется общее потребление энергии $E(\sigma) = \sum_{j \in \mathcal{J}} E_j(s_j)$, где $E_j(s_j)$ – потребление энергии работой работой j при скорости s_j, $j \in \mathcal{J}$.

Задача обозначается через $P|\beta, r_j, d_j|E$, если прерывания не допускаются, и через $P|\beta, pmtn, r_j, d_j|E$, если прерывания допускаются (в случае если миграция запрещена – $P|\beta, pmtn*, r_j, d_j|E$) [32, 90]. Здесь $\beta \in \{size_j,\ var,\ any,\ fix_j,\ set_j\}$ определяет тип работ, где var задает мягкие работы, а any – гибкие работы.

Задачи с согласованными моментами поступления и директивными сроками часто возникают на практике, когда работы поступившие в систему раньше выполняются раньше [22, 67, 115, 122]. Такие случаи соответствуют ситуациям, когда целью является поддержание гарантии справедливого обслуживания с учетом времени ожидания задач (см., например, [33]). Более того, приложения, такие как мультимедиа и телекоммуникации, организованы таким образом, что буферы ввода и воспроизведения работают по правилу *First In First Out* [133]. Формально моменты поступления и директивные сроки называются согласованными, если для любых двух работ i и j неравенство $r_i < r_j$ влечет $d_i \leq d_j$. Данный случай обозначается символом $\uparrow\uparrow$ после d_j во втором поле классического обозначения из трех полей [115].

Yao и др. [246] были одними из первых, кто исследовал задачи составления энергетически эффективных расписаний и разработали точный алгоритм под названием *YDS* для работ с прерываниями на одном процессоре. Реализация YDS из [246] имеет трудоемкость $O(n^3)$, которая позже была улучшена до $O(n^2)$ в [176]. Этот алгоритм часто используется как подпрограмма в других алгоритмах и при доказательстве труднорешаемости задач. Shioura и др. [220] также предложили точный алгоритм трудоемкости $O(n^2)$ для задачи с прерываниями, основанный на связи с задачами с управляемыми длительностями выполнения работ и субмодулярной оптимизацией.

Случай задачи на одном процессоре без прерываний является NP-трудным [27] даже для работ с древовидной структурой интервалов, доступных для выполнения работ. Antoniadis и др. [27] предложили алгоритм с оценкой точности $2^{4\alpha-3}$ для этого случая и алгоритм с оценкой точности $2^{5\alpha-4}$ для общего случая на одном процессоре без прерываний. Bampis и др. [32] разработали алгоритм с оценкой точности $2^{\alpha-1}(1+\varepsilon)^{\alpha}\tilde{B}^{\alpha}$ для последней задачи, где $\tilde{B}_{\alpha}$ – обобщённое число Белла. Cohen-Addad и др. [75] предложили новый вариант ЛП-релаксации для задачи без прерываний на одном процессоре и доказали, что разрыв двойственности не превышает $12^{\alpha-1}$. Они также предложили алгоритм с оценкой точности $(12(1+\varepsilon))^{\alpha-1}$.

В литературе рассматривается несколько вариантов задачи составления энергетически эффективных расписаний для однопроцессорных работ на нескольких процессорах. Они различаются возможностью прерывания и миграции работ. Задача с прерываниями и миграцией полиномиально разрешима, см., например, [21, 24, 34, 46, 220]. Авторы работ [21, 24, 220] представили различные способы сведения этой задачи к вариантам задачи о максимальном потоке. Такие алгоритмы могут применяться как подпрограммы в методах для задач с более сложной структурой, например, для построения нижней оценки на энергопотребление (см. [33, 122]). На текущий момент алгоритм [220] имеет наилучшую трудоемкость среди указанных алгоритмов.

Задача составления расписаний с прерываниями без миграции исследовалась в работах [22, 67, 122] и других. Chen и др. [67] предложили жадный алгоритм с оценкой точности 1.13 для задачи с общим моментом поступления и общим директивным сроком для всех работ. Albers и др. [22] предложили эвристические алгоритмы для задачи без миграции, основанные на назначении работ процессорам на первом этапе и определении оптимальных скоростей и порядка выполнения работ отдельно для каждого процессора на втором этапе. В результате были получены:

- алгоритм с оценкой точности $2\left(2-\frac{1}{m}\right)^{\alpha}$ для задач с общим моментом поступления или общим директивным сроком;
- алгоритмы с оценкой точности $\left(\alpha^{\alpha}2^{4\alpha}\right)$ для задач с единичными объёмами работ и произвольными директивными сроками, или про-

извольными объёмами работ и согласованными директивными сроками и моментами поступления.

Albers и др. [22] также доказали, что задача с единичным объёмами работ полиномиально разрешима для согласованных директивных сроков и моментов поступления, но становится NP-трудной в сильном смысле для произвольных директивных сроков. Greiner и др. [122] предложили общее сведение, преобразующее любое ρ-приближённое расписание для параллельных процессоров с миграцией в $\rho B_{\lceil\alpha\rceil}$-приближенное решение для параллельных процессоров без миграции, где $B_{\lceil\alpha\rceil}$ — $\lceil\alpha\rceil$-е число Белла.

Заметим, что задача без прерываний эквивалентна задаче с прерываниями без миграции в случае согласованных директивных сроков и моментов поступления. Поэтому результаты [22, 67, 122] для согласованных директивных сроков применимы и к задаче без прерываний. В [33] представлен алгоритм с оценкой точности $m^{\alpha}\,(\sqrt[m]{n})^{\alpha-1}$ для общего случая без прерываний, использующий идею преобразования оптимального расписания с прерываниями в расписание без прерываний. Cohen-Addad и др. [75] предложили алгоритм с оценкой точности $(5/2)^{\alpha-1}(1+\varepsilon)^{\alpha}\tilde{B}^{\alpha}(1+\rho)$ для задачи в общей постановке без прерываний, где ρ – отношение максимального и минимального объёмов работ.

В неоднородных системах каждый процессор характеризуется собственной функцией расчета мгновенной мощности, а параметры работ зависят от процессора. Неоднородность позволяет улучшить производительность системы за счёт выполнения каждой работы на наиболее подходящем типе процессоров [58]. Однако для эффективного использования неоднородных архитектур требуются новые алгоритмические подходы. Gupta и др. в [126] исследовали влияние неоднородности на сложность различных задач составления расписаний с учётом расхода энергии.

Ряд исследований посвящён анализу неоднородных задач с возможностью варьирования скоростей процессоров. Для задачи без миграции Bampis и др. [32] предложили алгоритм с оценкой точности $(1+\varepsilon)^{\alpha}\tilde{B}_{\alpha}$. Для случая с возможностью миграции работ разработан алгоритм, гарантирующий построение решения с аддитивной ошибкой ε за время,

полиномиальное от размера входа задачи и $1/\varepsilon$. Позже Albers и др. [23] предложили более быстрый алгоритм на основе отыскания потоков в сетях для задач с миграцией, плотность работ (отношение объема к длине допустимого интервала) в которых ограничена снизу «небольшой» константой. Для задачи без прерываний с зависящими от процессора объёмами работ Cohen-Addad и др. [75] доказали APX-трудность и предложили алгоритм с оценкой точности $2(1+\varepsilon)^{\alpha}5^{\alpha-1}\left(\frac{(1+r)r}{2}\right)^{\alpha}\tilde{B}_{\alpha}$, где r – максимальное отношение между двумя объёмами работ. Кроме того, были разработаны онлайн-алгоритмы для задач составления расписаний в неоднородных многопроцессорных системах с минимизацией суммы взвешенных (или невзвешенных) длительностей интервалов нахождения работ в системе и энергии [124, 125].

Напомним, что мягкими называются работы, которые могут изменять количество используемых процессоров во время выполнения. Энергоэффективное расписание для мягких работ может быть построено за полиномиальное время, как показано в [34]. Задача сначала формулируется как задача выпуклого программирования, где вычисляются длительности работ. Затем известный алгоритм за линейное время генерирует оптимальное решение для мягких работ с полученными длительностями (см., например, [90]).

Дополнительную информацию о составлении расписаний с распараллеливаемыми работами или учетом расхода энергии можно найти в обзорах [22, 115].

Основные результаты главы. В настоящей главе представлено алгоритмическое исследование различных NP-трудных случаев задачи составления энергетически эффективных расписаний с распараллеливаемыми работами.

Классическим критерием в теории расписаний является минимизация длины расписания, и для широкого круга задач с этим критерием было предложено множество алгоритмов (см., например, [90]). Для задачи составления расписаний жёстких работ с минимизацией расхода энергии в диссертации предлагается подход к вычислению нижней оценки на потребление энергии и предварительных длительностей работ. Это позволяет свести задачу с возможностью варьирования скоростей к

классической задаче минимизации длины расписания с фиксированными скоростями работ. В результате строятся новые сильно полиномиальные алгоритмы с гарантированными константными оценками точности для следующих случаев:

- случай без прерываний с согласованными директивными сроками, где каждая работа j требует не более $\lceil \frac{m}{2} \rceil$ процессоров;
- случай с прерываниями, согласованными директивными сроками, но без миграции;
- случай с миграцией.

Также предлагается подход на основе идеи преобразования оптимального расписания для работ на одном процессоре в приближенное расписание для жёстких работ. Используя эту идею, предлагается алгоритм с оценкой точности $3^{\alpha-1}$ для задачи без прерываний с согласованными директивными сроками и работами идентичного объема, для которых случай соответствующей задачи на одном процессоре полиномиально разрешим.

Формулируется задача линейного программирования с конфигурациями работ и предлагается «почти точный» алгоритм для задачи с миграцией и прерываниями. Алгоритм основан на методе эллипсоидов. Он возвращает решение с аддитивной ошибкой $\varepsilon > 0$ и работает за время, полиномиальное от $1/\varepsilon$ и размера входа индивидуальной задачи. Этот результат обобщает результат [32] для составления расписания работ на одном процессоре на распараллеливаемые работы в случае однородной системы.

1.2 NP-трудность задач

Предположим, что параметр α имеет одно значение для всех процессоров и каждая работа требует фиксированное количество процессоров. При таком предположении легко увидеть, что в любом оптимальном расписании любая работа выполняется с постоянной скоростью ввиду выпуклости функции расчета мгновенной мощности. Предположим, что известны скорости работ в оптимальном расписании и необходимо

найти допустимое расписание относительно моментов поступления и директивных сроков. Проверим гипотезу о том, что задачи с возможностью варьирования скоростей и учетом расхода энергии сложнее, чем их аналоги с фиксированными скоростями работ. Действительно, докажем NP-трудность задачи с жесткими работами и возможностью прерываний $P|size_j, pmtn, r_j, d_j|E$.

Теорема 1 *Задача $P|size_j, pmtn, r_j, d_j|E$ является NP-трудной в обычном смысле, даже если все работы имеют общий момент поступления, общий директивный срок и единичные объемы.*

Доказательство. Покажем, что NP-полная задача *Разбиение* [111] полиномиально сводится к распознавательной версии задачи составления расписаний $P|size_j, pmtn, r_j, d_j|E$.

Задача Разбиение: дано множество натуральных чисел $A = \{a_1, a_2, \ldots, a_n\}$. Существует ли подмножество $A' \subset A$ такое, что $\sum\limits_{a_i \in A'} a_i = \sum\limits_{a_i \in A \setminus A'} a_i$?

Пусть $a_1, a_2, \ldots, a_n$ задает индивидуальную задачу Разбиение и $\sum\limits_{a_i \in A} a_i = 2B$. Построим индивидуальную задачу $P|size_j, pmtn, r_j, d_j|E$ с n работами и B процессорами следующим образом. Для каждого a_j создается работа j. Полагается $size_j = a_j$, $W_j = 1$, $r_j = 0$, $d_j = 2$ для $j \in \mathcal{J}$. Требуется определить, существует ли расписание с потреблением энергии не превосходящим $E = 2B$. Из-за выпуклости функции расчета мгновенной мощности и того факта, что $2m = \sum_{j \in \mathcal{J}} (W_j \cdot size_j)$, допустимое расписание с потреблением энергии не превосходящим $2B$ не имеет простоев, а скорость каждого процессора равна 1 в течение всего интервала $[0, 2]$. Поэтому в каждый момент времени t сумма размеров одновременно выполняемых работ должна быть равна $B = \sum_{j \in \mathcal{J}_t} size_j$, где $\mathcal{J}_t$ — множество работ, выполняемых в момент времени t. Поэтому ответ «да» в задаче $P|size_j, pmtn, r_j, d_j|E$ с $E \leq 2B$ гарантирует ответ «да» в задаче Разбиение и наоборот. Распознавательная версия задачи $P|size_j, pmtn, r_j, d_j|E$ является NP-полной, значит исходная оптимизационная версия задачи является NP-трудной. □

В доказательстве теоремы 1 используется общая схема доказательства результатов NP-трудности в теории расписаний. Более того, дока-

зательство почти шаг за шагом повторяет доказательство NP-трудности для задачи $P|size_j, pmtn, r_j, d_j|C_{\max}$ [87]. Утверждается, что большинство доказательств NP-трудности для задач составления расписаний с критерием минимизации максимального временного смещения $L_{\max} = \max_{j \in J}\{C_j - d_j\}$ можно преобразовать в их аналоги с учетом расхода энергии и возможности варьирования скоростей процессоров. Отметим, что $C_{\max}$ является частным случаем $L_{\max}$, когда все работы имеют общий директивный срок.

Рассмотрим распознавательную версию П произвольной задачи составления расписаний с критерием $L_{\max}$: даны n работ и m процессоров. Каждый процессор имеет постоянную скорость. Каждая работа j характеризуется моментом поступления r_j, длительностью p_{ij}, зависящей от процессора, и директивным сроком d_j. Допустимое расписание – это такое назначение работ процессорам во времени, при котором каждая работа j начинается не ранее момента поступления r_j и завершается не позднее директивного срока d_j. Требуется найти допустимое расписание. Здесь не делается акцент на отдельно взятой системе планирования. Можно рассматривать однопроцессорные системы, параллельные идентичные или неоднородные процессоры, задачи поточного или цехового типов и т. д. Можно рассматривать как модели, в которых прерывания допускаются, так и те, в которых прерывания запрещены, а также модели, в которых есть частичный порядок между работами, модели с жесткими или многопроцессорными работами или другие варианты.

Пусть $r_{min} = \min_{j \in J} r_j$ и $d_{max} = \max_{j \in J} d_j$. Будем говорить, что расписание называется расписанием *«без простоев»*, если ни один процессор не простаивает в течение интервала $[r_{min}, d_{max}]$. Будем считать, что индивидуальная задача I характеризуется свойством «без простоев», если все ее допустимые расписания характеризуются свойством «без простоев». Пусть Π' есть подзадача П, которая содержит только индивидуальные задачи со свойством «без простоев».

Сначала покажем, что задача Π' может быть полиномиально сведена к соответствующей задаче с возможностью варьирования скоростей процессоров $\bar{\Pi}$. Действительно, пусть I есть индивидуальная задача Π'. Рассмотрим индивидуальную задачу $\bar{I}$ с возможностью варьирования

скоростей при том же наборе работ и процессоров. Каждая работа j имеет тот же момент поступления r_j и тот же директивный срок d_j, что и в I. Объем W_{ij} работы j на процессоре i равен длительности p_{ij} работы j на процессоре i в индивидуальной задаче I. Индивидуальная задача $\bar{I}$ имеет те же условия на процессоры и работы, что и индивидуальная задача I, за исключением одного – процессоры могут менять свою скорость при выполнении работ. Если процессор работает со скоростью s, то потребление энергии в единицу времени составляет s^α, где $\alpha > 1$ — константа. Таким образом, если процессор i выполняет работу j со скоростью s, то он тратит $\frac{W_{ij}}{s}$ единиц времени на выполнение работы. Требуется определить, существует ли расписание с такими скоростями работ, при которых общее потребление энергии не превосходит $m(d_{max} - r_{min})$. Ввиду выпуклости функции расчета мгновенной мощности и того факта, что все допустимые расписания для индивидуальной задачи I (если таковые имеются) должны быть «без простоев», расписание с требуемым потреблением энергии существует тогда и только тогда, когда существует допустимое расписание для индивидуальной задачи I. Из этого следует, что NP-полнота Π' влечет NP-полноту $\bar{\Pi}$.

Также отметим, что большинство доказательств NP-трудности для задач составления расписаний с критерием минимизации максимального временного смещения основано на полиномиальной сводимости известной NP-полной задачи к распознавательной версии задачи составления расписаний. В большинстве случаев либо полученная распознавательная версия задачи характеризуется свойством «без простоев», либо мы можем легко изменить эту задачу так, чтобы она характеризовалась этим свойством и имела допустимое расписание тогда и только тогда, когда в исходной индивидуальной задаче существует допустимое расписание.

Для иллюстрации предлагаемого метода рассмотрим задачу с одномодальными многопроцессорными работами $P|fix_j, pmtn|C_{\max}$. В [136] доказано, что $P|fix_j, pmtn|C_{\max}$ является NP-трудной в сильном смысле, даже если все работы имеют единичные длительности и $|fix_j| \leq 3$ для каждой работы $j \in \mathcal{J}$. Для сводимости используется задача о дробной вершинной раскраске графа $G = (V, A)$ с дробным хроматическим числом $\chi_f(G) \leqslant 3$. Сводимость осуществляется к распознавательной версии

задачи $P|fix_j, pmtn|C_{\max}$ таким образом, что каждая работа задействует не более трех процессоров и каждый процессор выполняет одну или две работы. Доказано, что расписание длины 3 для соответствующей индивидуальной задачи I из $P|fix_j, pmtn|C_{\max}$ существует тогда и только тогда, когда индивидуальная задача о дробной вершинной раскраске графа имеет дробное хроматическое число $\chi_f(G) \leqslant 3$. В доказательстве для задачи с энергетическим критерием добавляем «фиктивную» однопроцессорную работу для каждого процессора с двумя реальными работами. Очевидно, что расписание длины 3 в новой индивидуальной задаче существует тогда и только тогда, когда существует расписание длины 3 в индивидуальной задаче I. В итоге получаем следующий результат как следствие теоремы 3.1 из [136].

Теорема 2 *Задача $P|fix_j, pmtn, r_j, d_j|E$ с $|fix_j| \leq 3$ является NP-трудной в сильном смысле, даже если все работы имеют общий момент поступления, общий директивный срок и единичный объем.*

1.3 От минимизации длины расписания к минимизации потребления энергии

Рассматривается задача, где работы являются жесткими, то есть для каждой работы $j \in \mathcal{J}$ задано количество требуемых процессоров $size_j$. Отметим, что параметр $size_j$ для работы $j \in \mathcal{J}$ называется ее размером и указывает, что она может выполняться на на любом подмножестве из $size_j$ процессоров. Кроме того, работа j состоит из $size_j$ операций, каждая из которых характеризуется объемом W_{jl}, $l = 1, \ldots, size_j$, и должна быть выполнена на отдельном процессоре. То есть имеет место неравномерное разделение общего объема между процессорами.

Известно, что задача без миграции является NP-трудной в сильном смысле даже в случае однопроцессорных работ (см., например, [22]), а задача с возможностью миграции работ является NP-трудной в обычном смысле для жестких жестких работ, что показано в настоящей диссертации ранее.

Если не учитывать энергопотребление, то задача сводится к классическому случаю, при котором все процессоры работают с некоторой

постоянной скоростью. Основываясь на этом свойстве, здесь предлагаются двухэтапные алгоритмы для нескольких частных случаев задачи с жесткими работами. На первом этапе, строится вспомогательная задача о поиске максимального потока минимальной стоимости, следуя подходу, предложенному для однопроцессорных работ [34, 220]. Решение такой задачи позволяет найти промежуточные длительности работ и нижнюю оценку на потребление энергии. Затем на втором этапе задача сводится к задаче минимизации длины расписания (момент окончания последней работы) при фиксированных скоростях и длительностях, где используется один из алгоритмов списочного типа для построения приближенного решения.

1.3.1 Нижняя оценка

Опишем подход к вычислению нижней оценки на минимальное потребление энергии и предварительных длительностей для жестких работ с произвольными моментами поступления, директивными сроками, объемами и числом используемых процессоров.

Сформулируем специальную задачу поиска максимального потока минимальной стоимости. Интервал $[\min_{j\in J} r_j, \max_{j\in J} d_j)$ разбивается на подынтервалы $I_k = [i_{k-1}, i_k)$, используя моменты поступления r_j и директивные сроки d_j для $j \in \mathcal{J}$ как граничные точки. Обозначим длину интервала I_k через Δ_k, $k = 1, \ldots, \gamma$, где $\gamma \leq 2n-1$. Для работы j обозначим множество доступных интервалов через $\Gamma(j)$, где $\Gamma(j) = \{I_k : \ I_k \subseteq [r_j, d_j]\}$.

Строим двудольный граф (сеть) $G = (V, A)$. Множество вершин есть $V = \{s, t\} \cup \mathcal{J} \cup \mathcal{I}$, где $\mathcal{J}$ — вершины, соответствующие работам, а $\mathcal{I} = \{I_1, \ldots, I_\gamma\}$ — вершины, соответствующие интервалам. Множество дуг A есть $A = A^s \cup A^0 \cup A^t$, где

$$A^s = \{(s, j) : \ j \in \mathcal{J}\},$$

$$A^0 = \{(j, I_k) : \ j \in \mathcal{J}, \ I_k \in \Gamma(j)\},$$

$$A^t = \{(I_k, t) : \ I_k \in \mathcal{I}\}.$$

Таким образом источник s соединен с каждой вершиной-работой, каждая вершина-интервал соединена со стоком t, а каждая вершина-работа j соединена с вершинами-интервалами I_k, соответствующими доступным интервалам $\Gamma(j)$. Пропускные способности дуг задаются следующим образом:

$$\mu(s,j) = +\infty,\ (s,j) \in A^s,$$

$$\mu(j,I_k) = size_j \cdot \triangle_k,\ (j,I_k) \in A^0, \tag{1}$$

$$\mu(I_k,t) = m \cdot \triangle_k,\ (I_k,t) \in A^t. \tag{2}$$

Обозначим через $x(u,v)$ величину потока по дуге (u,v). Отметим, что $e_j = \frac{x(s,j)}{size_j}$ определяет общую длительность работы j, а $e_{j,I_k} = \frac{x(j,I_k)}{size_j}$ определяет длительность работы j в интервале I_k. Пусть стоимость потока $x(s,j)$ задается как $\frac{x(s,j)}{size_j}\sum_{l=1}^{size_j}\left(\frac{W_{jl}size_j}{x(s,j)}\right)^{\alpha}$, что является выпуклой функцией относительно $x(s,j)$. Стоимость потока на всех остальных дугах полагается равной нулю. В итоге формулируется задача поиска максимального потока $s-t$ в $G=(V,A)$, который минимизирует общую стоимость (аналогично задаче с учетом потребления энергии, где длительность умножается на мгновенную мощность).

$$\sum_{j=1}^{n}\frac{x(s,j)}{size_j}\sum_{l=1}^{size_j}\left(\frac{W_{jl}size_j}{x(s,j)}\right)^{\alpha}$$

$$=\sum_{j=1}^{n}\left(\frac{x(s,j)}{size_j}\right)^{1-\alpha}\left(\sum_{l=1}^{size_j}W_{jl}^{\alpha}\right). \tag{3}$$

Поставленная задача поиска максимального потока минимальной стоимости может быть решена за время $O(n^4)$ (см. алгоритм в разделе 1.6.1). Отметим, что имеет место индивидуальная выпуклая функция относительно $x(s,j)$ для работ $j \in \mathcal{J}$. Для вычисления суммы $\left(\sum_{l=1}^{size_j}W_{jl}^{\alpha}\right)$ по всем работам $j \in \mathcal{J}$, требуется время $O(\sum_{j\in\mathcal{J}} size_j)$. (Здесь и далее предполагается, что существует оракул для возведения в степень.) Нетрудно доказать, что любое допустимое расписание определяет допустимый поток с той же стоимостью в графе G. Обратное неверно, как мы увидим в примере 1, но оптимальное решение (3) является нижней оценкой на минимальное потребление энергии. Задача поиска максимального

потока минимальной стоимости гарантирует, что все операции одной и той же работы имеют одинаковую длительность, но их скорости могут быть разными.

Пример 1. Дано m «больших» работ $j = 1, \ldots, m$ с числом требуемых процессоров $size_j = m$ и объемами $W_{jl} = m - 1$, $l = 1, \ldots, m$, и m «малых» работ $j = m + 1, \ldots, 2m$ с числом требуемых процессоров $size_j = 1$ и объемами $W_{j1} = m$. Моменты поступления для всех работ равны 0. Все «малые» работы имеют общий директивный срок m^2, а директивный срок «большой» работы j равен $d_j = mj, j = 1, \ldots, m$.

Рассмотрим следующее оптимальное решение задачи поиска максимального потока минимальной стоимости с единичными скоростями работ и общей стоимостью m^3 (см. рис. 1). Пусть ровно одна «большая» работа j и одна «малая» работа $m + j$ назначены на интервал $I_j := [m(j-1); mj)$, и $x(j, I_j) = m(m-1)$, $x(m+j, I_j) = m$ для $j = 1, \ldots, m$. Общая загрузка каждой пары работ j и $m+j$ составляет m^2, и требуется $2m - 1$ единиц времени для выполнения этих работ со скоростью 1, но имеется только m единиц времени в каждом интервале. Таким образом, допустимое расписание со значением целевой функции m^3, соответствующее единичным скоростям работ, не может быть построено.

Отметим, что здесь можно построить $\left(2 - \frac{1}{m}\right)^{\alpha-1}$-приближенное расписание, увеличивая скорости работ в $\left(2 - \frac{1}{m}\right)$ раз.

1.3.2 Задача с согласованными моментами поступления и директивными сроками

В общем случае отношение целевой функции оптимального расписания без прерываний к нижней оценке (3) может быть очень большим даже в случае одного процессора (см. Предложение 1 в [33]). Построим приближенный алгоритм с гарантированной константной оценкой точности, основанный на использовании нижней оценки (3) и соответствующих длительностей работ, для постановки без прерываний с согласованными моментами поступления и директивными сроками, когда каждая работа j требует не более $\lceil \frac{m}{2} \rceil$ процессоров.

Работа j считается назначенной на интервал I_k, если величина потока

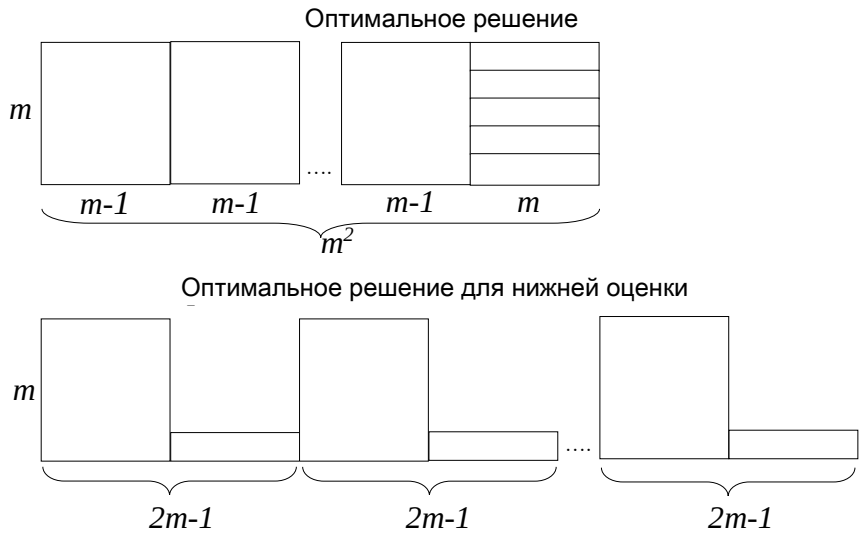


Рисунок 1 — Оптимальное расписание и оптимальное решение для нижней оценки.

$x(j, I_k) > 0$. Обозначим через $\mathcal{A}_{pr}$ оптимальное назначение работ во временные интервалы. Для длительностей e_j и e_{j,I_k} каждой работы $j \in \mathcal{J}$ в назначении $\mathcal{A}_{pr}$, пропускные способности (1) гарантируют, что

$$e_j \leq (d_j - r_j),\ j \in \mathcal{J}, \tag{4}$$

$$e_{j,I_k} \leq \Delta_k,\ j \in \mathcal{J},\ I_k \in \mathcal{I}. \tag{5}$$

Вычислим длительность выполнения работы $j \in \mathcal{J}$ по правилу $p_j = \frac{e_j}{\kappa}$, где $\kappa = \frac{m+\lceil \frac{m-1}{2} \rceil}{\lceil \frac{m+1}{2} \rceil}$. Расписание $\mathcal{S}_{npr}$ без прерываний строится на основе этих длительностей по правилу «Первой идет работа с минимальным директивным сроком» (англ., Earlyest Deadline First, EDF) за время $O(n^2)$. Правило EDF гласит, что текущая работа с минимальным директивным сроком начинается, как только становится свободным необходимое количество процессоров.

Покажем, что построенное расписание $\mathcal{S}_{npr}$ является допустимым, то есть каждая работа j завершается до своего директивного срока.

Пусть работы упорядочены в порядке неубывания моментов поступления (работы с одинаковым моментом поступления упорядочиваются по неубыванию директивных сроков). Обозначим через b_j время начала, а через $c_j = b_j + p_j$ – время завершения работы $j \in \mathcal{J}$ в $\mathcal{S}_{npr}$.

Теперь докажем, что неравенство $c_j \leq d_j$ выполняется для момента

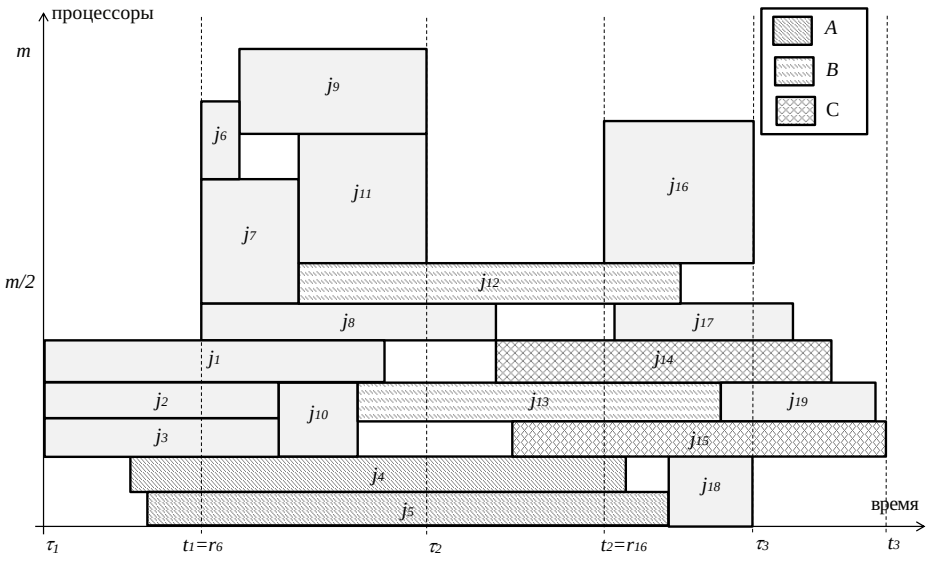


Рисунок 2 — Пример расписания $\mathcal{S}_{npr}$. Здесь $A = \{j_4,\ j_5\}$, $B = \{j_{12},\ j_{13}\}$ и $C = \{j_{14},\ j_{15}\}$ для момента времени t_2.

завершения каждой работы $j \in \mathcal{J}$.

Отметим, что в каждый момент времени в расписании $\mathcal{S}_{npr}$ либо по крайней мере $\lceil \frac{m+1}{2} \rceil$ процессоров заняты, либо по крайней мере $\lceil \frac{m}{2} \rceil$ процессоров простаивают. Разделим расписание $\mathcal{S}_{npr}$ на максимальные по включению интервалы двух типов: «больше половины» и «меньше половины» (см. рис. 2). В каждый момент времени в течение интервала типа «больше половины» по крайней мере $\lceil \frac{m+1}{2} \rceil$ процессоров выполняют некоторые работы. В каждый момент времени в течение интервала типа «меньше половины» по крайней мере $\lceil \frac{m}{2} \rceil$ процессоров простаивают. Пусть l обозначает количество интервалов типа «меньше половины». Обозначим через $[\tau_i, t_i)$, $i = 1, \ldots, l$, интервал типа «меньше половины». Тогда $[t_{i-1}, \tau_i)$, $i = 1, \ldots, l+1$, будет интервалом типа «больше половины». Считается, что $t_0 = 0$ и $\tau_{l+1} = \max_{j \in \mathcal{J}} c_j$. Расписание может начинаться с интервала типа «меньше половины», то есть $t_0 = \tau_1$, или может заканчиваться интервалом типа «меньше половины», то есть $t_l = \tau_{l+1}$. Более того, каждый момент времени t_i соответствует моменту поступления некоторой работы, поскольку все работы требуют не более $\lceil \frac{m}{2} \rceil$ процессоров.

Сначала рассмотрим работу $j \in \mathcal{J}$, которая поступает в интервале типа «меньшего половины» $[\tau_i, t_i)$. Поскольку работы идут на выпол-

нение согласно правилу EDF, то j начинает свое выполнение в момент своего поступления r_j. Поскольку j имеет меньшую длительность в расписании $\mathcal{S}_{npr}$, чем в назначении $\mathcal{A}_{pr}$ $\left(p_j = \frac{e_j}{\kappa} < e_j\right)$, и так как $e_j \leq (d_j - r_j)$, получаем $c_j \leq d_j$.

Теперь рассмотрим работу $j \in \mathcal{J}$, которая поступает в интервале типа «более половины» $[t_i, \tau_{i+1})$. Обозначим через $J_i = \{k \in \mathcal{J} : r_k < t_i\}$ множество работ, которые поступают до момента t_i. Пусть $P_{npr,i}(t)$ обозначает суммарную продолжительность времени, в течение которого работы в J_i выполняются после момента t для каждого $t = t_i, \ldots, t_l$ в $\mathcal{S}_{npr}$, а $E_{pr,i}(t)$ – суммарную продолжительностью времени, в течение которого работы J_i назначаются в интервалы после t в $\mathcal{A}_{pr}$ для каждого $t = t_i, \ldots, t_l$. Эти суммарные продолжительности времени учитывают не только длительности работ, но и количество требуемых процессоров. Если $t = t_i$, то используются обозначения $P_{npr,i}$ и $E_{pr,i}$, соответственно. Утверждается, что неравенство $P_{npr,i} \leq \frac{E_{pr,i}}{\kappa}$ выполняется для каждого $i = 0, \ldots, l$ (см. Лемму 1 ниже).

Как упоминалось ранее, для интервала $[t_i, \tau_{i+1})$ существует работа q с моментом начала, равным $b_q = r_q = t_i$ в расписании $\mathcal{S}_{npr}$.

Поскольку мометы поступления и директивные сроки согласованы, то все работы $q, \ldots, j$ размещаются в интервале $[t_i, d_j)$ в назначении $\mathcal{A}_{pr}$ и общий временной объем $E_{pr,i}$ работ в J_i также используется в течение этого интервала.

Следовательно, неравенства $e_j \leq (d_j - t_i)$ и $E_{pr,i} + \sum_{k=q}^{j} e_k size_k \leq (d_j - t_i)m$ имеют место с учетом задания пропускных способностей дуг (1) и (2). Так как по крайней мере $\lceil \frac{m+1}{2} \rceil$ процессоров заняты в интервале $[t_i, s_j)$ в расписании $\mathcal{S}_{npr}$, то для времени завершения работы j имеем

$$c_j \leq t_i + \frac{P_{npr,i} + \sum_{k=q}^{j-1} p_k size_k}{\lceil \frac{m+1}{2} \rceil} + p_j$$

$$\leq t_i + \frac{E_{pr,i} + \sum_{k=q}^{j-1} e_k size_k}{\lceil \frac{m+1}{2} \rceil \kappa} + \frac{e_j}{\kappa}$$

$$\leq t_i + \frac{E_{pr,i} + \sum\limits_{k=q}^{j} e_k size_k + e_j \lceil \frac{m-1}{2} \rceil}{\lceil \frac{m+1}{2} \rceil \kappa}$$

$$\leq t_i + \frac{m(d_j - t_i) + (d_j - t_i)\lceil \frac{m-1}{2} \rceil}{\lceil \frac{m+1}{2} \rceil \kappa} = d_j. \quad (6)$$

Итак, полученное расписание $\mathcal{S}_{npr}$ допустимо, и каждая работа $j \in \mathcal{J}$ имеет длительность $p_j = \frac{e_j}{\kappa}$ в этом расписании.

Напомним, что оптимальное значение (3) для назначения $\mathcal{A}_{pr}$ является нижней оценкой на потребление энергии в оптимальном расписании без прерываний для рассматриваемой задачи с согласованными моментами поступления и директивными сроками.

Разделив длительность e_j работы j на κ, умножаем скорость каждой операции этой работы на тот же коэффициент. Общее потребление энергии тогда увеличивается на коэффициент $\kappa^{\alpha-1}$. Пусть

$$\varphi_m = \begin{cases} \frac{4}{m+1} & \text{если } m \text{ нечетно,} \\ \frac{6}{m+2} & \text{если } m \text{ четно.} \end{cases}$$

При $\kappa = 3 - \varphi_m$ верна теорема 3.

Теорема 3 *Для задач составления расписаний* $P|size_j \leq \lceil \frac{m}{2} \rceil, r_j, d_j \uparrow\uparrow |E$ *и* $P|size_j \leq \lceil \frac{m}{2} \rceil, pmtn*, r_j, d_j \uparrow\uparrow |E$ *за время* $O(n^4 + n \max\limits_{j \in \mathcal{J}} size_j)$ *может быть найдено* $(3 - \varphi_m)^{\alpha-1}$*-приближенное решение.*

Докажем лемму 1.

Лемма 1 *Неравенство* $P_{npr,i}(t) \leq \frac{E_{pr,i}(t)}{\kappa}$ *имеет место для всех* $i = 0, \ldots, l$ *и* $t = t_i, \ldots, t_l$.

Доказательство. Доказываем индукцией по i.

В базе индукции рассмотрим два случая. Если $t_0 \neq \tau_1$, то $J_0 = \emptyset$ и $P_{npr,0}(t) = E_{pr,0}(t) = 0$.

Если $t_0 = \tau_1$, то расписание начинается с интервала типа «меньше половины», и каждая работа $j \in J_1$ начинается в момент поступления, т. е. $b_j = r_j$.

В этом случае получаем $P_{npr,1}(t) = \sum_{j \in J_1'}(r_j + p_j - t) size_j$ и $E_{pr,1}(t) \geq \sum_{j \in J_1'}(r_j + e_j - t) size_j$ с учетом свойства (5), где $J_1' = \{j \in J_1 : \ c_j > t\}$. Следовательно,

$$E_{pr,1}(t) - \kappa P_{npr,1}(t) \geq \sum_{j \in J_1'}(r_j - t)(1 - \kappa) size_j \geq 0.$$

Предположим, что утверждение леммы 1 верно для i. Докажем, что $P_{npr,i+1}(t) \leq \frac{E_{pr,i+1}(t)}{\kappa}$. Множество работ $J_{i+1} = \{j \in \mathcal{J} : r_j < t_{i+1}\}$ с $c_j > t$ можно разделить на следующие три подмножества (на рис. 2 представлены подмножества A, B и C для $J_{i+1} = J_2$):

1) подмножество работ A с $b_j < t_i$,

2) подмножество работ B с $t_i \leq b_j \leq \tau_{i+1}$,

3) подмножество работ C с $\tau_{i+1} < b_j < t_{i+1}$.

Рассмотрим два случая: (I) $\tau_{i+1} \geq \frac{t_i + (\kappa - 1)t}{\kappa}$ и (II) $\tau_{i+1} < \frac{t_i + (\kappa - 1)t}{\kappa}$.

Случай (I): $\tau_{i+1} \geq \frac{t_i + (\kappa - 1)t}{\kappa}$.

Обозначим через $P_A \leq P_{npr,i}$ общее количество времени, в течение которого работы из A выполняются после момента t_i, и $|A| = \sum_{j \in A} size_j$, $|B| = \sum_{j \in B} size_j$. Поскольку в расписании $\mathcal{S}_{npr}$ нет прерываний, то каждая работа $j \in A$ выполняется в течение всего интервала $[t_i, t)$. В результате:

$$P_{npr,i+1}(t) = (P_A - |A|(t - t_i)) \tag{7}$$

$$+ \sum_{j \in B}(b_j + p_j - t) size_j + \sum_{j \in C}(b_j + p_j - t) size_j. \tag{8}$$

По индукции и с использованием $p_j = \frac{e_j}{\kappa}$, имеем

$$E_{pr,i+1}(t) \geq E_{pr,i} + \sum_{j \in J_{i+1} \setminus J_i} e_j size_j - m(t - t_i) \tag{9}$$

$$\geq \kappa \left(P_{npr,i} + \sum_{j \in J_{i+1} \setminus J_i} p_j size_j \right) - m(t - t_i). \tag{10}$$

Величина $P_{npr,i} + \sum_{j \in J_{i+1} \setminus J_i} p_j size_j$ соответствует общему времени, в течение которого работы из J_{i+1} выполняются после момента t_i в рас-

писании $\mathcal{S}_{npr}$. Для работ из A эта величина равна P_A. Работы из B и C вносят вклад в рассматриваемую величину в размере $\sum_{j\in B\cup C} p_j size_j$. Более того, вклад в $P_{npr,i} + \sum_{j\in J_{i+1}\setminus J_i} p_j size_j$ составляет не менее

$$\left(\left\lceil\frac{m+1}{2}\right\rceil - |A| - |B|\right)(\tau_{i+1} - t_i) + \sum_{j\in B}(b_j - t_i)size_j \tag{11}$$

единиц времени от работ из $J_{i+1}\setminus\{A\cup B\}$, поступивших до момента τ_{i+1}. Действительно, не менее $\lceil\frac{m+1}{2}\rceil$ процессоров используются в интервале $[t_i, \tau_{i+1})$. Однако величины $|A|(\tau_{i+1}-t_i)$ и $|B|(\tau_{i+1}-t_i)-\sum_{j\in B}(b_j-t_i)size_j$ задействуются в этом интервале для работ из A и B, соответственно. Отсюда следует, что

$$E_{pr,i+1}(t) \geq \kappa\left(\left\lceil\frac{m+1}{2}\right\rceil - |A| - |B|\right)(\tau_{i+1} - t_i) \tag{12}$$

$$+\kappa\left(\sum_{j\in B}(b_j + p_j - t_i)size_j + \sum_{j\in C} p_j size_j\right) \tag{13}$$

$$+\kappa P_A - m(t - t_i). \tag{14}$$

Таким образом,

$$\frac{E_{pr,i+1}(t)}{\kappa} - P_{npr,i+1}(t) \geq |A|(t - t_i) - \frac{m(t-t_i)}{\kappa} \tag{15}$$

$$-\sum_{j\in B} size_j(t_i - t) - \sum_{j\in C} size_j(b_j - t) \tag{16}$$

$$+\left(\left\lceil\frac{m+1}{2}\right\rceil - |A| - |B|\right)(\tau_{i+1} - t_i) \tag{17}$$

$$\geq \tau_{i+1}\left(\left\lceil\frac{m+1}{2}\right\rceil - |A| - |B|\right)$$

$$+(|A| + |B|)t - \left\lceil\frac{m+1}{2}\right\rceil t_i - \frac{m(t-t_i)}{\kappa} \tag{18}$$

$$= (|A| + |B|)(t - \tau_{i+1}) + \left\lceil\frac{m+1}{2}\right\rceil(\tau_{i+1} - t_i) - \frac{m(t-t_i)}{\kappa}. \tag{19}$$

Второе неравенство следует из соотношений $b_j < t$ для $j \in C$ и $size_j \geq 1$ для $j \in \mathcal{J}$. Теперь предположим, что $|A| + |B| \geq 1$, и вспомним, что

$\tau_{i+1} \geq \frac{t_i+(\kappa-1)t}{\kappa}$ и $\kappa = \frac{m+\lceil\frac{m-1}{2}\rceil}{\lceil\frac{m+1}{2}\rceil}$. Таким образом, получаем

$$\frac{E_{pr,i+1}(t)}{\kappa} - P_{npr,i+1}(t) \geq t - \left\lceil\frac{m+1}{2}\right\rceil t_i - \frac{m(t-t_i)}{\kappa} \tag{20}$$

$$+ \left\lceil\frac{m-1}{2}\right\rceil \frac{(t_i+(\kappa-1)t)}{\kappa} = 0. \tag{21}$$

Если $|A| = |B| = 0$, то только работы $j \in C \subseteq J_{i+1}$ могут быть выполнены после момента времени t, но в таком случае

$$E_{pr,i+1}(t) \geq \sum_{j\in C}(r_j + e_j - t)size_j, \tag{22}$$

$$P_{npr,i+1}(t) = \sum_{j\in C}(r_j + p_j - t)size_j \tag{23}$$

и, следовательно, $\frac{E_{pr,i+1}(t)}{\kappa} - P_{npr,i+1}(t) \geq 0$.

Случай (II): $\tau_{i+1} < \frac{t_i+(\kappa-1)t}{\kappa}$.

Пусть $\delta_{npr,j}(t) = b_j + p_j - t > 0$ будет временем выполнения работы $j \in A \cup B \cup C$ после момента времени t в расписании $\mathcal{S}_{npr}$. Только работы из A поступают до момента t_i и выполняются после t. Таким образом, имеем

$$\sum_{j\in A}\delta_{npr,j}(t)size_j = P_{npr,i}(t) \leq \frac{E_{pr,i}(t)}{\kappa}. \tag{24}$$

Рассмотрим теперь работу $j \in B$. Предположим, что $e_j \leq t - t_i$, тогда

$$\delta_{npr,j}(t) = b_j + p_j - t \leq \tau_{i+1} + \frac{e_j}{\kappa} - t \tag{25}$$

$$< \frac{(\kappa-1)t + t_i + t - t_i - t\kappa}{\kappa} = 0. \tag{26}$$

Это приводит к противоречию, так как по определению время выполнения $\delta_{npr,j}(t) > 0$.

Следовательно, $e_j > t - t_i$ для $j \in B$.

Пусть $\sigma_{pr,j}(t)$ обозначает длительность j после момента времени t в $\mathcal{A}_{pr}$, где $\sigma_{pr,j}(t) \geq t_i + e_j - t$ с учетом свойств (5) и $r_j \geq t_i$.

Так как $b_j \leq \tau_{i+1}$ для $j \in B$ и $\tau_{i+1} < \frac{(\kappa-1)t+t_i}{\kappa}$, то отсюда следует, что

$$\frac{\sigma_{pr,j}(t)}{\kappa} - \delta_{npr,j}(t) \geq \frac{t_i + e_j - t}{\kappa} - (b_j + p_j - t) \tag{27}$$

$$> \frac{t_i - t}{\kappa} - \frac{(\kappa - 1)t + t_i}{\kappa} + t = 0. \tag{28}$$

Для работы $j \in C$ имеем $\delta_{npr,j}(t) \leq \frac{\sigma_{pr,j}(t)}{\kappa}$, так как эта работа начинается в момент своего поступления r_j. Для завершения доказательства, осталось отметить, что $P_{npr,i+1}(t) \leq \frac{E_{pr,i}(t)}{\kappa} + \frac{1}{\kappa} \sum\limits_{j \in B \cup C} \sigma_{pr,j}(t) size_j \leq \frac{E_{pr,i+1}(t)}{\kappa}$. $\square$

Общий момент поступления или директивный срок. Результат теоремы 3 можно улучшить для жестких работ с общим моментом поступления или директивным сроком. В этом случае нижняя граница на целевую функцию вычисляется на первом этапе за время $O(n\log n + n \max\limits_{j \in \mathcal{J}} size_j)$. Затем, на втором этапе, алгоритм списочного типа (см., например, [191]) позволяет построить $\left(2 - \frac{1}{m}\right)^{\alpha - 1}$-приближенное расписание. Более того, можно построить приближенный алгоритм с двумя подобными этапами для жестких работ с прерываниями, имеющих общий момент поступления или директивный срок. Для работ с произвольным числом требуемых процессоров алгоритм имеет относительную погрешность $(3 - \varphi_m)^{\alpha - 1}$ в случае использования специального списочного алгоритма с прерываниями работ на втором этапе. Описание алгоритмов представлено в разделе 1.6.

1.3.3 Задача с прерываниями без миграции при согласованных моментах поступления и директивных сроках

Рассмотрим задачу с прерываниями без миграции при согласованных моментах поступления и директивных сроках и построим приближенный алгоритм.

Разделим множество работ на два подмножества, первое, в котором каждая работа требует не более $\lceil \frac{m}{2} \rceil$ процессоров, и второе, в котором все работы характеризуются $size_j > \lceil \frac{m}{2} \rceil$. Будем называть работы из первого подмножества *малыми* работами, а работы из второго подмножества — *большими* работами.

Для первой подзадачи только с малыми работами $(3-\varphi_m)^{\alpha-1}$-приближенное расписание строится за время $O(n^4 + n \max\limits_{j\in\mathcal{J}} size_j)$ с помощью алгоритма, предложенного в разделе 1.3.2.

В любом допустимом расписании второй подзадачи с большими работами не более одной работы может выполняться в каждый момент времени. В итоге эта подзадача сводится к задаче с одним процессором с теми же моментами поступления и директивными сроками для работ. Пусть p_j обозначает время пыполнения работы $j \in \mathcal{J}$. Тогда целевая функция задается следующим образом

$$\sum_{j=1}^{n} p_j \sum_{l=1}^{size_j} \left(\frac{W_{jl}}{p_j}\right)^{\alpha} = \sum_{j=1}^{n} p_j^{1-\alpha} \left(\sum_{l=1}^{size_j} W_{jl}^{\alpha}\right).$$

Полученная задача составления расписаний для одного процессора характеризуется целевой функцией, которая состоит из зависящих от работ компонент. Такая задача без прерываний с согласованными моментами поступления и директивными сроками может быть решена за время $O(n^2 + n \max\limits_{j\in\mathcal{J}} size_j)$, как показано в теореме 5 в [220]. Здесь точное решение строится с помощью подхода, аналогичного представленному в радразделе 1.3.1 для вычисления нижней оценки на потребление энергии в случае жестких работ. После этого, размещая каждую большую работу j на первых $size_j$ процессорах в том же интервале, что и в задаче с одним процессором, получаем оптимальное расписание без прерываний для второй подзадачи.

Далее строим недопустимое расписание для всех жестких работ, объединяя решения обеих подзадач. Затем общее расписание разбивается на $O(n)$ непересекающихся интервалов, используя моменты начала и завершения работ в качестве точек разрыва. В каждом интервале на процессоре выполняется не более одной малой работы и не более одной большой работы. Чтобы получить допустимое расписание, делим каждый интервал на два подынтервала в пропорции $1 : (3 - \varphi_m)$. В интервале увеличивается скорость операций большой работы в $4 - \varphi_m$ раза и они размещаются в первом подынтервале. Аналогично увеличивается скорость операций малых работ в $\frac{4-\varphi_m}{3-\varphi_m}$ раза и они размещаются во втором подын-

тервале. В результате получаем допустимое расписание с прерываниями, для которого относительная погрешность составляет $(4-\varphi_m)^{\alpha-1}$. Таким образом, имеет место следующая теорема.

Теорема 4 *Для задачи с прерываниями* $P|size_j, pmtn*, r_j, d_j \uparrow\uparrow |E$ *может быть построено* $(4-\varphi_m)^{\alpha-1}$*-приближенное расписание за время* $O(n^4 + n\max\limits_{j\in\mathcal{J}} size_j)$.

1.3.4 Задача с прерываниями и миграцией

Для задачи в общей постановке с миграцией работ $P|size_j, pmtn, r_j, d_j|E$ при произвольных моментах поступления и директивных сроках предлагается двухэтапный $\left(2-\frac{1}{m}\right)^{\alpha-1}$-приближенный алгоритм, где первый этап такой же, как представлен в разделе 1.3.1.

Пусть I есть произвольный временной интервал для задачи поиска максимального потока минимальной стоимости и его длина равна Δ. Обозначим через $\mathcal{J}' = \{j \in \mathcal{J} :\ x(j,I) > 0\}$ подмножество работ, которые назначаются во временной интервал I согласно решению первого этапа. Определение пропускной способности дуг (1)-(2) предполагает, что $\sum_{j\in\mathcal{J}'} e_{j,I} size_j \le m\Delta$ и $e_{j,I} = \frac{x(j,I)}{size_j} \le \Delta$.

На втором этапе для построения расписания в интервале I используется алгоритм списочного типа без прерываний, который работает следующим образом. Всякий раз, когда подмножество процессоров начинает простаивать, алгоритм ставит в расписание работу, которая требует не больше процессоров, чем доступно, пока все работы из $\mathcal{J}'$ не будут поставлены в расписание. Трудоемкость алгоритма составляет $O(n^2)$.

Утверждается, что длина построенного расписания не превышает величину $\left(2-\frac{1}{m}\right)\Delta$ (см. лемму 5 ниже). Увеличивая скорость операций каждой работы в $\left(2-\frac{1}{m}\right)$ раз, получаем расписание длиной не более Δ. Общее потребление энергии для интервала I в таком случае увеличивается в $\left(2-\frac{1}{m}\right)^{\alpha-1}$ раз. Итоговое расписание строится путем объединения расписаний, найденных для интервалов, поэтому получаем $\left(2-\frac{1}{m}\right)^{\alpha-1}$-приближенное решение исходной задачи составления расписаний с жесткими работами. Заметим, что количество интервалов не превышает $2n-1$ и, следовательно, время выполнения второго этапа $O(n^3)$. В результате

имеем

Теорема 5 *Для задачи составления расписаний с прерываниями и миграцией* $P|size_j, pmtn, r_j, d_j|E$ *можно построить* $\left(2 - \frac{1}{m}\right)^{\alpha-1}$*-приближенное решение за время* $O(n^4 + n \max\limits_{j \in \mathcal{J}} size_j)$.

Используя результаты из [137], можно показать, что оценка точности $\left(2 - \frac{1}{m}\right)^{\alpha-1}$ является достижимой, даже если все работы являются однопроцессорными. Действительно, рассмотрим такое семейство задач, где работы с одинаковым числом требуемых процессоров поступают в порядке неубывания их объемов. Пусть имеется $(m^2 - m)$ работ, имеющих размер $size_j = 1$ и объем $W_j = 1$, и одна однопроцессорная работа объема m. Все работы имеют общий момент поступления $r_j = 0$ и директивный срок $d_j = m$. Оптимальное расписание характеризуется общим потреблением энергии m^2, где скорость каждого процессора равна 1, работа длины m выполняется в интервале $[0, m)$, например, на первом процессоре, а все остальные работы выполняются в произвольном порядке в интервале $[0, m)$ на процессорах $\{2, \ldots, m\}$. Для интервала $[0, m)$ результирующее расписание, полученное с помощью алгоритма списочного типа без прерываний, характеризуется энергопотреблением $\left(2 - \frac{1}{m}\right)^{\alpha-1} m^2$, так как сначала выполняются $(m^2 - m)$ работ объема $W_j = 1$, а скорость каждого процессора равна $\left(2 - \frac{1}{m}\right)$.

Более того, как видно из примера 1 в разделе 1.3.1, оценка точности алгоритма не может быть улучшена, даже если мы будем использовать точный алгоритм для минимизации длины расписания на втором этапе.

Перейдем к доказательству леммы.

Лемма 2 *Дано* m *параллельных процессоров, интервал* $I = [a, a + \Delta)$ *и набор работ* $\mathcal{J}'$ *с длительностями* $e_{j,I} \leq \Delta$ *и размерами* $size_j$*, где*

$$\sum_{j \in \mathcal{J}'} e_{j,I} size_j \leq m\Delta.$$

Длина расписания S*, построенного алгоритмом списочного типа без прерываний, не превышает* $\left(2 - \frac{1}{m}\right) \Delta$.

Доказательство. Пусть T обозначает длину расписания S. Если в любой момент времени в S используется не менее $\lceil \frac{m+1}{2} \rceil$ процессоров, то имеем

$$\Delta \geq \frac{1}{m} \sum_{j \in \mathcal{J}'} e_{j,I} size_j \geq \left\lceil \frac{m+1}{2} \right\rceil \frac{T}{m} \geq \frac{T}{2 - 1/m}.$$

В противном случае предположим, что I' – последний временной интервал в расписании S, в котором используется $m_{I'} < \lceil \frac{m+1}{2} \rceil$ процессоров, задействуемых в течение I'. По построению расписания S существует такая работа j, которая выполняется в течение всего интервала I'. Пусть C_j обозначает время завершения j в S.

Легко видеть, что в каждый момент времени в течение интервала $[a, C_j - e_{j,I})$ расписание S использует не менее $m - size_j + 1 \geq m - m_{I'} + 1$ процессоров (иначе работа j должна быть поставлена в расписание раньше). Более того, не менее $m_{I'}$ процессоров используется в интервале $[C_j - e_{j,I}, C_j)$, поэтому каждая работа, выполняемая в интервале $[C_j, T + a)$, требует не менее $m - m_{I'} + 1$ процессоров. Таким образом, общая загрузка всех процессоров $\sum_{j \in \mathcal{J}'} e_{j,I} size_j$ составляет не менее

$$m_{I'} e_{j,I} + (T - e_{j,I})(m - m_{I'} + 1) \leq \Delta m.$$

Если $e_{j,I} \geq \frac{T}{2 - \frac{1}{m - m_{I'} + 1}}$, то

$$\frac{T}{2 - \frac{1}{m}} \leq \frac{T}{2 - \frac{1}{m - m_{I'} + 1}} \leq e_{j,I} \leq \Delta.$$

В противном случае, поскольку $1 \leq m_{I'} < \lceil \frac{m+1}{2} \rceil$, имеем

$$\begin{aligned} \Delta &\geq \frac{1}{m} \left((m - m_{I'} + 1)T - e_{j,I}(m - 2m_{I'} + 1) \right) \\ &\geq \frac{T}{m} \left(m - m_{I'} + 1 - \frac{m - 2m_{I'} + 1}{2 - \frac{1}{m - m_{I'} + 1}} \right) \\ &= \frac{T(m - m_{I'} + 1)}{(2m - 2m_{I'} + 1)} \geq \frac{mT}{2m - 1} = \frac{T}{2 - \frac{1}{m}}. \end{aligned}$$

□

1.4 От однопроцессорных работ к многопроцессорным работам

В работе [22] показано, что оптимальное расписание без прерываний для однопроцессорных работ с единичными объемами и согласованными моментами поступления и директивными сроками можно найти за полиномиальное время. Предложен точный алгоритм *RR* [22], который сначала упорядочивает работы согласно моментам их поступления, а затем назначает их на процессоры согласно стратегии *Round Robin* (работа номер j размещается на процессоре $j \bmod m$). Далее для каждого процессора, учитывая назначенные работы, строится оптимальное расписание. Задача составления расписаний на одном процессоре с согласованными моментами поступления и директивными сроками может быть решена за время $O(n^2)$ (см., например, [176, 220]).

С помощью полиномиальной сводимости задач Разбиение и 3-Разбиение можно показать, что задача $P|size_j, pmtn*, r_j, d_j|E$ является NP-трудной в обычном смысле, а задача $P|size_j, r_j, d_j|E$ является NP-трудной в сильном смысле, даже если все работы имеют общий момент поступления, общий директивный срок и единичные объемы операций. В этом разделе предлагается модификация алгоритма *RR* для жестких работ без прерываний с согласованными моментами поступления и директивными сроками и единичными объемами операций ($W_{jl} = 1$ для всех $l = 1, \ldots, size_j$ и $j \in \mathcal{J}$), и доказывается, что алгоритм имеет относительную погрешность $3^{\alpha-1}$.

Алгоритм сначала упорядочивает жесткие работы по неубыванию моментов поступления и назначает их процессорам с использованием стратегии Round Robin. После этого отыскивается оптимальное решение для каждого процессора отдельно. Таким образом, получаем *полудопустимое расписание*, в котором каждая работа выполняется между моментом поступления и директивным сроком, но интервалы выполнения работы на разных процессорах могут быть разными. Далее строится допустимое расписание без прерываний для жестких работ из полученных решений для каждого процессора (см. пример на рис. 3). Предлагаемый алгоритм будем называть RR-RIGID (см. Алгоритм 1).

Докажем, что алгоритм RR-RIGID строит допустимое расписание, для которого соблюдаются моменты поступления и директивные сроки работ. Обозначим момент начала и момент завершения работы j на процессоре i в полудопустимом расписании σ_{LB} через $B_{ji}(\sigma_{LB})$ и $C_{ji}(\sigma_{LB})$, соответственно. В расписании σ_{LB} работа j выполняется на процессоре i в интервале $[B_{ji}(\sigma_{LB}), C_{ji}(\sigma_{LB}))$, независимо от ее выполнения на других процессорах. Поскольку расписание σ_{LB} полудопустимо, то имеем $r_j \leq B_{ji}(\sigma_{LB}) < C_{ji}(\sigma_{LB}) \leq d_j$. Работа j состоит из $size_j$ операций, и операции одной и той же работы назначаются процессорам по очереди. Таким образом, первые три шага алгоритма RR-RIGID совпадают с алгоритмом RR, представленным в [22]. Обозначим через O_{ji} операцию работы j, которая обрабатывается на процессоре i. Рассмотрим две работы j_1 и j_2, и пусть $j_1 < j_2$. В [22] показано, что в расписании операции начинаются и заканчиваются в порядке их назначения процессорам согласно правилу Round Robin (см. Утверждение I в [22]). Из этого следует, что любая операция работы j_1 начинается и заканчивается не позднее любой операции работы j_2.

Обозначим момент начала и момент завершения работы j в расписании σ_{RR} через $B_j(\sigma_{RR})$ и $C_j(\sigma_{RR})$, соответственно.

Предположим, что работа j начинает свое выполнение в момент поступления, т. е. $B_j(\sigma_{RR}) = r_j$. Получаем

$$C_j(\sigma_{RR}) = r_j + p_j \leq B_{j\mu(j)}(\sigma_{LB}) + \frac{e_{j\mu(j)}}{3}$$

$$< C_{j\mu(j)}(\sigma_{LB}) \leq d_j.$$

Пусть работа j начинает выполняться после момента поступления, то есть $B_j(\sigma_{RR}) > r_j$. Тогда существует другая работа j_1, которая завершается в момент времени $B_j(\sigma_{RR})$ и она была назначена одному из процессоров, выполняющих работу j. Будем называть работу j_1 *непосредственным предшественником* работы j и обозначать такое отношение предшествования через $j_1 \prec j$. Далее повторяем данную процедуру с работой j_1: либо работа j_1 начинает свое выполнение в момент поступления, либо некий предшественник работы j_1 завершает свое выполнение в момент времени $B_{j_1}(\sigma_{RR}) > r_{j_1}$. В итоге будет построена цепочка ра-

бот $j_k \prec j_{k-1} \prec \cdots \prec j_1 \prec j_0 = j$, где работа j_k начинается в момент времени r_{j_k}. Итак, имеем

$$C_j(\sigma_{RR}) = r_{j_k} + \sum_{l=0}^{k} p_{j_l} = r_{j_k} + \sum_{l=0}^{k} \frac{e_{j_l\mu(j_l)}}{3}.$$

Алгоритм 1 Алгоритм RR-RIGID

1: Упорядочить работы по неубыванию моментов поступления. Работы с идентичными моментами поступления упорядочиваются по неубыванию директивных сроков.
2: Распределить работы по процессорам по правилу Round Robin.
3: Для каждого процессора, учитывая назначенные ему работы, построить оптимальное расписание без прерываний независимо от других процессоров. В результате получаем полудопустимое расписание, которое обозначим через σ_{LB}.
4: Пусть e_{ji} есть время выполнения работы j на процессоре i в σ_{LB}. Не ограничивая общности считаем, что $e_{ji} = 0$, если работа j не выполняется на i. Положим время выполнения работы j равным $p_j = \max\limits_{i=1,\ldots,m} \left(\frac{e_{ji}}{3}\right)$ и идентифицируем «критический» процессор как $\mu(j) = \arg\max\limits_{i=1,\ldots,m} \left(\frac{e_{ji}}{3}\right)$.
5: Составляем расписание для жестких работ в порядке неубывания моментов поступления согласно назначению, полученному на Шаге 2, используя такие скорости, чтобы время выполнения работы j было равно p_j. Получаем расписание σ_{RR}.
6: **Возвращаем в качестве результата** σ_{RR}.

Утверждается, что операции $O_{j_l\mu(j_l)}$ и $O_{j_{l+3}\mu(j_{l+3})}$ не пересекаются ни для каких $l = 0, \ldots, k-3$. Действительно, поскольку $j_{l+2} \prec j_{l+1}$, существует такой процессор i, который выполняет операции работ j_{l+2} и j_{l+1}. Отсюда следует, что $C_{j_{l+2}i}(\sigma_{LB}) \leq B_{j_{l+1}i}(\sigma_{LB})$. Далее, из отношений предшествования $j_{l+3} \prec j_{l+2}$ и $j_{l+1} \prec j_l$ и Утверждения I из [22] получаем $C_{j_{l+3}\mu(j_{l+3})}(\sigma_{LB}) \leq C_{j_{l+2}i}(\sigma_{LB}) \leq B_{j_{l+1}i}(\sigma_{LB}) \leq B_{j_l\mu(j_l)}(\sigma_{LB})$. Следовательно, не более трех операций $O_{j_l\mu(j_l)}$, $l = 0, \ldots, k$, выполняются одновременно в каждый момент времени в σ_{LB} (см., например, прямоугольники с пунктирной границей на рис. 3). Кроме того, все эти операции выпол-

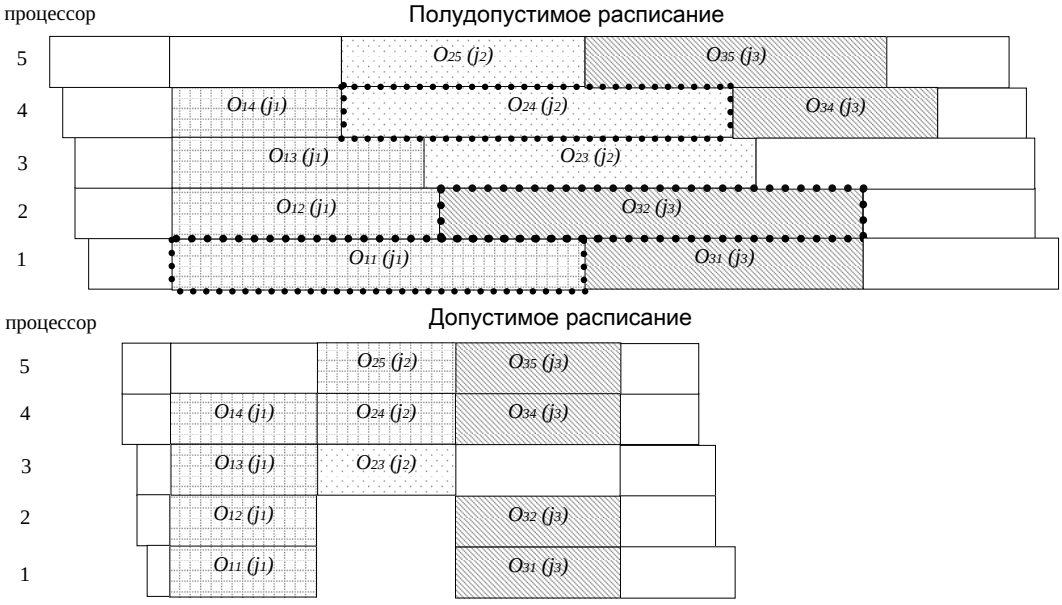


Рисунок 3 — Фрагменты полудопустимого расписания σ_{LB} и допустимого расписания σ_{RR}. Прямоугольники с пунктирной границей демонстрируют интервалы выполнения работ на «критических» процессорах.

няются внутри интервала $[r_{j_k}, d_j)$ в σ_{LB}. Таким образом, имеем

$$C_j(\sigma_{RR}) = r_{j_k} + \sum_{l=0}^{k} \frac{e_{j_l \mu(j_l)}}{3} \leq d_j.$$

Получаем, что σ_{RR} – допустимое расписание.

Поскольку скорости выполнения работ из расписания σ_{LB} увеличиваются не более чем в 3 раза, то общее потребление энергии увеличивается не более чем в $3^{\alpha-1}$ раза. Время работы алгоритма RR-RIGID зависит от реализации шагов 2 и 3. Самый простой способ предполагает хранение для каждого процессора списка назначенных ему работ. В этом случае на шаге 3 решается задача назначения скоростей работам и их упорядочения для каждого процессора отдельно. Как показано в теореме 5 из [220], оптимальное расписание с прерываниями для одного процессора можно найти за время $O(n^2)$. Заметим, что любое расписание с прерываниями для согласованных моментов поступления и директивных сроков можно преобразовать в расписание без прерываний с тем же потреблением энергии, размещая работы в порядке неубывания моментов поступления.

Таким образом, общее время работы алгоритма RR-RIGID составляет $O(mn^2)$. Отметим, что время работы алгоритма зависит от количества процессоров m. Однако возможна более эффективная реализация. Разделим набор процессоров на группы в соответствии с назначением работ. Два процессора принадлежат одной группе тогда и только тогда, когда им назначено одно и то же подмножество работ. Перейдем к описанию деталей реализации.

Имея упорядоченный список работ, назначаем работы процессорам по правилу Round Robin и объединяем процессоры в группы следующим образом.

Изначально имеется одна группа, состоящая из m процессоров, и ни одна работа не размещена на процессорах. После назначения первой работы на первые $size_1$ процессоров будут получены две группы $\{1, \ldots, size_1\}$ и $\{size_1 + 1, \ldots, m\}$.

Для каждой группы храним первый и последний процессор группы, и подмножество работ, назначенных им. Пусть k будет последним процессором, на котором размещена работа $(j-1)$, $j = 2, \ldots, n$. Тогда работа j размещается на процессорах $(k + l) \bmod m,\ l = 1, \ldots, size_j$.

Предположим, что x обозначает номер группы, где k — последний процессор, y — номер группы, содержащей процессор $i = (k+size_j) \bmod m$, а z — номер текущей последней группы. Если i — последний процессор в группе y, то работа j назначается в группы $x + 1, \ldots, y$. В противном случае y разбивается на две группы, используя i в качестве разделяющего процессора, и группы перенумеровываются в порядке возрастания индексов первого процессора в них. Далее работа j назначается в группы $x + 2, \ldots, z + 1$ и $1, \ldots, y$, если $k + size_j > m$, и в группы $x + 1, \ldots, y$ в противном случае.

При назначении очередной работы процессорам образуется не более одной новой группы. Из этого следует, что количество групп не превышает количества работ. В этом случае на Шаге 3 решается задача составления расписаний с учетом расхода энергии для каждой группы процессоров отдельно.

Выбрав лучшую из двух реализаций, получаем время работы алгоритма $O(\min\{m, n\}n^2)$.

Теорема 6 *Алгоритм RR-RIGID является* $3^{\alpha-1}$*-приближенным для задач составления расписаний* $P|size_j, pmtn*, r_j, d_j \uparrow\uparrow, W_j = W|E$ *и* $P|size_j, r_j, d_j \uparrow\uparrow, W_j = W|E$*. Вычислительная сложность алгоритма есть* $O(\min\{m,n\}n^2)$.

1.5 Задача линейного программирования с конфигурациями работ

Построим «почти точный» алгоритм для задачи, где разрешены прерывания и миграция работ между процессорами. Сначала задача формулируется в виде задачи линейного программирования с конфигурациями работ, содержащей экспоненциальное число переменных и полиномиальное числом ограничений. Далее показывается как получить $(OPT + \varepsilon)$-приближенное решение с помощью метода эллипсоидов за время, полиномиальное от размера входных данных и $1/\varepsilon$, где OPT — оптимальное решение, а $\varepsilon > 0$.

1.5.1 Дискретизация скоростей

Под конфигурацией c будем понимать допустимое подмножество $\mathcal{J}_c$ работ, которые могут выполняться одновременно, т. е. $\sum\limits_{j \in \mathcal{J}_c} size_j \leq m$, совместно с назначением значения скорости для каждой операции работы. Обозначим через $\mathcal{C}$ множество всех возможных конфигураций. Любое допустимое расписание содержит ровно одну конфигурацию в каждый момент времени t. Мощность $\mathcal{C}$ не ограничена, поскольку скорости процессоров могут принимать любые действительные значения. Сначала выполним дискретизацию возможных значений скоростей и рассмотрим только конечное число скоростей, с которыми могут работать процессоры.

Для дискретизации скоростей определяем минимальную и максимальную скорости любого процессора в оптимальном расписании. Для отыскания минимальной скорости рассмотрим работу $j \in \mathcal{J}$. Обратим внимание, что длительность j в любом допустимом расписании не превышает $(d_j - r_j)$. Выпуклость функции расчета мгновенной мощности гарантирует, что минимальная скорость каждого процессора больше или равна

минимальной скорости среди всех работ, то есть имеем

$$S_{LB} = \min_{j \in \mathcal{J}} \min_{l=1,\ldots,size_j} \frac{W_{jl}}{d_j - r_j}.$$

Для вычисления максимальной скорости назначим все работы в наименьший по длительности допустимый интервал и получим

$$S_{UB} = \frac{\sum_{j \in \mathcal{J}} \max_{l=1,\ldots,size_j} W_{jl}}{\min_{j \in \mathcal{J}} (d_j - r_j)}.$$

Не ограничивая общности предположим, что все моменты поступления и директивные сроки являются целыми числами и, следовательно, $S_{UB} \leq nW$, где $W = \max_{j \in \mathcal{J}} \max_{l=1,\ldots,size_j} W_{jl}$.

Выберем произвольную константу $\delta > 0$. Рассмотрим только скорости из множества $S_\delta = \{S_{LB}, (1+\delta)S_{LB}, (1+2\delta)S_{LB}, \ldots, (1+k\delta)S_{LB}\}$, где k – наименьшее целое число, такое что $(1+k\delta)S_{LB} \geq S_{UB}$. Следовательно, количество возможных значений для скорости ограничено $k \leq \frac{S_{UB}}{\delta S_{LB}}$, что является линейной функцией от $\frac{1}{\delta}$ и экспоненциальной функцией от размера входных данных.

Положим $\delta = \left(1 + \frac{\epsilon}{2^{\alpha-1}E_{LB}}\right)^{\frac{1}{\alpha}} - 1$, где E_{LB} – нижняя оценка (3) потребления энергии. Это позволит ограничить количество дискретных скоростей экспоненциальной функцией от размера входа задачи и полиномом от $1/\varepsilon$. Действительно, имеем $k \leq \frac{S_{UB}}{\left((1+\varepsilon/(2^{\alpha-1}E_{LB}))^{\frac{1}{\alpha}}-1\right)S_{LB}} \leq \frac{\alpha S_{UB} 2^{\alpha-1} E_{LB}}{\varepsilon S_{LB}}$.

Рассмотрим теперь оптимальное расписание для индивидуальной задачи, и пусть OPT обозначает оптимальное потребление энергии. Построим из оптимального расписания расписание σ, округляя скорости процессоров до ближайших дискретных значений. Отношение потребления энергии любого процессора i в любой момент времени t в σ к потреблению энергии того же процессора i в момент времени t в оптимальном расписании не превышает $(1+\delta)^\alpha = 1 + \frac{\epsilon}{2^{\alpha-1}E_{LB}} \leq 1 + \frac{\epsilon}{OPT}$, где последнее неравенство следует из $OPT \leq 2^{\alpha-1}E_{LB}$ (см. теорему 5). Суммируя значения для всех процессоров и всех временных интервалов, получаем, что энергопотребление в расписании σ не превышает $(1+\delta)^\alpha OPT \leq OPT + \epsilon$. В итоге имеем

Лемма 3 *Для задачи* $P|size_j, pmtn, r_j, d_j|E$ *существует допустимое расписание с общим потреблением энергии не превосходящим* $OPT + \varepsilon$, *где используется конечное (экспоненциальное от размера входа индивидуальной задачи и полиномиальное от* $1/\varepsilon$*) число дискретных скоростей процессоров для любого* $\varepsilon > 0$.

Далее будем рассматривать только расписания, которые удовлетворяют лемме 3.

1.5.2 Модель линейного программирования с конфигурациями работ

Предположим, что моменты времени $t_0 < t_1 < \cdots < t_\gamma$ соответствуют моментам поступления и директивным срокам работ, упорядоченным по возрастанию. Как и в разделе 1.3.1, обозначим множество всех возможных интервалов вида $I_k = [i_{k-1}, i_k)$, $k = 1, \ldots, \gamma$, через $\mathcal{I}$. Пусть Δ_I обозначает длину интервала $I \in \mathcal{I}$. Введем переменную $x_{I,c}$ для каждого $I \in \mathcal{I}$ и $c \in \mathcal{C}$, которая соответствует общему времени в интервале $I \in \mathcal{I}$, где процессоры работают в соответствии с конфигурацией $c \in \mathcal{C}$.

Пусть $S_{j,l,c}$ задает скоростью операции l работы j в соответствии с конфигурацией c. Обозначим через $E_{I,c}$ мгновенную мощность процессоров (потребление энергии в единицу времени), если они работают согласно конфигурации c в течение интервала I, т. е. $E_{I,c} = \sum_{j \in (I,c)} \sum_{l=1}^{size_j} (S_{j,l,c})^\alpha$. Здесь (I, c) – это множество работ, которые активны в интервале I и выполняются на некоторых процессорах согласно конфигурации c. В результате получаем следующую задачу линейного программирования с конфигурациями работ:

$$\sum_{I \in \mathcal{I}, c \in \mathcal{C}} E_{I,c} \cdot x_{I,c} \to \min, \tag{29}$$

$$\sum_{c \in \mathcal{C}} x_{I,c} \le \Delta_I, \ I \in \mathcal{I}, \tag{30}$$

$$\sum_{I,c: j \in (I,c)} \frac{S_{j,l,c} \cdot x_{I,c}}{W_{jl}} \ge 1, \ j \in \mathcal{J}, l = 1, \ldots, size_j, \tag{31}$$

$$x_{I,c} \ge 0, \ I \in \mathcal{I}, c \in \mathcal{C}. \tag{32}$$

Конфигурации, назначенные в интервал I, могут выполняться в произвольном порядке. Неравенство (30) гарантирует, что ровно одна конфигурация выполняется в каждый момент времени $t \in I$ для каждого интервала $I \in \mathcal{I}$. Ограничение (31) задает полное выполнение всех операций работы $j \in \mathcal{J}$.

Вышеуказанная модель имеет экспоненциальное число переменных, но число ее ограничений равно $O(\sum_{j=1}^{n} size_j)$. В [123] показано, что если двойственная задача разрешима методом эллипсоидов с отделяющим оракулом за полиномиальное время, то исходная задача линейного программирования тоже разрешима за полиномиальное время.

Построим двойственную задачу к (29)-(32):

$$\sum_{j \in \mathcal{J}} \sum_{l=1}^{size_j} \lambda_{jl} - \sum_{I \in \mathcal{I}} \mu_I \Delta_I \to \max, \tag{33}$$

$$\sum_{j \in (I,c)} \sum_{l=1}^{size_j} \frac{S_{j,l,c}}{W_{jl}} \lambda_{jl} - \mu_I \leq E_{I,c}, \ I \in \mathcal{I}, \ c \in \mathcal{C}, \tag{34}$$

$$\lambda_{jl} \geq 0, \mu_I \geq 0, \ j \in \mathcal{J}, \ l = 1, \ldots, size_j, \ I \in \mathcal{I}. \tag{35}$$

Двойственная задача линейного программирования имеет $O(\sum_{j=1}^{n} size_j)$ переменных и экспоненциальное число ограничений. В следующем разделе будет представлен полиномиальный отделяющий оракул, то есть будет построен полиномиальный алгоритм, который по заданному решению двойственной задачи определяет, является ли это решение допустимым или, в противном случае, идентифицирует нарушенное ограничение. Из этого следует, что двойственная задача линейного программирования может быть решена методом эллипсоидов с отделяющим оракулом за полиномиальное время (см. теорему 6.4.9 в [123]). Значит можно найти оптимальное решение для исходной задачи линейного программирования, решив ее только с переменными, соответствующими ограничениям, которые были обнаружены нарушенными во время выполнения метода эллипсоидов, и положив значения всех остальных переменных равными нулю. Число этих нарушенных ограничений ограничено числом вызовов отделяющего оракула. В свою очередь, число вызовов отделяющего ора-

кула и число элементарных арифметических операций ограничены полиномом от «сложности граней» ϕ соответствующего многогранника P (см. теорему (6.6.5) в [123]). Многогранник P имеет «сложность граней» не более ϕ, если существует система неравенств с рациональными коэффициентами, имеющая множество решений P, и такая, что длина кодировки каждого неравенства системы не превосходит ϕ. В нашем случае «сложность граней» ϕ может быть ограничена полиномом от длины входа индивидуальной задачи и $\log(1/\varepsilon)$.

1.5.3 Отделяющий оракул

Отделяющий оракул для двойственной задачи линейного программирования работает следующим образом. Для каждого интервала $I \in \mathcal{I}$ проверяется наличие нарушенного ограничения. Напомним, что множество $\mathcal{I}$ содержит $O(n)$ интервалов. Для заданного I достаточно проверить минимум среди значений $\sum_{j\in(I,c)} \sum_{l=1}^{size_j} \left((S_{j,l,c})^\alpha - \frac{S_{j,l,c}}{W_{jl}} \lambda_{jl} \right)$ по всем возможным конфигурациям c. Если это минимальное значение меньше $(-\mu_I)$, то имеется нарушенное ограничение. В противном случае, если ни одного нарушенного ограничения не найдено для всех $I \in \mathcal{I}$, то решение двойственной задачи допустимо.

Для каждой операции l работы $j \in \mathcal{J}$, которая выполняется в интервале I, выражение $(S_{j,l,c})^\alpha - \frac{S_{j,l,c}}{W_{jl}} \lambda_{jl}$ достигает минимальное значение при дискретном значении $v_{jl} \in S_\delta$, которое является одним из двух дискретных скоростей, наиболее близких к значению $\left(\frac{\lambda_{jl}}{W_{jl} \cdot \alpha} \right)^{1/(\alpha-1)}$. Отметим, что минимизируется выпуклая функция одной переменной по набору возможных дискретных значений. Значение $\left(\frac{\lambda_{jl}}{W_{jl} \cdot \alpha} \right)^{1/(\alpha-1)}$ получается при минимизации $(S_{j,l,c})^\alpha - \frac{S_{j,l,c}}{W_{jl}} \lambda_{jl}$, если нет дискретизации скоростей, и оно вычисляется приравниванием производной последнего выражения к значению ноль. Для операции l работы j дискретное значение $v_{jl} \in S_\delta$ отыскивается бинарным поиском за время полиномиальное относительно $1/\varepsilon$ и размера входа задачи. Следовательно, для заданного интервала

I, требуется найти конфигурацию c, при которой минимизируется

$$\sum_{j\in(I,c)} \sum_{l=1}^{size_j} \left((v_{j,l})^\alpha - \frac{v_{j,l}}{W_{jl}} \lambda_{jl} \right). \tag{36}$$

Задача минимизации выражения (36) сводится к задаче о максимальном рюкзаке и может быть решена за время $O(mn + n \max\limits_{j\in\mathcal{J}} size_j)$ с помощью следующего алгоритма динамического программирования, основанного на подходе, предложенном в [134]. Здесь $m \leq \sum\limits_{j=1}^{n} size_j$, так как в противном случае задача $P|size_j, pmtn, r_j, d_j|E$ тривиальна. (Отметим, что $size_j,\ j \in \mathcal{J}$, является частью входных данных.)

Обозначим подмножество работ, которые доступны в интервале I, через $\mathcal{J}' = \{j'_1, \ldots, j'_{n'}\}$ и пусть $V(j) = \sum_{l=1}^{size_j} \left(\frac{v_{j,l}}{W_{jl}} \lambda_{jl} - (v_{j,l})^\alpha \right)$ задает вес работы $j \in \mathcal{J}'$. Пусть F_{ik} обозначает множество всех подмножеств работ из $\{j'_1, \ldots, j'_k\}$, которые могут выполняться одновременно на процессорах из $\{1, \ldots, i\}$ и пусть

$$F(i,k) = \max\{ \sum_{j\in f_{ik}} V(j) : f_{ik} \in F_{ik} \}.$$

Тогда задача проверки на допустимость сводится к вычислению $\max\{F(i,n') :\ i = 1, \ldots, m\}$ для каждого интервала $I \in \mathcal{I}$, что можно сделать с помощью рекурсии:

$$F(0,k) = F(i,0) = 0,\ i = 1, \ldots, m,\ k = 0, 1, \ldots, n';$$

$$F(i,k) = F(i,k-1),\ i = 1, \ldots, size_{j'_k} - 1,\ k = 1, \ldots, n';$$

$$F(i,k) = \max\{F(i,k-1),\ F(i - size_{j'_k}, k-1) + V(j'_k)\},$$

$$i = size_{j'_k}, \ldots, m,\ k = 1, \ldots, n'.$$

Следовательно, существует отделяющий оракул для двойственной задачи, который может быть выполнен за время, полиномиальное относительно $1/\varepsilon$ и размера входа задачи. Таким образом, имеем полиномиальный алгоритм.

Теорема 7 *Для задачи $P|size_j, pmtn, r_j, d_j|E$ допустимое расписание с*

потреблением энергии не превосходящим $OPT + \varepsilon$ *может быть найдено за время, полиномиальное относительно размера входа индивидуальной задачи и* $1/\varepsilon$.

Представленный алгоритм можно обобщить на неоднородную задачу $P|size_j, pmtn, r_j, d_j, \alpha_i|E$, где каждый процессор характеризуется индивидуальной функцией мощности. Единственным изменением в модели линейного программирования с конфигурациями работ для этой задачи является величина $E_{I,c}$, а именно $E_{I,c} = \sum_{j \in (I,c)} \sum_{l=1}^{size_j} (S_{j,l,c})^{\alpha_{i(j,l,c)}}$. Здесь $i(j,l,c)$ – это процессор, на который назначена операция l жесткой работы j в конфигурации c.

Отделяющий оракул для двойственной задачи линейного программирования сводится к минимизации

$$\sum_{j \in (I,c)} \sum_{l=1}^{size_j} \left((v_{j,l})^{\alpha_{i(j,l,c)}} - \frac{v_{j,l}}{W_{jl}} \lambda_{jl} \right). \tag{37}$$

Для каждой операции работы j «оптимальная» дискретная скорость $v_{jl} \in S_\delta$ также может быть найдена с помощью бинарного поиска, потому что каждая компонента суммы (37) является функцией одной переменной с единственным минимумом на множестве положительных значений. Следовательно, имеет место следующий результат.

Следствие 1 *Для задачи* $P|size_j, pmtn, r_j, d_j, \alpha_i|E$ *допустимое расписание с общим потреблением энергии не превосходящим* $OPT + \varepsilon$ *может быть найдено за время, полиномиальное относительно размера входа индивидуальной задачи и* $1/\varepsilon$.

1.5.4 Одномодальные двухпроцессорные работы

Рассмотрим задачу с одномодальными многопроцессорными работами ($fix_j,\ j \in J$). Модель линейного программирования с конфигурациями работ имеет для нее тот же вид, что и для задачи с жесткими работами за исключением суммирований по fix_j вместо $size_j$. Задача минимизации выражения (36) для задачи с одномодальными двухпроцессорынми работами (то есть $|fix_j| = 2$ для $j \in \mathcal{J}$) сводится к задаче

поиска максимального взвешенного паросочетания в мультиграфе, вершины которого соответствуют процессорам, а ребра – работам. В граф добавляется ребро $\{i, i'\}$ с весом, равным $\left(\frac{v_j}{W_j}\lambda_j - v_j^{\alpha}|fix_j|\right)$ для каждой работы $j \in \mathcal{J}$ доступной в интервале I, требуемый набор процессоров которой состоит из i и i' (то есть $fix_j = \{i, i'\}$). Максимальное взвешенное паросочетание в таком мультиграфе можно найти за время, полиномиальное от n, и оно определяет конфигурацию c, которая минимизирует (36).

Следовательно, существует отделяющий оракул для двойственной задачи, которая имеет трудоемкость, полиномиальную от $1/\varepsilon$ и размера входа задачи.

Теорема 8 *Для задачи $P|fix_j, pmtn, r_j, d_j|E$ с $|fix_j| = 2$ можно построить допустимое расписание с потреблением энергии не превосходящим $OPT + \varepsilon$ за время, полиномиальное от $1/\varepsilon$ и размера входа индивидуальной задачи.*

Как и в случае жестких работ, представленный подход можно обобщить на случай задачи $P|fix_j, pmtn, r_j, d_j, \alpha_i|E$ с $|fix_j| = 2$. Таким образом, имеем

Следствие 2 *Для задачи $P|fix_j, pmtn, r_j, d_j, \alpha_i|E$ с $|fix_j| = 2$ можно построить допустимое расписание с потреблением энергии не превосходящим $OPT + \varepsilon$ за время, полиномиальное от $1/\varepsilon$ и размера входа индивидуальной задачи.*

Сложность отделяющего оракула меняется, когда работы задействуют по три процессора. В этом случае задача минимизации выражения (36) является NP-трудной, поскольку к ней сводится NP-полная задача о трехмерном сочетании.

Теорема 9 *Задача минимизации выражения (36) для двойственной задачи линейного программирования является NP-трудной в случае $P|fix_j, pmtn, r_j, d_j|E$ с $|fix_j| = 3$.*

Доказательство. Задача о трехмерном сочетании формулируется следующим образом: задано множество $M \subseteq X \times Y \times Z$, где X, Y, и Z –

непересекающиеся множества с одинаковым числом элементов q. Вопрос в том, содержит ли M сочетание, т. е. подмножество $M' \subseteq M$ такое, что $|M'| = q$ и никакие два элемента M' не совпадают ни по одной координате.

Для индивидуальной задачи о трехмерном сочетании построим следующую индивидуальную задачу $P|fix_j, pmtn, r_j, d_j|E$. Положим $n = |M|$, $m = 3q$, $W_j = 1$, $r_j = 0$, $d_j = 1$ и $|fix_j| = 3$ для всех $j \in \mathcal{J}$. Предположим, что каждый элемент $l = (x, y, z)$ из M соответствует работе j, а координаты l задают требуемые процессоры для работы j, то есть $fix_j = \{x, y, z\}$.

Отделяющий оракул для построенной индивидуальной задачи сводится к поиску максимального значения $\sum_{j\in(I,c)} (S_{j,c}\lambda_j - 3S_{j,c}^{\alpha})$ для единственного интервала $I = (0, 1]$ по всем возможным конфигурациям c. Пусть λ_j будут идентичными для всех работ $j \in \mathcal{J}$ и равны $\lambda > 0$, тогда «оптимальные» дискретные скорости $v_j \in S_\delta$ будут также идентичными для всех работ. Обозначим такую скорость через v. Следовательно, требуется найти конфигурацию c, которая максимизирует

$$\sum_{j\in(I,c)} (v\lambda - 3v^{\alpha}). \tag{38}$$

Отметим, что в каждый момент времени может выполняться не более q работ. Следовательно, конфигурация c со свойством $\sum_{j\in(I,c)} (v\lambda - 3v^{\alpha}) \geqslant q(v\lambda - 3v^{\alpha})$ существует и содержит ровно q работ тогда и только тогда, когда задача о трехмерном сочетании имеет ответ «да». Построенная сводимость полиномиальна, а, следовательно, теорема доказана. □

1.6 Вычисление нижней оценки и частные случаи

Опишем точный алгоритм вычисления нижней оценки на потребление энергии, а также приближенные алгоритмы для частных случаев задачи составления расписаний жестких работ, которые имеют общий момент поступления и/или общий директивный срок.

1.6.1 Алгоритм для вычисления нижней оценки

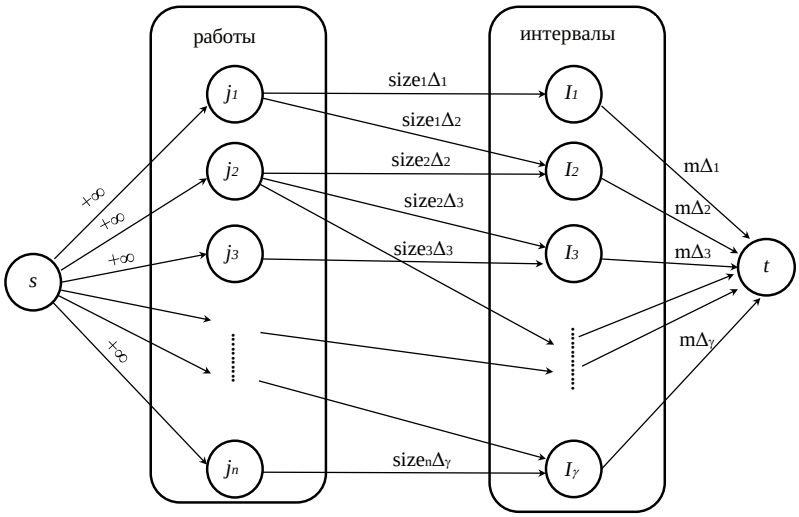


Рисунок 4 — Пример графа $G = (V, A)$.

Используя подход из [220], предлагается точный алгоритм для поиска максимального потока в двудольном графе $G = (V, A)$, представленном в разделе 1.3.1 (см. рис. 4), при котором минимизируется общая стоимость

$$\sum_{j=1}^{n} \frac{x(s,j)}{size_j} \sum_{l=1}^{size_j} \left(\frac{W_{jl} size_j}{x(s,j)} \right)^{\alpha}.$$

Сначала переформулируем задачу в терминах субмодулярной оптимизации. Определим многогранник

$$Q = \{q \in \mathcal{R}^n : \text{ существует допустимый s} - \text{t поток } x$$

$$\text{в } G = (V, A) \text{ с } x(s,j) = q_j \text{ для } j = 1, \ldots, n\}.$$

Пусть $q(X) := \sum_{j \in X} q_j$ есть общая длительность всех работ из X в предположении, что они выполняются на одном процессоре, а функция ранга полиматроида $\varphi : 2^{\mathcal{J}} \to \mathbb{R}$ задается как

$$\varphi(X) = \max \left\{ \sum_{j \in X} y(s,j) : \ y \text{ есть допустимый поток в } G \right\}.$$

Тогда все возможные максимальные потоки можно описать через базовый многогранник $B(\varphi) = \{q \in Q_{(+)}(\varphi) : \ q(X) \leq \varphi(X), \ X \in 2^{\mathcal{J}}\}$ многогранника $Q_{(+)}(\varphi) = \{q \in \mathbb{R}^n_+ : \ q(\mathcal{J}) = \varphi(\mathcal{J})\}$. Для $X \subseteq \mathcal{J}$ и $h = 1, \ldots, \gamma$, обозначим через

$$\eta(X, h) := \sum_{j \in X: \ I_h \in \Gamma(j)} size_j$$

количество процессоров, которые могут быть использованы работами из X в интервале I_h. Тогда значение $\varphi(x)$ задается формулой в явном виде

$$\varphi(x) = \sum_{h=1}^{\gamma} \min\{m, \eta(X, h)\} \cdot \Delta_h,$$

и соответствует общей продолжительности всех временных интервалов, доступных для выполнения работ из множества X.

В терминах субмодулярной оптимизации, рассматриваемую задачу можно переформулировать так

$$\sum_{j=1}^{n} \left(\frac{q_j}{size_j}\right)^{1-\alpha} \left(\sum_{l=1}^{size_j} W_{jl}^{\alpha}\right) \to \min,$$

$$q \in B(\varphi). \tag{39}$$

Для решения задачи (39) будем использовать декомпозиционный алгоритм из [220]. Схема представлена в Алгоритме 2. Отметим, что подзадачи, которые должны быть решены на шагах 5 и 6, имеют общую структуру. Следовательно, исходная задача решается рекурсивно. Количество подзадач, возникающих в декомпозиционном алгоритме, равно $O(n)$, и поэтому шаги 1, 2 и 3 выполняются $O(n)$ раз. Следовательно, общее время работы декомпозиционного алгоритма составляет $O(nT_{123}(n))$, где $T_{123}(n)$ — временная сложность шагов 1, 2 и 3.

Шаг 1 может быть реализован за время $O(n)$, используя необходимое условие

$$\frac{\partial \left(\sum_{i=1}^{n} \left(\frac{q_i}{size_i}\right)^{1-\alpha} \left(\sum_{l=1}^{size_i} W_{il}^{\alpha}\right)\right)}{\partial q_j} = \lambda, \ j \in \mathcal{J}. \tag{40}$$

Алгоритм 2 Декомпозиционный алгоритм

1: Найти оптимальное решение $b \in \mathbb{R}^n$ следующей упрощенной задачи с одним ограничением:

$$\sum_{j=1}^{n} \left(\frac{q_j}{size_j}\right)^{1-\alpha} \left(\sum_{l=1}^{size_j} W_{jl}^{\alpha}\right) \to \min,$$

$$q(\mathcal{J}) = \varphi(\mathcal{J}),\ q_j \geq 0,\ j = 1, \ldots, n. \tag{41}$$

2: Найти максимальный вектор $c \in \mathbb{R}^n$, удовлетворяющий условию:

$$c(X) \leq \varphi(X),\ X \subseteq \mathcal{J},\ 0 \leq c_j \leq b_j,\ j = 1, \ldots, n.$$

3: Найти непустое множество $Y^* \subseteq \mathcal{J}$ такое, что

$$c(Y^*) = \varphi(Y^*),\ c_j = b_j,\ j \in \mathcal{J} \setminus Y^*.$$

4: Если $Y^* = \mathcal{J}$, то вернуть вектор c. Иначе перейти к шагу 5.

5: Найти оптимальное решение $q_1 \in \mathbb{R}^{|Y^*|}$ задачи:

$$\sum_{j \in Y^*} \left(\frac{q_j}{size_j}\right)^{1-\alpha} \left(\sum_{l=1}^{size_j} W_{jl}^{\alpha}\right) \to \min,$$

$$q(Y^*) = \varphi(Y^*),\ q(X) \leq \varphi(X),\ X \subset Y^*.$$

6: Найти оптимальное решение $q_2 \in \mathbb{R}^{n-|Y^*|}$ задачи:

$$\sum_{j \in \mathcal{J} \setminus Y^*} \left(\frac{q_j}{size_j}\right)^{1-\alpha} \left(\sum_{l=1}^{size_j} W_{jl}^{\alpha}\right) \to \min,$$

$$q(\mathcal{J} \setminus Y^*) = \varphi(\mathcal{J}) - \varphi(Y^*),$$
$$q(X) \leq \varphi(X \cup Y^*) - \varphi(Y^*),\ X \subset \mathcal{J} \setminus Y^*.$$

7: Вернуть прямую сумму $q^* = q_1 \oplus q_2$.

Итак, объединяя (40) и (41), получаем

$$q_j = \frac{\varphi(\mathcal{J}) \left(size_j^{\alpha-1} \sum_{l=1}^{size_j} (W_{jl})^{\alpha}\right)^{1/\alpha}}{\sum_{i \in \mathcal{J}} \left(size_i^{\alpha-1} \sum_{l=1}^{size_i} (W_{il})^{\alpha}\right)^{1/\alpha}},\ j \in \mathcal{J}. \tag{42}$$

Задача шага 2 сводится к задаче о максимальном потоке в сети G_b,

полученной из сети $G = (V, A)$ путем замены пропускных способностей $\mu(s, j) = +\infty$ на $b(j)$ для всех $j \in \mathcal{J}$. Для максимального потока x^* в G_b оптимальное решение задачи шага 2 задается выражением $c_j = x^*(s, j),\ j \in \mathcal{J}$. На шаге 3 множество Y^* строится из минимального s-t разреза (S^*, T^*) в G_b как $Y^* = S^*$ ввиду $c(S^*) = \varphi(S^*),\ c_j = b_j,\ j \in T^*$. Результаты для шагов 2 и 3 непосредственно следуют из Леммы 3 [220]. Максимальный поток в G_b может быть найден за время $O(n^3)$ с помощью алгоритма Карзанова, а минимальный s-t разрез в G_b вычисляется за $O(n^2)$ операций [141]. Таким образом, временная сложность декомпозиционного алгоритма составляет $O(n^4)$.

1.6.2 Общий момент поступления и директивный срок

Предположим, что все работы являются жесткими и имеют общий момент поступления r и/или общий директивный срок d. Построим сильно полиномиальные приближенные алгоритмы с константной оценкой точности для случаев, когда миграция работ между процессорами не допускается. Алгоритмы состоят из двух этапов. На первом этапе строится нижняя оценка на минимальное потребление энергии и вычислются промежуточные длительности работ. Затем, на втором этапе, вычисляются итоговые скорости работ и строится расписание.

Рассмотрим случай задачи без прерываний, когда все работы поступают в момент времени $r = 0$ и имеют общий директивный срок d.

Первый этап. Нижнюю оценку на целевую функцию можно найти за время $O(n^4 + n \max\limits_{j \in \mathcal{J}} size_j)$, используя метод, представленный в разделе 1.3.1. Здесь же предлагается более эффективный подход для рассматриваемого класса задач.

Построим следующую задачу выпуклого программирования, где e_j соответствует длительности работы $j \in \mathcal{J}$:

$$\sum_{j=1}^{n} e_j \sum_{l=1}^{size_j} \left(\frac{W_{jl}}{e_j} \right)^{\alpha} \to \min, \tag{43}$$

$$\sum_{j=1}^{n} size_j e_j = md. \tag{44}$$

Ограничение (44) задает условие на использование процессоров в интервале $[0, d)$. Потребление энергии задается в виде (43). Эта задача может быть решена с помощью метода Лагранжа. Определим функцию Лагранжа $L(e_j, \lambda)$ стандартным образом

$$\sum_{j=1}^{n} e_j \sum_{l=1}^{size_j} \left(\frac{W_{jl}}{e_j}\right)^{\alpha} + \lambda \left(\sum_{j=1}^{n} size_j e_j - md\right). \tag{45}$$

Необходимые и достаточные условия для оптимальности решения позволяют найти длительности работ

$$e_j = \frac{mdB_j}{\sum_{j'=1}^{n} size_{j'} B_{j'}}, j \in \mathcal{J}, \tag{46}$$

где $B_j = \left(\sum_{l=1}^{size_j} \frac{W_{jl}^{\alpha}(\alpha-1)}{size_j}\right)^{\frac{1}{\alpha}}$.

Отметим, что e_j может оказаться больше d для некоторых работ. Для корректировки решения в таком случае будем использовать следующую процедуру.

Пусть m' обозначает текущее количество свободных процессоров, а $\mathcal{J}'$ – текущее подмножество рассматриваемых работ. Изначально $\mathcal{J}' := \mathcal{J}$ и $m' := m$. Далее перебираем работы одну за другой в порядке невозрастания значений B_j.

Если текущая работа i имеет значение $B_i \geq \frac{\sum_{j \in \mathcal{J}'} B_j size_j}{m'}$, то полагаем ее длительность $p_i := d$ и обновляем $\mathcal{J}' := \mathcal{J}' \setminus \{i\}$ и $m' := m' - size_i$. После этого переходим к следующей работе. В противном случае все работы $l \in \mathcal{J}'$ удовлетворяют неравенству $B_l < \frac{\sum_{j \in \mathcal{J}'} B_j size_j}{m'}$, и их длительности полагаются равными $p_l := \frac{B_l m' d}{\sum_{j \in \mathcal{J}'} B_j size_j}$. Временная сложность процедуры составляет $O(n \log n + n \max\limits_{j \in \mathcal{J}} size_j)$.

Указанные ранее свойства гарантируют, что $\sum_{j \in \mathcal{J}} p_j size_j = md$, $p_j \leq d$, и дают нижнюю оценку на целевую функцию, равную $\sum_{j \in \mathcal{J}} p_j \sum_{l=1}^{size_j} \left(\frac{W_{jl}}{p_j}\right)^{\alpha}$.

На втором этапе используется списочный алгоритм из [191] для построения допустимого расписания без прерываний.

Второй этап. Для построения расписания используем следующий алгоритм: Всякий раз, когда подмножество процессоров становится свободным, списочный алгоритм ставит в расписание работу, которая требует не больше процессоров, чем доступно, пока все работы из $\mathcal{J}$ не будут назначены. Трудоемкость алгоритма составляет $O(n^2)$.

Утверждается, что длина построенного расписания не превышает $\left(2 - \frac{1}{m}\right) d$ (доказательство аналогично доказательству Леммы 5 в разделе 1.3.4).

Увеличивая скорость каждой операции работы в $\left(2 - \frac{1}{m}\right)$ раз, получаем расписание длиной не более d. Общее потребление энергии увеличивается в $\left(2 - \frac{1}{m}\right)^{\alpha-1}$ раз по сравнению с нижней оценкой. В результате имеет место

Теорема 10 *Для задач теории расписаний* $P|size_j, r_j = r, d_j = d|E$ *и* $P|size_j, pmtn*, r_j = r, d_j = d|E$ *можно построить* $\left(2 - \frac{1}{m}\right)^{\alpha-1}$*-приближенное расписание за время* $O(n^2 + n \max\limits_{j \in \mathcal{J}} size_j)$.

1.6.3 Общий момент поступления или директивный срок

Здесь рассматривается задача с прерываниями без миграции, где все работы поступают в момент времени $r = 0$, но имеют индивидуальные директивные сроки. Предварительные длительности e_j работ и нижняя оценка на целевую функцию вычисляются за время $O(n^4 + n \max\limits_{j \in \mathcal{J}} size_j)$ с помощью решения задачи поиска максимального потока минимальной стоимости, представленной в разделе 1.3.1.

Затем используется списочный алгоритм с упорядочением работ по неубыванию директивных сроков для построения приближенного решения. Работы ставятся в расписание в порядке неубывания директивных сроков по следующему правилу. Если $size_i > \left\lceil \frac{m}{2} \right\rceil$, то работа i назначается в конце текущего расписания. В противном случае работа i ставится в расписание в самый ранний момент времени, когда $size_i$ процессоров свободны, и выполняется в течение e_i единиц времени, игнорируя интервалы с работами, имеющими $size_j > \left\lceil \frac{m}{2} \right\rceil$. Временная сложность алгоритма составляет $O(n^2)$.

Утверждается, что время завершения C_j каждой работы j в построенном расписании не превышает $(3-\varphi_m)\,d_j$ (см. Лемму 4 ниже).

Таким образом, увеличение скоростей в $(3-\varphi_m)$ раз приводит к допустимому расписанию. Общее потребление энергии увеличивается в не более чем $(3-\varphi_m)^{\alpha-1}$ раз.

Очевидно, что если поменять ролями моменты поступления и директивные сроки, то представленный алгоритм может также использоваться для случая работ с индивидуальными моментами поступления, но общим директивным сроком. В результате имеем

Теорема 11 *Для задач теории расписаний* $P|size_j, pmtn*, r_j = r, d_j|E$ *и* $P|size_j, pmtn*, r_j, d_j = d|E$ *можно найти* $(3-\varphi_m)^{\alpha-1}$*-приближенное расписание за время* $O(n^4 + n \max\limits_{j\in\mathcal{J}} size_j)$.

Напомним, что

$$\varphi_m = \begin{cases} \frac{4}{m+1} & \text{если } m \text{ нечетно,} \\ \frac{6}{m+2} & \text{если } m \text{ четно,} \end{cases}$$

и докажем следующую лемму.

Лемма 4 *Дано* m *процессоров и множество работ* $\mathcal{J}$ *с директивными сроками* d_i*, длительностями* $e_i \le d_i$ *и требуемым числом процессоров* $size_i$*, где* $\sum_{j\in\mathcal{J}_i} e_j size_j \le m d_i$ *для каждой работы* $i \in \mathcal{J}$*,* $\mathcal{J}_i = \{j \in \mathcal{J} : \ d_j \le d_i\}$*. Утверждается, что для каждой работы* $i \in \mathcal{J}$ *время завершения* C_i *не превышает* $(3-\varphi_m)\,d_i$ *в расписании* S*, построенном списочным алгоритмом с упорядочением работ по неубыванию директивных сроков.*

Доказательство. Рассмотрим произвольный директивный d_i, где i есть работа, имеющая максимальное время завершения C_i в расписании S среди всех работ с директивным сроком, равным d_i.

Отметим, что $C_j \le C_i$ для всех работ $j \in \mathcal{J}_i$. Пусть S_i обозначает ту часть расписания S, которая содержит только работы из подмножества $\mathcal{J}_i$ и занимает интервал $[0, C_i)$. Покажем, что $C_i \le (3-\varphi_m)\,d_i$.

Если в любой момент времени в расписании S_i используется не менее $\left\lceil \frac{m+1}{2} \right\rceil$ процессоров, то имеем

$$d_i \geq \frac{1}{m} \sum_{j \in \mathcal{J}_i} e_j size_j \geq \left\lceil \frac{m+1}{2} \right\rceil \frac{C_i}{m} \geq \frac{C_i}{2 - \frac{1}{m}}.$$

В противном случае предположим, что l – это последняя работа в расписании S_i, которая требует $size_l \leq \left\lceil \frac{m}{2} \right\rceil$ процессоров. Легко увидеть, что в интервалах $[0, C_l - e_l)$ и $[C_l, C_i)$ используется не менее $\left\lceil \frac{m+1}{2} \right\rceil$ процессоров в каждый момент времени, и не менее $size_l$ процессоров используется в интервале $[C_l - e_l, C_l)$. Таким образом, общая загрузка всех процессоров в расписании S_i составляет не менее

$$\left\lceil \frac{m+1}{2} \right\rceil (C_i - e_l) + size_l e_l \leq \sum_{j \in \mathcal{J}_i} e_j size_j \leq d_i m.$$

Если $e_l \geq \frac{C_i}{3 - \varphi_m}$, то $C_i \leq (3 - \varphi_m)\, d_l \leq (3 - \varphi_m)\, d_i$.

В противном случае, ввиду того, что $size_l \geq 1$, имеем

$$\begin{aligned} m d_i &\geq \left\lceil \frac{m+1}{2} \right\rceil C_i - e_l \left(\left\lceil \frac{m+1}{2} \right\rceil - size_l \right) \\ &\geq \left\lceil \frac{m+1}{2} \right\rceil C_i - \left\lceil \frac{m-1}{2} \right\rceil e_l \\ &\geq C_i \left(\left\lceil \frac{m+1}{2} \right\rceil - \left\lceil \frac{m-1}{2} \right\rceil \frac{1}{3 - \varphi_m} \right) = \frac{m C_i}{3 - \varphi_m}. \end{aligned}$$

□

Глава 2

Задачи составления расписаний с ограничением на потребление энергии

Вторая глава посвящена исследованию задач составления расписаний с ограничением на потребление энергии при классических временных критериях – длина расписания и сумма моментов завершения.

1. Исследуются вопросы труднорешаемости задач теории расписаний с многопроцессорнымии и распараллеливаемыми работами при классических критериях и бюджетных ограничениях на расход энергии, выделяются полиномиально разрешимые и NP-трудные частные случаи.

2. Для задач с ограничением на расход энергии предлагаются подходы к построению нижних оценок на целевую функцию, которые основаны:
   – на представлении исходной задачи как задачи выпуклого программирования и решении ее с использованием условий Каруша-Куна-Таккера;
   – на представлении исходной задачи как задачи линейного программирования с экспоненциальным числом переменных и полиномиальным числом ограничений и решении ее методом эллипсоидов с использованием полиномиального оракула отсечения.

   При построении допустимых решений применяются либо жадные алгоритмы, либо методы декомпозиции.

Указанные результаты являются первыми приближенными алгоритмами с гарантированными оценками точности для многопроцессорных и распараллеливаемых работ с учетом расхода энергии и обобщают известные результаты для однопроцессорных работ.

Результаты второй главы опубликованы в работах [6, 7, 151, 153–156].

2.1 Длина расписания

Энергопотребление вычислительных устройств является важной проблемой, которая активно исследуется в настоящее время [21,24,46,75,115, 220]. Одним из способов снижения энергопотребления является динамическое масштабирование скорости, при котором процессор может динамически изменять свою скорость. Выполнение работы (операции) на более низкой скорости более энергоэффективно, однако занимает больше времени и может повлиять на производительность работы системы.

В настоящем разделе рассматривается задача составления расписаний, где скорости работ влияют на потребление энергии, работы допускают параллельное выполнение и минимизируется длина расписания при ограничении на суммарное потребление энергии.

Имеется множество работ $\mathcal{J} = \{1, \ldots, n\}$, которые должны быть выполнены на m процессорах, имеющих возможность работать с различной скоростью. Рассматриваются жесткие работы и одномодальные многопроцессорные работы. Напомним, что в первом случае для каждой работы $j \in \mathcal{J}$ задан размер $size_j$, показывающий, что работа может быть выполнена на любом подмножестве параллельных процессоров указанной мощности. Во втором случае выполнение одномодальной работы $j \in \mathcal{J}$ одновременно требует заранее определенного подмножества fix_j заданных процессоров. В обоих случаях работа $j \in \mathcal{J}$ характеризуется общим объемом V_j (время выполнения на одном процессоре с единичной скоростью). Будем считать, что все работы поступают в момент времени 0, если не указано иначе. Прерывания, миграция между процессорами и частичный порядок могут быть разрешены или запрещены в рассматриваемых вариантах задачи.

Жесткие работы с прерываниями и *миграцией* возникают, когда приложения выполняются в системе с общей памятью, поддерживающей работу с UNIX, Windows или аналогичными операционными системами. В этом случае создание, прерывание, возобновление и завершение потока (или процесса) допускаются и не требуют больших затрат времени. Задачи *без миграции* возникают в приложениях для компьютерных систем с распределенной памятью (например, на компьютерах есть отдельные образы памяти, управляемые независимыми операционными системами).

Здесь затраты на миграцию существенно выше, и потоки не мигрируют. Таким образом, во время работы приложения используемые наборы процессоров не меняются. Одномодальные многопроцессорные работы возникают когда приложению требуется определенный набор ресурсов: сетевые подключения, специализированные процессоры (GPU, FPGA), библиотеки, лицензии и т. д.

Рассматривается стандартная однородная модель в контексте настройки скоростей процессоров. Если процессор работает со скоростью s, то потребление энергии в единицу времени (*мгновенная мощность*) равна s^α, где $\alpha > 1$ – заданная константа. Каждый из m процессоров может работать с переменной скоростью. Предполагается, что доступно непрерывное множество скоростей процессоров из заданного диапазона. В общем случае будем предполагать, что общий объем работы V_j $j \in \mathcal{J}$, может быть произвольным образом распределен между используемыми процессорами. Ввиду выпуклости функции расчета мгновенной мощности существует оптимальное решение, в котором общий объем работы равномерно распределяется между процессорами, т. е., если работа j использует m_j процессоров, то объемы одинаковы для всех m_j процессоров (обозначаются через $W_j := \frac{V_j}{m_j}$), и все эти процессоры работают с одинаковой скоростью при выполнении работы. Большинство результатов можно обобщить на случай неравномерного распределения объема работы между процессорами.

Требуется найти найти допустимое расписание наименьшей длины (момент завершения последней работы, makespan, $C_{\max}$), при котором суммарное потребление энергии не превышает заданный бюджет E. Такое условие возникает в случае, когда мощность батареи ограничена, то есть задача актуальна в приложениях, связанных с вычислительными устройствами, срок службы которых зависит от ограниченной эффективности батареи (например, многоядерные ноутбуки). Более того, в реальной практике возникают двухкритериальные задачи минимизации потребления энергии и некоторой характеристики расписания. Возможный подход к решению – ограничить одну из целевых функций (например, потребление энергии) и оптимизировать другую.

Рассмотрим пример с тремя жесткими работами, пятью процессо-

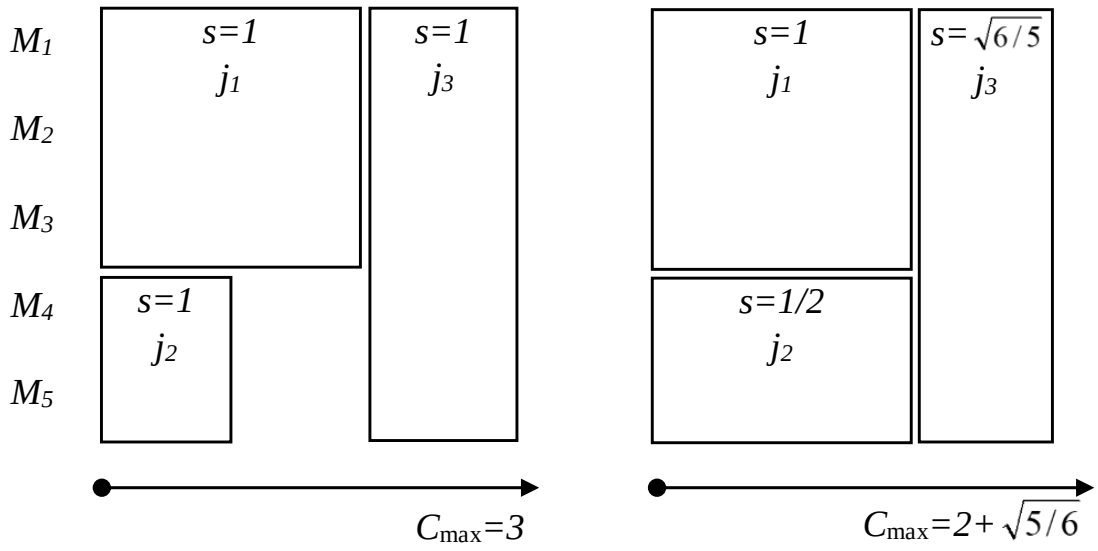


Рисунок 5 — Два допустимых расписания для задачи с $\alpha = 2$, $n = 3$, $m = 5$ и $E = 13$.

рами, $\alpha = 2$, и ограничением на потребление энергии $E = 13$. Пусть объемы равны $V_{j_1} = 6$, $V_{j_2} = 2$ и $V_{j_3} = 5$; размеры равны $size_{j_1} = 3$, $size_{j_2} = 2$ и $size_{j_3} = 5$. Два допустимых расписания для этого примера представлены на рис. 5. В первом случае все работы выполняются с единичной скоростью на всех используемых процессорах и $C_{\max} = 3$. Во втором случае процессоры работают с одинаковой скоростью, выполняя одну и ту же работу, но разные работы выполняются с разной скоростью, и $C_{\max} = 2 + \sqrt{5/6}$.

Вариант задачи без прерываний и с прерываниями при возможности варьирования скоростей и учете расхода энергии обозначается через $P|m_j, energy|C_{\max}$ и $P|m_j, pmtn, energy|C_{\max}$, соответственно. Здесь $m_j \in \{size_j, fix_j\}$ указывает тип работ (жесткие или одномодальные). В случае наличия частичного порядка между работами в обозначение (второе поле) добавляется «prec».

2.1.1 Предыдущие исследования

Задачи с учетом расхода энергии для однопроцессорных работ. Варианты задачи составления расписаний с критерием минимизации длины расписания на одном процессоре исследованы в литературе.

Для задачи на одном процессоре существует оптимальное расписание без прерываний, в котором все работы выполняются с одинаковой скоростью. Это расписание может быть построено за линейное время (см., например, [201]). Bunde [62] предложил точный конструктивный алгоритм для однопроцессорного случая с произвольными моментами поступления работ, трудоемкость которого равна $O(n)$.

Многопроцессорный вариант задачи минимизации длины расписания без миграции является NP-трудным даже в случае двух процессоров [62]. Задача без прерываний является NP-трудной в сильном смысле. Pruhs и др. [201] предложили PTAS для задачи без миграции с независимыми работами, имеющими идентичные моменты поступления. Здесь минимизация длины расписания сводится к минимизации l_α нормы загрузки процессоров, и в полученном расписании процессоры завершают работу одновременно. Также предложен алгоритм с оценкой точности $O(\log^{1+2/\alpha} m)$ для работ без миграции при наличии отношений предшествования. В формируемом расписании сумма мгновенных мощностей, с которыми работают процессоры, постоянна во времени. Bampis и др. [31] предложили алгоритм с оценкой точности $\left(2 - \frac{1}{m}\right)$, который позволяет вычислить длительности работ и нижнюю оценку на энергопотребление. Это позволяет свести задачу с настройкой скоростей работ к классической задаче минимизации длины расписания с фиксированными скоростями работ. Быстрый алгоритм построения приближенного решения был предложен в [62] для многопроцессорного случая без миграции с независимыми работами, имеющими идентичные объёмы и произвольные моменты поступления. Алгоритм назначает работы процессорам с использованием стратегии Round Robin и далее находит расписание для каждого процессора отдельно.

Shabtay и др. [216] исследовали близкую по постановке задачу составления расписаний на одном процессоре и на идентичных параллельных процессорах, где длительности работ p_j зависят от потребления ограниченного невозобновимого общего ресурса: $p_j(R_j) = \left(\frac{W_j}{R_j}\right)^\kappa$. Здесь W_j — объём работы j, R_j – количество ресурса, выделенного для выполнения работы j, а $0 < \kappa \leq 1$ – заданная положительная константа. Были предложены точные полиномиальные алгоритмы для следующих случаев:

- однопроцессорная задача минимизации длины расписания с отношениями предшествования;
- многопроцессорная задача минимизации длины расписания с независимыми работами и возможностью прерываний;
- многопроцессорная задача минимизации суммы моментов завершения при запрете прерываний выполнения работ.

Отметим, что алгоритмы из [216] могут быть адаптированы для решения этих трёх случаев в контексте составления расписаний с учетом расхода энергии на одном процессоре.

Алгоритм с неконстантной оценкой точности для жёстких работ с общим моментом поступления и общим директивным сроком был предложен в [173]. Здесь объёмы работ сначала рассматриваются как длительности, и строится допустимое расписание с помощью алгоритма списочного типа. Затем полученное расписание модифицируется путём назначения скоростей работам с учетом общего допустимого объема энергии.

Минимизация длины расписания для распараллеливаемых работ. Теперь рассмотрим известные результаты для классической задачи составления расписаний с минимизацией длины расписания для жёстких работ с заданными длительностями и без ограничений на расход энергии. Если количество процессоров является частью входа, то задача с прерываниями и жёсткими работами является NP-трудной [90]. Более того, для этой задачи не может быть построен алгоритм с оценкой точности лучше чем $\frac{3}{2}$ относительно оптимума [47]. В [47] также предложен алгоритм с оценкой точности $\left(2 - \frac{1}{m}\right)$. Если количество процессоров ограничено константой, то задача с прерываниями и жесткими работами полиномиально разрешима [137]. Задача без прерываний при произвольном числе процессоров является NP-трудной в сильном смысле. Задача с двумя процессорами является NP-трудной в обычном смысле, но становится NP-трудной в сильном смысле при наличии частичного порядка между работами [60, 91]. Задача полиномиально разрешима в случае, когда количество процессоров ограничено константой, а работы имеют одинаковые длительности [40, 90]. Приближенные алгоритмы оценкой точности 2 были предложены для независимых работ с иден-

тичными и произвольными моментами поступления [90]. Различные алгоритмы с константной оценкой точности были предложены для работ с заданным частичным порядком. Эти алгоритмы используют стратегии построения расписаний списочного и уровневого типов [90].

Задача составления расписаний однопроцессорных работ без прерываний, требующих не менее трёх процессоров, является NP-трудной в сильном смысле и для нее не существует полиномиального приближенного алгоритма с константной оценкой точности относительно оптимума, если $P \neq NP$ [136]. Однако оптимальное расписание может быть найдено за полиномиальное время для однопроцессорных работ на двух процессорах [163]. Задача без прерываний является NP-трудной даже в случае составления расписаний работ на двух или трёх процессорах [132, 164]. Приближенные алгоритмы и границы неаппроксимируемости построены для частных случаев в работах, представленных в книге [90].

Метод эллипсоидов и условия Каруша-Куна-Таккера в теории расписаний. Основная идея метода эллипсоидов заключается в следующем. Алгоритм начинает работу с начального эллипсоида, содержащего все множество допустимых решений. На каждом шаге рассматривается центр эллипсоида. Если он не является допустимым решением, то отыскивается ограничение, которое нарушено для выбранной точки. В этом случае проводится гиперплоскость через центр так, чтобы все допустимые решения лежали по одну сторону от этой гиперплоскости. Если центр является допустимым решением, то отсекающая гиперплоскостью строится на основе целевой функции. В любом случае гиперплоскость делит эллипсоид на две части, одна из которых содержит оптимальное решение. Затем строится новый эллипсоид меньшего объёма. Объёмы эллипсоидов сходятся к 0 экспоненциально, что гарантирует, что центры, являющиеся допустимыми решениями, будут стремиться к оптимальному решению экспоненциально быстро, а время работы алгоритма будет полиномиальным [123, 193].

Условия Каруша-Куна-Таккера являются необходимыми и достаточными для того, чтобы решения задач выпуклой оптимизации были оптимальными [57, 166].

Рассмотрим следующую задачу выпуклого программирования:

$$f(x) \to \min$$

$$g_i(x) \leq 0,\ i = 1, \ldots, m,$$

$$x \in R^n,$$

где все функции g_i дифференцируемы, и существует внутренняя точка x' такая, что $g_i(x') < 0$ для всех $i = 1, \ldots, m$.

Пусть λ_i будет двойственной переменной, связанной с ограничением $g_i(x) \leq 0$. Условия Каруша-Куна-Таккера имеют вид:

$$g_i(x) \leq 0,\ i = 1, \ldots, m,$$

$$\lambda_i \geq 0,\ i = 1, \ldots, m,$$

$$\lambda_i g_i(x) = 0,\ i = 1, \ldots, m,$$

$$\nabla L(x, \lambda) = 0,$$

где $L(x, \lambda) = f(x) + \sum_{i=1}^{n} \lambda_i g_i(x)$ – функция Лагранжа. Условия Каруша-Куна-Таккера необходимы и достаточны для того, чтобы решения $x \in R^n$ и $\lambda \in R^m$ были прямо и двойственно оптимальными [57].

Метод эллипсоидов и условия Каруша-Куна-Таккера часто применяются в алгоритмических конструкциях для задач составления расписаний с многопроцессорными работами и выпуклыми функциями потребления ресурсов.

Выделим три основных направления. К первому направлению относятся выпуклые задачи математического программирования с полиномиальным числом ограничений (линейные ограничения для учета временных характеристик расписаний и выпуклые ограничения для задания потребления ресурсов), где используется метод эллипсоидов [193] для решения задачи с произвольной заданной точностью за полиномиальное время. Этот подход был успешно интегрирован в алгоритмы для задач составления энергетически эффективных расписаний или задач с оптимизацией максимальной температуры при заданных временных окнах выполнения работ и для оптимизации качества обслуживания при ограничениях на расход энергии или максимальную температуру [24, 31, 35,

36, 168, 202].

Второе направление включает задачи линейного программирования с конфигурациями, где рассматриваются все возможные комбинации работ с учётом интервалов их существования и доступных ресурсов машин (процессоров). Требуется выбрать подмножество конфигураций, соответствующее оптимальному решению. Получающиеся задачи линейного программирования содержат экспоненциальное количество переменных (из-за экспоненциального количества конфигураций) и полиномиальное количество ограничений. Они могут быть решены методом эллипсоидов с полиномиальным отделяющим оракулом за полиномиальное время [123]. Используя этот подход, были предложены «почти точные» алгоритмы для жёстких работ и расписаний на двух процессорах с фиксированными длительностями, многопроцессорных работ с миграцией и критерием минимизации энергии, а также задач цехового типа с прерываниями [32, 134, 136].

В третьем направлении исходная задача формулируется как модель целочисленного выпуклого программирования, далее строится ее линейная релаксация с экспоненциальным числом переменных, которая решается методом эллипсоидов, а затем вычисляется целочисленное приближенное решение с помощью рандомизированного округления. Этот подход позволяет получить приближенные алгоритмы для задачи составления энергетически эффективных расписаний, балансировки нагрузки на неоднородных параллельных машинах и составления расписаний на неоднородных параллельных машинах с нелинейными целевыми функциями от моментов завершения [182].

Классические условия Каруша-Куна-Таккера [57,166] позволяют определить структурные свойства оптимальных расписаний и разработать более быстрые и менее сложные комбинаторные алгоритмы по сравнению с методом эллипсоидов. Примеры можно найти в [24, 31, 35, 37, 202, 216] для задач с выпуклыми функциями потребления ресурсов и различными критериями. Условия Каруша-Куна-Таккера могут быть адаптированы и обобщены для задач многокритериальной оптимизации [224], задач с интервальными данными, нечёткими целевыми функциями [26], негладких, полуопределённых, полубесконечных, двухуровневых задач [81]

и других подходов.

Основные результаты раздела. В диссертации строятся приближенные алгоритмы, основанные на двух этапах. На первом этапе вычисляется нижняя оценка на целевую функцию и находятся длительности работ с использованием вспомогательной задачи выпуклого программирования, метода эллипсоидов или условий Каруша-Куна-Таккера. Затем на втором этапе исходная задача преобразуется в классическую задачу без учета расхода энергии и используются алгоритмы списочного типа для построения расписаний с гарантированной константной оценкой точности. Исследуются следующие случаи минимизации длины расписания:

- жёсткие работы без прерываний;
- жёсткие работы с отношениями предшествования и без прерываний;
- жёсткие работы с прерываниями и произвольными моментами поступления;
- однопроцессорные работы на двух процессорах без прерываний.

В работе применяется метод эллипсоидов [123, 193] для решения выпуклых или линейных задач математического программирования с рациональными входными данными. Этот метод возвращает решение с рациональными компонентами. Однако оптимальное решение рассматриваемых задач может содержать иррациональные компоненты. Рассмотрим простой пример, где $n = m$; $size_j = 1$ (или $|fix_j| = 1$), $W_j = 1$ для всех $j \in \mathcal{J}$; $\alpha = 3$ и $E = \frac{n}{2}$. Легко видеть, что в оптимальном решении все работы имеют длительности, равные $\sqrt{2}$, и длина расписания равна $\sqrt{2}$. Таким образом, метод эллипсоидов решает так называемые *слабые* задачи оптимизации и генерирует «почти точное» решение, для которого целевая функция не превышает $C^*_{\max} + \varepsilon$, а ограничения могут быть нарушены, но не более чем на аддитивную ошибку ε для любого заданного $\varepsilon > 0$ ($C^*_{\max}$ – длина оптимального расписания). Временная сложность полиномиальна от $\log\frac{1}{\varepsilon}$ и размера входа задачи. Увеличение длительностей работ в $(1+\varepsilon)$ раз позволяет соблюдать ограничения без аддитивной ошибки ε, но приводит к погрешности $(1+\varepsilon)$ для длины

расписания (см., например, [32]). Для простоты описания будем игнорировать аддитивную ошибку в ограничениях.

Иррациональные значения возникают в алгоритмах из-за функции расчета мгновенной мощности s^α для скорости s. Будем аппроксимировать эти значения с любой заданной точностью $\varepsilon > 0$ с помощью одного из известных численных методов за время, полиномиальное от $\log\frac{1}{\varepsilon}$ и размера входа (см., например, метод Ньютона или метод дихотомии в [167]).

2.1.2 От настраиваемых скоростей к фиксированным скоростям

Минимизация длины расписания является классическим критерием в задачах теории расписаний с фиксированными длительностями работ. Разработано много алгоритмов для разных вариантов задач с таким критерием. Для задачи с возможностью варьирования скоростей процессоров при учете расхода энергии предлагается метод решения, основанный на вычислении нижней оценки на длину расписания и предварительных оценок длительностей работ. Это позволяет свести задачу с возможностью варьирования скоростей процессоров при учете расхода энергии к классической задаче минимизации длины расписания с фиксированными скоростями работ.

Жеские работы без прерываний. С помощью полиномиальной сводимости задач Разбиение и 3-Разбиение можно показать, что задачи составления расписаний $P2|size_j, energy|C_{\max}$ и $P|size_j, pmtn, energy|C_{\max}$ являются NP-трудными в обычном смысле, а задача $P|size_j, energy|C_{\max}$ является NP-трудной в сильном смысле (принцип доказательства основан на методе из раздела 1.2). Предлагается двухэтапный алгоритм с гарантированной константной оценкой точности для случая жестких работ без прерываний.

Первый этап. Ослабим условие, что процессоры должны выполнять работу одновременно. Точнее, заменяем работу $j \in \mathcal{J}$ на $size_j$ однопроцессорных работ с объемом W_j при условии, что все эти однопроцессорные копии будут иметь одинаковую длительность p_j. Тогда при

указанных предположениях можно сформулировать следующую задачу выпуклого программирования с переменными p_j и T, где p_j есть фактическая длительность каждой копии работы j:

$$T \to \min, \tag{47}$$

$$\max_{j \in \mathcal{J}} p_j \leq T, \tag{48}$$

$$\frac{1}{m} \sum_{j \in \mathcal{J}} size_j p_j \leq T, \tag{49}$$

$$\sum_{j \in \mathcal{J}} size_j W_j^{\alpha} p_j^{1-\alpha} \leq E, \tag{50}$$

$$p_j \geq 0, \ j \in \mathcal{J}. \tag{51}$$

Ограничения (48)-(49) гарантируют, что длительность любой работы не превышает T, и общая загрузка всех процессоров не превосходит Tm. Неравенство (50) задает условие, что общее потребление энергии не превышает бюджет E.

Задача (47)-(51) может быть решена с использованием условий ККТ (Каруша-Куна-Таккера) с любой заданной точностью $\varepsilon' > 0$ (см., например, [57, 166]). Трудоемкость алгоритма решения (см. описание в разделе 2.3.5) составляет $O\left(n \log\left(\frac{1}{\varepsilon'} nm W_{\max}\right)\right)$, где $W_{\max} = \max_{j \in \mathcal{J}} W_j$. Пусть $(p_1^*, \ldots, p_n^*, T^*)$ обозначает «почти» оптимальное решение с точностью ε' для (47)-(51). В результате получаем «почти» нижнюю границу T^* оптимальной длины расписания, C_{max}^*, то есть $T^* \leq C_{max}^* + \varepsilon'$.

Второй этап. Расписание S без прерываний строится с использованием алгоритма списочного типа [191]: Всякий раз, когда подмножество процессоров становится свободным, алгоритм назначает жесткую работу j длительностью p_j^*, которая не требует больше процессоров, чем доступно (если это возможно). Трудоемкость алгоритма составляет $O(n^2)$.

Лемма 5 *Для задачи $P|size_j, energy|C_{\max}$ алгоритм списочного типа без прерываний строит допустимое расписание S длины не более $\left(2 - \frac{1}{m}\right) T^*$.*

Доказательство. Пусть T' обозначает длину расписания S. Если в

любой момент времени расписания S используется не менее $\lceil \frac{m+1}{2} \rceil$ процессоров, то имеем

$$T^* \geq \frac{1}{m} \sum_{j \in \mathcal{J}} p_j^* size_j \geq \left\lceil \frac{m+1}{2} \right\rceil \frac{T'}{m} \geq \frac{T'}{2 - 1/m}.$$

В противном случае предположим, что I – последний интервал времени в расписании S, где используется $m_I < \lceil \frac{m+1}{2} \rceil$ процессоров.

По построению S, существует работа j, которая выполняется в течение всего интервала I. Пусть C_j есть время завершения j в S. Легко увидеть, что в каждый момент времени в течение интервала $[0, C_j - p_j^*)$ согласно расписанию S используется не менее $m - size_j + 1 \geq m - m_I + 1$ процессоров (иначе работа j должна была быть начата раньше). Более того, не менее m_I процессоров используется в интервале $[C_j - p_j, C_j)$, поэтому каждая работа, выполняемая в интервале $[C_j, T')$, требует не менее $m - m_I + 1$ процессоров. Таким образом, общая загрузка всех процессоров $\sum_{j \in \mathcal{J}} p_j^* size_j$ составляет не менее

$$m_I p_j^* + (T' - p_j^*)(m - m_I + 1) \leq T^* m. \tag{52}$$

Поскольку $1 \leq m_I < \lceil \frac{m+1}{2} \rceil$, $m - 2m_I + 1 > 0$ и $p_j^* \leq T^*$ для всех работ $j \in \mathcal{J}$, имеем из (52), что

$$\begin{aligned} T' &\leq \frac{mLB + p_j^*(m - 2m_I + 1)}{m - m_I + 1} \leq \frac{T^*(2m - 2m_I + 1)}{m - m_I + 1} \\ &= 2T^* - T^* \frac{1}{m - m_I + 1} \leq T^* \left(2 - \frac{1}{m}\right). \end{aligned}$$

Используя результаты из [120], получаем, что относительная оценка точности $\left(2 - \frac{1}{m}\right)$ для алгоритма списочного типа является достижимой, даже если все работы задействуют при выполнении только один процессор. Действительно, рассмотрим индивидуальную задачу с $(m^2 - m)$ работами малого объема, которые имеют размеры $size_j = 1$ и объемы $W_j = 1$, и однопроцессорной работой объема m, т. е. $n = m^2 - m + 1$. Граница на потребление энергии E равна m^2. В оптимальном расписании работа большого объема назначается на первый процессор, а m работ малого объема каждому из оставшихся $m - 1$ процессоров; скорость каждого

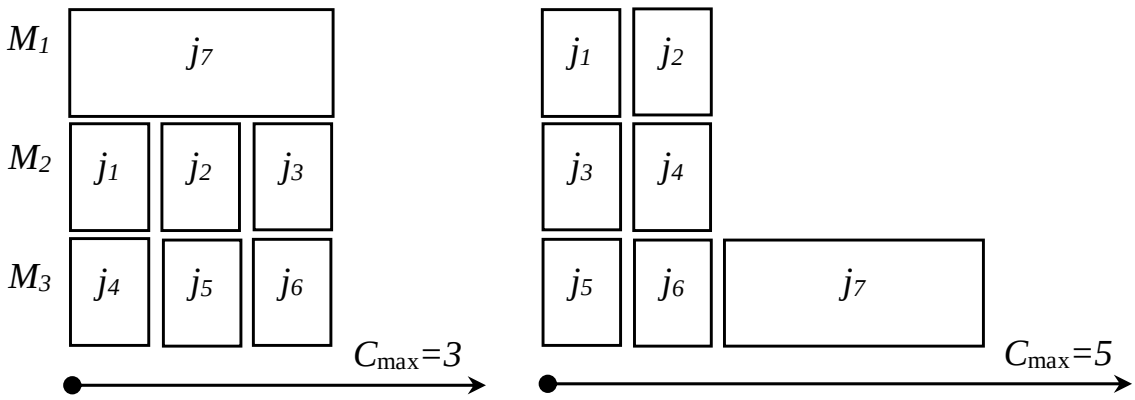


Рисунок 6 — Оптимальное расписание длины m и возможное расписание, построенное алгоритмом списочного типа, длина которого равна $2m-1$ для случая $m=3$.

процессора равна 1. Длина оптимального расписания составляет m.

Из модели для построения нижней оценки (47)-(51) имеем $T^* = m$, $p_1^* = \ldots = p_{n-1}^* = 1$ и $p_n^* = m$. Предположим, что работы упорядочены по неубыванию длительностей p_j^*. Тогда результирующее расписание, полученное с помощью алгоритма списочного типа, имеет длину $2m-1$, поскольку (m^2-m) работ длительностью $p_j^* = 1$ выполняются первыми (см. рис. 6).

Если выбрать точность $\varepsilon' = \frac{\varepsilon}{2}$ в алгоритме для построения нижней оценки при любом заданном $\varepsilon > 0$, то получим следующий результат.

Теорема 12 *Для задачи составления расписаний $P|size_j, energy|C_{\max}$ приближенное расписание длины $\left(2-\frac{1}{m}\right)C_{max}^* + \varepsilon$ можно построить за время, полиномиальное от $\log\frac{1}{\varepsilon}$ и размера входа индивидуальной задачи.*

Для жестких работ с произвольными моментами поступления можно использовать тот же алгоритм построения допустимого расписания с временем начала первой работы, равным времени поступления последней работы, поэтому имеет место

Следствие 3 *Для задачи $P|r_j, size_j, energy|C_{\max}$ приближенное расписание длины не более $\left(3-\frac{1}{m}\right)C_{max}^* + \varepsilon$ можно построить за время, полиномиальное от $\log\frac{1}{\varepsilon}$ и размера входа индивидуальной задачи.*

Жеские работы с частичным порядком. Рассмотрим случай задачи без прерываний с жесткими работами и отношениями предшествованиями. Если работа j предшествует работе j' (обозначение $j \prec j'$), то это будет означать, что j' не может начаться, пока j не завершена. Отношения предшествования будем представлять ориентированным ациклическим графом $G = (J, A)$, где дуга $(j, j') \in A$ тогда и только тогда, когда работа j предшествует работе j'. Кроме того, предполагается, что каждая работа j требует $size_j \leq qm$ процессоров, где $0 < q < 1$.

В [91] доказана NP-трудность в сильном смысле задачи составления расписаний $P2|size_j, chain|C_{\max}$ для случая, когда частичный порядок состоит из набора цепей. Используя метод из раздела раздела 1.2 и полиномиальную сводимость из [91], можно доказать, что задача составления расписаний $P2|size_j, chain, energy|C_{\max}$ также NP-трудна в сильном смысле. Построим приближенный алгоритм для задачи с произвольным числом процессоров $P|size_j, prec, energy|C_{\max}$.

Первый этап. Сформулируем следующую задачу выпуклого программирования для получения нижней оценки на длину расписания:

$$T \to \min, \tag{53}$$

$$\frac{1}{m} \sum_{j \in \mathcal{J}} p_j size_j \leq T, \tag{54}$$

$$C_j \leq T,\ j \in \mathcal{J}, \tag{55}$$

$$p_j \leq C_j,\ j \in \mathcal{J}, \tag{56}$$

$$C_j + p_{j'} \leq C_{j'},\ (j, j') \in A, \tag{57}$$

$$\sum_{j \in \mathcal{J}} size_j W_j^{\alpha} p_j^{1-\alpha} \leq E, \tag{58}$$

$$C_j \geq 0,\ p_j \geq 0,\ j \in \mathcal{J}. \tag{59}$$

Здесь переменная p_j (C_j) соответствует длительности (времени завершения) работы $j \in \mathcal{J}$. Неравенства (56)-(57) задают ограничение на время завершения работ с учетом длительностей и ограничений предшествования. Задача выпуклого программирования (53)-(59) позволяет

получить полиномиальный алгоритм для вычисления нижней оценки, поскольку она может быть решена с произвольной заданной точностью с помощью метода эллипсоидов (см., например, [193]). Для заданной точности $\varepsilon' > 0$ (аддитивная ошибка) трудоемкость метода эллипсоидов линейно зависит от $\log\frac{1}{\varepsilon'}$ [193].

Для применимости метода эллипсоидов необходимо проверить два дополнительных технических условия. Первое условие заключается в том, что значения всех переменных должны быть ограничены сверху некоторой величиной R. Второе условие заключается в том, что для задачи выпуклого программирования существует допустимая точка (решение), и каждая точка в окрестности радиуса r от нее также допустима. Тогда трудоемкость метода эллипсоидов будет полиномиальным от $\log\frac{R}{r}$. Первое условие и границу R можно получить, используя условие, что длина оптимального расписания ограничена ($p_j \leq \left(\frac{size_j W_j^\alpha}{E}\right)^{\frac{1}{\alpha-1}}$, $C_j \leq T$ для всех $j \in \mathcal{J}$, и $T \leq \sum_{i\in\mathcal{J}} \left(\frac{size_i W_i^\alpha}{E}\right)^{\frac{1}{\alpha-1}}$). Следовательно, величина R ограничена полиномом от входных данных задачи. Второе условие выполняется для точки $(p'_j,\ C'_j, T')$, определяемой следующим образом: $p'_j := W_j \left(\frac{\sum_i W_i size_i}{E}\right)^{\frac{1}{\alpha-1}} + 1,\ j \in \mathcal{J}$, $C'_1 := (p'_1 + 1) + 1$, $C'_j := C'_{j-1} + (p'_j + 1) + 1$ для $j = 2, \ldots, n$ (здесь предполагается, что работы упорядочены в соответствии с частичным порядком) и величина T' достаточно велика, такая что

$$T' - 1 \geq \max\left\{ C'_n + 1; \frac{1}{m}\sum_{j\in\mathcal{J}} \left((p'_j + 1) size_j\right) \right\}.$$

Следовательно, неравенства выполняются в шаре радиуса 1 относительно $(p'_j,\ C'_j,\ T')$, то есть $r = 1$.

Второй этап. Пусть $(p_1^*, \ldots, p_n^*, C_1^*, \ldots, C_n^*, T^*)$ обозначает «почти» оптимальное решение для (53)–(59), построенное методом эллипсоидов с точностью ε'. В результате получаем «почти» нижнюю оценку T^* на оптимальную длину расписания, C^*_{max}, т. е. $T^* \leq C^*_{max} + \varepsilon'$. Используя найденные длительности работ, строится допустимое расписание S с помощью алгоритма списочного типа, учитывающего частичный порядок:

всякий раз, когда подмножество процессоров становится свободным, алгоритм назначает жесткую работу, которая требует не больше процессоров, чем доступно, и все ее предшественники завершили выполнение (если это возможно). Время работы алгоритма составляет $O(n^3)$.

Лемма 6 *Для задачи* $P|size_j \leq qm, prec, energy|C_{\max}$ *алгоритм списочного типа, учитывающий частичный порядок, строит допустимое расписание* S *длины не превосходящей* $\left(\frac{(2-q)m}{(1-q)m+1}\right) T^* < \frac{2-q}{1-q} T^*$.

Доказательство. Пусть $t_1 < t_2 < \cdots < t_\mu < t_{\mu+1}$ соответствуют моментам, когда завершаются работы в расписании S, упорядоченные по возрастанию, $\mu \leq n$. Определим интервалы $I_{i-1} = (t_{i-1}, t_i]$, $i = 1, \ldots, \mu + 1$, где $t_0 := 0$ – время начала первой работы, а $t_{\mu+1} := L_S$ – время завершения последней работы в расписании S.

Будем рассматривать интервалы в обратном порядке $I_\mu, I_{\mu-1}, \ldots, I_2, I_1, I_0$ и разбивать их на два непересекающихся подмножества $\Im_1$ и $\Im_2$ следующим образом. Пусть j_1 обозначает работу, которая завершается в момент времени L_S и начинается в момент времени t_{k_1}. Интервалы $I_{k_1}, I_{k_1+1}, \ldots, I_\mu$ добавляются в подмножество $\Im_1$ и называются покрытыми работой j_1. Возможны две причины, по которым работа j_1 не ставится на выполнение в момент времени t_{k_1-1}:

(C1) работа j_2 выполняется в интервале I_{k_1-1}, и эта работа предшествует работе j_1;

(C2) работа j_1 может начать выполнение в момент t_{k_1-1}, но в интервале I_{k_1-1} имеется менее $size_{j_1}$ доступных процессоров.

В первом случае (C1) идентифицируем момент времени t_{k_2}, когда начинается работа j_2. Интервалы $I_{k_2}, I_{k_1+1}, \ldots, I_{k_1-1}$ добавляются в множество $\Im_1$, и называются покрытыми работой j_2. Далее переходим к проверке того, почему работа j_2 не начинается в момент времени t_{k_2-1} и так далее. Во втором случае (C2) интервал I_{k_1-1} добавляется в множество $\Im_2$. Далее переходим к интервалу I_{k_1-2} и проверяем, почему работа j_1 не начинается в момент времени t_{k_1-2} и так далее.

После выполнения вышеприведенных действий все интервалы I_i, $i = 0, \ldots, \mu$, будут разделены на два непересекающихся подмножества $\Im_1$

и $\Im_2$. Более того, будет идентифицирована цепочка работ $j_k \prec j_{k-1} \prec j_{k-2} \prec \cdots \prec j_1$ такая, что каждый интервал из $\Im_1$ покрывается одной из работ в цепочке. Пусть T_1 и T_2 обозначают общую длину всех интервалов из $\Im_1$ и $\Im_2$, соответственно. Тогда длина расписания

$$L_S = T_1 + T_2. \tag{60}$$

Из модели (53)–(59) имеем

$$T^* \geq C^*_{j_1} \geq C^*_{j_1} - C^*_{j_k} + p^*_{j_k} \geq \sum_{l=1}^{k-1} (C^*_{j_l} - C^*_{j_{l+1}}) + p^*_{j_k} \geq \sum_{l=1}^{k} p^*_{j_l} = T_1. \tag{61}$$

Кроме того, неравенство (54) гарантирует, что

$$T^* \geq \frac{1}{m} \sum_{j \in \mathcal{J}} p^*_j size_j. \tag{62}$$

Количество занятых процессоров в интервале $I_i \in \Im_2$ будет не менее $(1-q)m+1$. В противном случае в этом интервале могло бы выполняться больше работ, поскольку все работы $j \in \mathcal{J}$ имеют $size_j \leq qm$. Также, по крайней мере один процессор используется в течение каждого интервала $I_i \in \Im_1$. Следовательно, общая нагрузка

$$\sum_{j \in \mathcal{J}} p^*_j size_j \geq T_1 + ((1-q)m + 1)\, T_2 \tag{63}$$

Агрегируя выражения (60), (61), (62) и (63) получаем следующую оценку на отношение длины расписания L_S к нижней оценке T^*

$$\frac{L_S}{T^*} \leq \frac{T_1 + T_2}{\max\{T_1;\ T_1/m + (1 - q + 1/m)T_2\}} =: \rho. \tag{64}$$

Если $T_1 \geq \frac{T_1}{m} + \left(1 - q + \frac{1}{m}\right) T_2$, т. е.

$$\frac{T_2}{T_1} \leq \frac{1 - 1/m}{1 - q + 1/m},$$

то имеем

$$\rho = \frac{T_1 + T_2}{T_1} \leq 1 + \frac{1 - 1/m}{1 - q + 1/m}.$$

Если $T_1 \leq \frac{T_1}{m} + \left(1 - q + \frac{1}{m}\right) T_2$, т. е.

$$T_2 \geq T_1 \frac{1 - 1/m}{1 - q + 1/m},$$

то имеем

$$\rho = \frac{T_1 + T_2}{T_1/m + (1 - q + 1/m)T_2} \leq \frac{T_1 + T_1 \frac{1-1/m}{1-q+1/m}}{T_1/m + (1 - 1/m)T_1} = 1 + \frac{1 - 1/m}{1 - q + 1/m}.$$

Это завершает доказательство. □

Отметим, что оценка точности $\left(\frac{(2-q)m}{(1-q)m+1}\right)$ достигается в случае однопроцессорных работ с пустым частичным порядком, как можно видеть из примера в разделе 2.1.2.

При выборе точности $\varepsilon' = \left(\frac{1-q}{2-q}\varepsilon\right)$ в методе эллипсоидов для любого заданного $\varepsilon > 0$, получим следующий результат.

Теорема 13 *Для задачи* $P|size_j, prec, energy|C_{\max}$ *с* $size_j \leq qm$, $0 < q < 1$, *приближенное расписание длиной не более* $\left(\frac{2-q}{1-q}\right) C^*_{max} + \varepsilon$ *можно найти за время, полиномиальное от* $\log\frac{1}{\varepsilon}$ *и размера входа индивидуальной задачи.*

Жесткие работы с прерываниями и произвольными моментами поступления. В этом подразделе рассматривается задача с жесткими работами и возможностью прерываний. Предлагается полиномиальный приближенный алгоритм для случая, когда работы имеют произвольные моменты поступления $r_j \geq 0,\ j \in \mathcal{J}$.

Первый этап. Построим модель выпуклого программирования:

$$T \to \min, \tag{65}$$

$$r_j + \sum_{i \in R_j} p_i \leq T,\ j \in \mathcal{J}_{\lfloor \frac{m}{2} \rfloor}, \tag{66}$$

$$r_j + \frac{1}{m} \sum_{i \in \mathcal{J}:\ r_i \geq r_j} p_i size_i \leq T,\ j \in \mathcal{J}, \tag{67}$$

$$r_j + p_j \leq T,\ j \in \mathcal{J}, \tag{68}$$

$$\sum_{j \in \mathcal{J}} size_j p_j \left(\frac{W_j}{p_j}\right)^{\alpha} \leq E, \tag{69}$$

$$p_j \geq 0,\ j \in \mathcal{J}, \tag{70}$$

где $\mathcal{J}_{\lfloor \frac{m}{2} \rfloor} = \{j \in \mathcal{J} : size_j > \lfloor \frac{m}{2} \rfloor\}$, $R_j = \{i \in \mathcal{J}_{\lfloor \frac{m}{2} \rfloor} : r_i \geq r_j\}$ для $j \in \mathcal{J}_{\lfloor \frac{m}{2} \rfloor}$.

Таким образом, получаем задачу выпуклого программирования с полиномиальным числом ограничений. Ограничение (66) гарантирует, что не более одной работы из $\mathcal{J}_{\lfloor \frac{m}{2} \rfloor}$ может выполняться одновременно. Ограничение (67) показывает, что общий объем работ с моментом поступления не менее r_j не превышает $(T - r_j)m$.

Задача (65)-(70) может быть решена методом эллипсоидов [193] с произвольной заданной точностью $\varepsilon' > 0$ за полиномиальное время. Предварительно необходимо проверить два технических условия, как и в предыдущем случае. Первое условие следует из ограниченности сверху полиномом компонент оптимального решения: $p_j \leq \left(\frac{size_j W_j^{\alpha}}{E}\right)^{\frac{1}{\alpha-1}}$ для всех $j \in \mathcal{J}$ и $T \leq \max_{j \in \mathcal{J}} r_j + \sum_{j \in \mathcal{J}} \left(\frac{size_j W_j^{\alpha}}{E}\right)^{\frac{1}{\alpha-1}}$. Второе условие выполняется для точки (p'_j, T'), определяемой как

$$p'_j := W_j \left(\frac{\sum_i W_i size_i}{E}\right)^{\frac{1}{\alpha-1}} + 1$$

для всех $j \in \mathcal{J}$, и T' достаточно велико, чтобы

$$T' - 1 \geq \max \left\{ \max_{j \in \mathcal{J}_{\lfloor \frac{m}{2} \rfloor}} \left(r_j + \sum_{i \in R_j} (p'_i + 1) \right) ; \right.$$

$$\left. \max_{j \in \mathcal{J}} \left(r_j + \frac{1}{m} \sum_{i \in \mathcal{J}:\ r_i \geq r_j} (p'_i + 1) size_i \right) ; \max_{j \in \mathcal{J}} \left(r_j + (p'_j + 1) \right) \right\}.$$

Второй этап. Пусть $(p_1^*, \ldots, p_n^*, T^*)$ обозначает «почти» оптимальное решение для (65)-(70), то есть $T^* \leq C_{max}^* + \varepsilon'$. Полагаем длительность каждой работы j равной p_j^* и применяем алгоритм списочного типа с учетом размеров работ $size_j$ и возможности прерываний и получаем приближенное расписание. В каждый момент необходимости принятия решения (т. е. при поступлении в систему или завершении некоторой работы) все работы, которые выполняются в текущий момент, прерываются. Далее еще не завершенные работы рассматриваются в порядке невозрастания размеров $size_j$, и столько из них назначаются процессорам по жадному принципу, сколько это возможно. Временная сложность алгоритма составляет $O(n^2)$.

Лемма 7 *Для задачи* $P|r_j, size_j, pmtn, energy|C_{\max}$ *алгоритм списочного типа с учетом размеров работ* $size_j$ *и возможности прерываний позволяет построить допустимое расписание* S *длины не более* $\left(2 - \frac{1}{m}\right) T^*$.

Доказательство. Пусть l есть последняя работа в построенном расписании, а C_l обозначает время завершения работы l. Выделим два случая: (I) $size_l > \left\lfloor \frac{m}{2} \right\rfloor$ и (II) $size_l \leq \left\lfloor \frac{m}{2} \right\rfloor$. Будем назвать работу j большой, если $size_j > \left\lfloor \frac{m}{2} \right\rfloor$.

(I) $size_l > \left\lfloor \frac{m}{2} \right\rfloor$. Рассмотрим последний интервал, где в каждый момент времени выполняется некоторая большая работа. Пусть z будет первой большой работой в этом интервале. Из свойств алгоритма списочного типа с учетом размеров работ $size_j$ и возможностью прерываний имеем $C_l = r_z + (C_l - r_z) \leq r_z + \sum_{i \in R_z} p_i^* \leq T^*$, где последнее неравенство следует из ограничения (66).

(II) $size_l \leq \left\lfloor \frac{m}{2} \right\rfloor$. Пусть $[\tau, C_l]$ соответствует максимальному по длительности интервалу, где в каждый момент времени выполняется работа с $size_j \geq size_l$. Тогда общий объем процессорного времени по работам с моментами поступления, большими или равными τ, будет не менее

$$(r_l - \tau) size_l + (C_l - r_l - p_l^*)(m - size_l + 1) + p_l^* size_l \leq \sum_{i \in \mathcal{J}:\ r_i \geq \tau} p_i^* size_i. \quad (71)$$

Из ограничений (67) и (68) следуют неравенства:

$$r_l + p_l^* \le T^*,$$

$$\tau + \frac{1}{m} \sum_{i \in \mathcal{J}:\ r_i \ge \tau} p_i^* size_i \le T^*.$$

Пусть $\Delta = \max\{(r_l - \tau) + p_l^*, \frac{1}{m} \sum_{i \in \mathcal{J}:\ r_i \ge \tau} p_i^* size_i\}$. Тогда имеем $p_l^* + r_l \le \Delta + \tau$ и $\Delta + \tau \le T^*$.

Предположим, что $C_l > \left(2 - \frac{1}{m}\right)(\tau + \Delta)$. Тогда получаем

$$(r_l - \tau) size_l + (C_l - r_l - p_l^*)(m - size_l + 1) + p_l^* size_l$$

$$> (r_l - \tau) size_l + \left(\left(2 - \frac{1}{m}\right)(\tau + \Delta) - r_l - p_l^*\right)(m - size_l + 1) + p_l^* size_l$$

$$= \Delta m + (\Delta + \tau - p_l^* - r_l)(m - 2size_l + 1) + \tau(m - size_l) + \left(\frac{\tau}{m} + \frac{\Delta}{m}\right)(size_l - 1)$$

$$\ge \Delta m \ge \sum_{i \in \mathcal{J}:\ r_i \ge \tau} p_i^* size_i.$$

Второе неравенство с конца следует из $\Delta + \tau \ge p_l^* + r_l$, $size_l \le \left\lfloor \frac{m}{2} \right\rfloor$ и $size_l \ge 1$. Получаем противоречие с (71). Следовательно, $C_l \le \left(2 - \frac{1}{m}\right)(\tau + \Delta) \le \left(2 - \frac{1}{m}\right) T^*$. $\square$

Используя результаты из [120], приходим к выводу, что оценка точности $\left(2 - \frac{1}{m}\right)$ является достижимой даже в случае, если все работы являются однопроцессорными (см. пример в разделе 2.1.2).

Зададим точность $\varepsilon' = \frac{\varepsilon}{2}$ в методе эллипсоидов, тогда при любом заданном $\varepsilon > 0$ будет верна следующая теорема.

Теорема 14 *Для задачи* $P|r_j, size_j, pmtn, energy|C_{\max}$ *может быть построено расписание длиной не более* $\left(2 - \frac{1}{m}\right) C_{max}^* + \varepsilon$ *за время, полиномиальное от* $\log\frac{1}{\varepsilon}$ *и размера входа индивидуальной задачи.*

Напомним, что в разделе 2.1.2 представлен алгоритм с оценкой точности $\left(3 - \frac{1}{m}\right)$ для жестких работ без прерываний с произвольными моментами поступления.

Задача $P|r_j, size_j \in \{1, m\}, pmtn, energy|C_{\max}$**.** Теперь рассмотрим случай, когда каждая работа требует либо один, либо все процессоры. Покажем, что «почти» оптимальное расписание для задачи $P|r_j, size_j \in \{1, m\}, pmtn, energy|C_{\max}$ может быть найдено за полиномиальное время.

Обозначим через $\mathcal{J}_1$ множество однопроцессорных работ, а через $\mathcal{J}_m$ множество m-процессорных работ, $\mathcal{J} = \mathcal{J}_1 \cup \mathcal{J}_m$. Моменты поступления работ $r_j,\ j \in \mathcal{J}$, делят горизонт планирования на интервалы $I_i = [s_i, s_i + l_i)$, $i = 1, \ldots, \gamma \leq n$. Отметим, что длина последнего интервала l_γ заранее неизвестна и рассматривается здесь как переменная. Введем вещественные переменные $p_{ji} \geq 0$ (длительность работы j в интервале I_i) и построим модель выпуклого программирования

$$l_\gamma \to \min, \tag{72}$$

$$\frac{1}{m} \sum_{j \in \mathcal{J}_1} p_{ji} \leq l_i - \sum_{j \in \mathcal{J}_m} p_{ji},\ i = 1, \ldots, \gamma, \tag{73}$$

$$\max_{j \in \mathcal{J}_1} \{p_{ji}\} \leq l_i - \sum_{j \in \mathcal{J}_m} p_{ji},\ i = 1, \ldots, \gamma, \tag{74}$$

$$\sum_{j \in \mathcal{J}_1} \left(\sum_{i=1}^{\gamma} p_{ji} \right)^{1-\alpha} W_j^\alpha + \sum_{j \in \mathcal{J}_m} m \left(\sum_{i=1}^{\gamma} p_{ji} \right)^{1-\alpha} W_j^\alpha \leq E, \tag{75}$$

$$p_{ji} = 0,\ i = 1, \ldots, \gamma,\ j \in \{j' \in \mathcal{J} :\ r_{j'} > s_i\}. \tag{76}$$

Представленную модель можно решить методом эллипсоидов с любой заданной точностью $\varepsilon > 0$ за время, полиномиальное от $\log \frac{1}{\varepsilon}$ и размера входа задачи. Отметим, что верхние оценки для переменных и шар с допустимыми решениями строятся аналогично модели (65)-(70). Применяя метод эллипсоидов для решения задачи (72)-(76), получим длину расписания не более $s_\gamma + l_\gamma^* + \varepsilon$ и длительности фрагментов p_{ji} для каждой работы j. Затем для каждого интервала строится допустимое расписание, где сначала размещаются m-процессорные работы, а потом используется алгоритм из [90] для однопроцессорных работ.

Одномодальные двухпроцессорные работы без прерываний. NP-трудность задачи $P|fix_j|C_{\max}$ с $|fix_j| = 2$ была доказана с помощью полиномиальной сводимости известной задачи о реберной раскраске графа [74]. Используя предлагаемый в диссертации метод (см. раздел 1.2) и сводимость из [74], можно доказать, что задача составления расписаний $P|fix_j,\ |fix_j| = 2, energy|C_{\max}$ также является NP-трудной. Построим 2-приближенный алгоритм для этой задачи.

Обозначим через $\mathcal{J}_i = \{j \in \mathcal{J} :\ i \in fix_j\}$ подмножество работ, использующих процессор $i,\ i = 1, \ldots, m$. Построим следующую модель выпуклого программирования для получения нижней оценки на длину расписания

$$T \to \min, \tag{77}$$

$$\sum_{j \in \mathcal{J}_i} p_j \leq T,\ i = 1, \ldots, m, \tag{78}$$

$$\sum_{i=1}^{m} \sum_{j \in \mathcal{J}_i} p_j \left(\frac{W_j}{p_j}\right)^{\alpha} \leq E, \tag{79}$$

$$p_j \geq 0,\ j \in \mathcal{J}. \tag{80}$$

Задача (77)-(80) может быть решена методом эллипсоидов с любой заданной точностью за полиномиальное время. Пусть $(p_1^*, \ldots, p_n^*, T^*)$ обозначает «почти» оптимальное решение для (77)-(80) с точностью $\frac{\varepsilon}{2} > 0$.

Полагаем длительность каждой работы j равной p_j^* и применяем «fix_j-жадный» алгоритм, который назначает работу, как только появляются процессоры, необходимые для ее выполнения: В момент времени 0 и далее каждый раз, когда завершается какая либо работа, работы, требующие процессоры, доступные на текущий момент, ставятся в расписание.

Теперь докажем, что «fix_j-жадный» алгоритм строит 2-приближенное расписание.

Лемма 8 *«fix_j-жадный» алгоритм строит допустимое расписание S длиной не более $2T^*$ для задачи $P|fix_j, energy|C_{\max}$ с $|fix_j| = 2$.*

Доказательство. Пусть j_l соответствует работе с самым поздним временем завершения в расписании S, и $fix_{j_l} = \{M_1, M_2\}$. Тогда в любой момент времени до начала j_l должен быть занят либо процессор M_1,

либо процессор M_2. Но процессор M_1 может быть занят работами, отличными от j_l, не более $\sum_{j\in\mathcal{J}_{M_1}\setminus\{j_l\}} p_j^*$ единиц времени, и аналогичная оценка выполняется для M_2. Таким образом, длина S не больше, чем $\sum_{j\in\mathcal{J}_{M_1}\setminus\{j_l\}} p_j^* + \sum_{j\in\mathcal{J}_{M_2}\setminus\{j_l\}} p_j^* + p_{j_l}^* < 2T^*$ ввиду ограничения (78). □

Используя результаты из [74], можно показать, что оценка точности 2 для «fix_j-жадного» алгоритма является асимптотически точной. Действительно, рассмотрим пример с $n = k^2$ работами $W_j = 1$. Пусть число процессоров m равно $(k^2 + 1)$, а бюджет на расход энергии E равен $2k^2$. Положим $fix_j = \{k^2 + 1; j\}$, $j = 1, \ldots, k$, и $fix_{k+(k-1)(j-1)+i} = \{j; k + (k - 1)(j - 1) + i\}$, $j = 1, \ldots, k,\ i = 1, \ldots, k - 1$. В оптимальном расписании скорость каждого процессора равна 1, а работы добавляются в расписание в порядке $(1, 2, \ldots, k, k + 1, \ldots, k^2)$. Таким образом, оптимальная длина расписания составляет k. Из модели (77)-(80) для нижней оценки имеем $T^* = k$, $p_j^* = 1$ для всех $j \in \mathcal{J}$. Предположим, что работы упорядочены согласно последовательности $(k + 1, \ldots, k^2, 1, 2, \ldots, k)$. Тогда расписание, полученное с помощью «fix_j-жадного» алгоритма, имеет длину $2k - 1$, поскольку работы из $\{k + 1, \ldots, k^2\}$ выполняются в первую очередь в интервале $[0, k - 1)$ (см. рис. 7). Имеет место

Теорема 15 *Для задачи составления расписаний $P|fix_j, energy|C_{\max}$ с $|fix_j| = 2$ приближенное расписание длины не более $2C_{max}^* + \varepsilon$ можно построить за время, полиномиальное от $\log\frac{1}{\varepsilon}$ и размера входа индивидуальной задачи.*

2.1.3 Неравномерное распределение объема работы между процессорами

Результаты для жестких работ представлены для случая, когда общий объем работы может быть произвольным образом распределен между используемыми процессорами. Однако теоремы 12, 13 и 14 можно обобщить на случай с неравномерным распределением объема работы между процессорами. Пусть работа $j \in \mathcal{J}$ состоит из $size_j$ операций с объемами W_{jl}, $l = 1, \ldots, size_j$, и каждая операция должна выполняться на отдельном процессоре.

Необходимо выполнить корректировку первого этапа предложенных

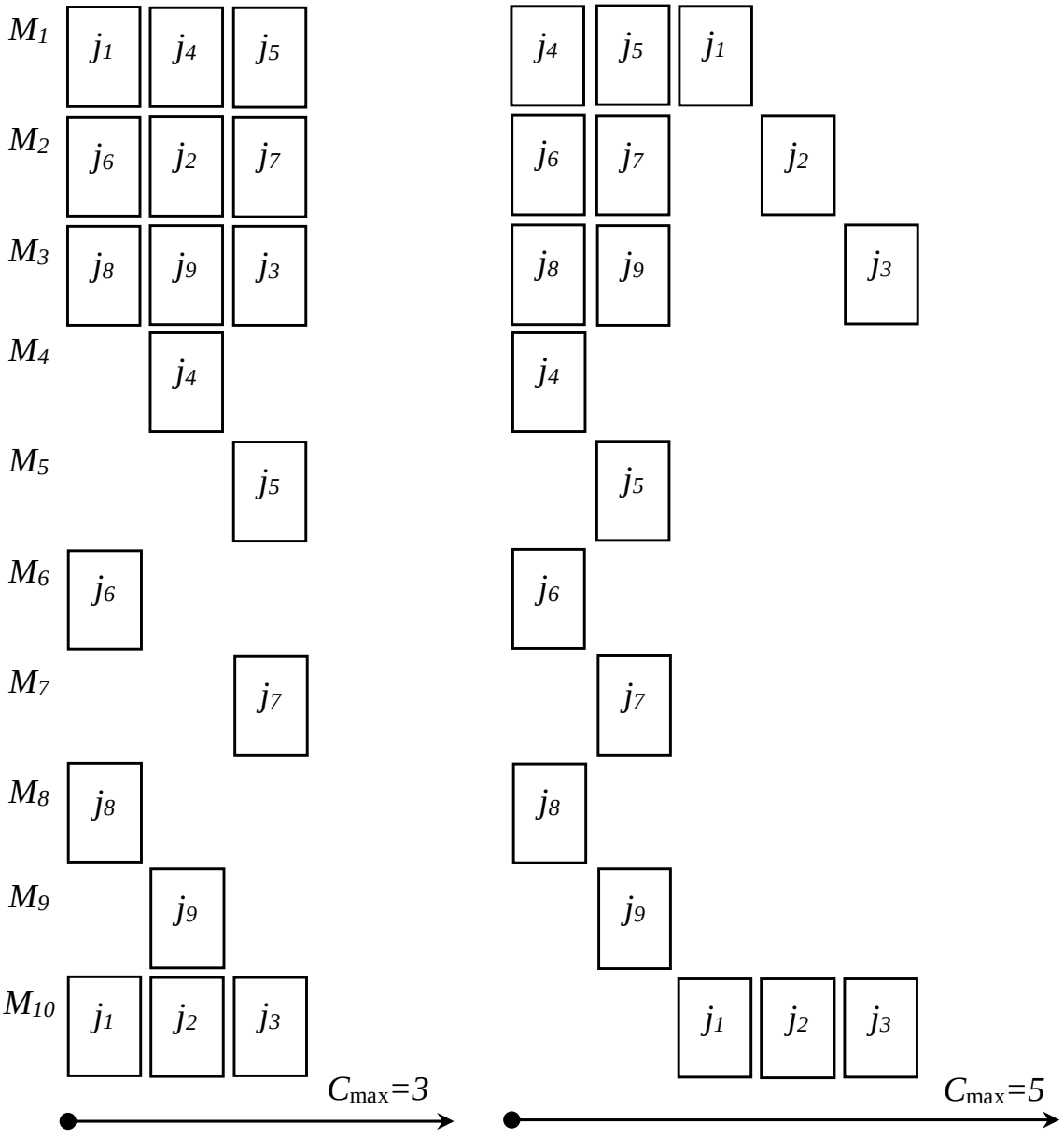


Рисунок 7 — Оптимальное расписание длины k и одно из возможных расписаний, полученных алгоритмом списочного типа, длины $2k-1$ при $k=3$.

алгоритмов. А именно, в моделях для построения нижней оценки изменяется ограничение на потребление энергии следующим образом

$$\sum_{j\in\mathcal{J}}\sum_{l=1}^{size_j} W_{jl}^{\alpha}\cdot p_j^{1-\alpha}\le E.$$

Теорему 15 можно обобщить на случай задачи составления расписаний без прерываний $P|fix_j, W_{ji}, energy, \alpha_i|C_{\max}$ с $|fix_j|=2$, где каждый процессор имеет индивидуальную функцию расчета мгновенной мощности s^{α_i} и объемы W_{ji} зависят от процессора. Требуется модифицировать ограничение (79) согласно правилу

$$\sum_{i=1}^{m}\sum_{j\in\mathcal{J}_i} p_j\left(\frac{W_{ji}}{p_j}\right)^{\alpha_i}\le E.$$

2.1.4 Задача линейного программирования с конфигурациями работ

Рассмотрим задачу с прерываниями и миграцией для жестких работ и одномодальных многопроцессорных работ. Сначала для случая дискретного набора скоростей процессоров и рациональных значений потребления энергии (обозначаемых размещением «*discr*» во втором поле обозначений задач) строятся «почти точные» алгоритмы на основе моделей линейного программирования с конфигурациями работ. Для непрерывного множества скоростей процессоров и произвольных интенсивностей потребления энергии разрабатывается приближенный алгоритм. Этот алгоритм использует дискретизацию скоростей и предыдущий подход.

Дискретный набор скоростей. Предположим, что процессоры могут работать со скоростями только из некоторого дискретного набора, а интенсивности потребления энергии являются рациональными числами. Сформулируем задачу линейного программирования с конфигурациями работ, где число переменных экспоненциально, а число ограничений полиномиально, и построим полиномиальный алгоритм решения на основе метода эллипсоидов.

Конфигурация C есть допустимое подмножество $\mathcal{J}_C$ работ, которые могут выполняться одновременно, совместно с назначением скоростей для процессоров. Обозначим через $\mathcal{C}$ множество всех возможных конфигураций. В допустимом расписании в каждый момент времени t может выполняться только одна конфигурация. Пусть $S_{j,C}$ есть скоростью работы j в конфигурации C, а $E_C = \sum_{j \in J_C} (S_{j,C})^\alpha m_j$ есть потребление энергии в единицу времени всеми процессорами, работающими в соответствии с конфигурацией C (E_C — рациональное число, поскольку мы предполагаем, что все $(S_{j,C})^\alpha$ рациональны). Здесь параметр $m_j = size_j$ для задачи $P|size_j, pmtn, energy, dscr|C_{\max}$, и $m_j = |fix_j|$ для задачи $P|fix_j, pmtn, energy, dscr|C_{\max}$.

Напомним, что рассматривается случай с дискретным набором скоростей процессоров. Общее число конфигураций конечно, но экспоненциально от размера входа задачи. Пусть переменная x_C задает общую длительность, когда процессоры работают в соответствии с конфигурацией C. Построим модель линейного программирования с конфигурациями работ:

$$\sum_{C \in \mathcal{C}} x_C \to \min, \tag{81}$$

$$\sum_{C \in \mathcal{C}:\ j \in J_C} \frac{S_{j,C} x_C}{W_j} \geq 1,\ j \in \mathcal{J}, \tag{82}$$

$$-\sum_{C \in \mathcal{C}} E_C x_C \geq -E, \tag{83}$$

$$x_C \geq 0,\ C \in \mathcal{C}. \tag{84}$$

Неравенство (83) гарантирует, что общее потребление энергии не превышает верхнюю границу E. Ограничение (82) задает условие, что каждая работа $j \in \mathcal{J}$ выполняется в требуемом объеме. Модель линейного программирования имеет экспоненциальное число переменных, но число ее ограничений линейно от n. В [123] показано, что если двойственная задача линейного программирования разрешима за полиномиальное время, то прямая задача также может быть решена за полиномиальное время (входные данные обеих задач – рациональные числа). Рассмотрим

двойственную задачу линейного программирования к (81)-(84):

$$\sum_{j\in\mathcal{J}} \lambda_j - E\mu \to \max, \tag{85}$$

$$\sum_{j\in J_C} \frac{S_{j,C}\lambda_j}{W_j} - E_C\mu \le 1,\ C \in \mathcal{C}, \tag{86}$$

$$\lambda_j \ge 0,\ j \in \mathcal{J}, \tag{87}$$

$$\mu \ge 0. \tag{88}$$

Двойственная задача имеет $O(n)$ переменных и экспоненциальное число ограничений. *Отделяющий оракул* – это алгоритм, который при заданном решении задачи линейного программирования, определяет допустимо это решение или нет, и в случае отрицательного ответа идентифицирует нарушенное ограничение. Далее будут построены полиномиальные отделяющие оракулы для двойственной задачи линейного программирования (85)–(88). Поэтому двойственная задача может быть решена методом эллипсоидов за полиномиальное время (см. [123]). Таким образом, оптимальное решение с заданной точностью $\varepsilon > 0$ может быть найдено для исходной (прямой) задачи (81)-(84) за полиномиальное время.

Отделяющий оракул для двойственной задачи линейного программирования работает следующим образом. Для поиска нарушенного ограничения достаточно найти минимум среди значений

$$\sum_{j\in J_C} \left(S^{\alpha}_{j,C} size_j \mu - \frac{S_{j,C}\lambda_j}{W_j} \right)$$

по всем конфигурациям C. Если это минимальное значение меньше (-1), то имеет место нарушенное ограничение. В противном случае решение двойственной задачи допустимо.

Для каждой работы $j \in \mathcal{J}$ величина $\left(S^{\alpha}_{j,C} size_j \mu - \frac{S_{j,C}\lambda_j}{W_j} \right)$ достигает минимального значения при дискретном значении v_j, которое отыскивается как одна из двух ближайших дискретных скоростей к значению

$\left(\frac{\lambda_j}{\alpha W_j m_j \mu}\right)^{\frac{1}{\alpha-1}}$.

Для работы j дискретное значение v_j вычисляется с помощью бинарного поиска за время, полиномиальное от длины входа задачи. В итоге, необходимо найти конфигурацию C, при которой минимизируется

$$\sum_{j \in J_C} \left(v_j^\alpha m_j \mu - \frac{v_j \lambda_j}{W_j} \right). \tag{89}$$

Задача минимизации (89) для $P|size_j, pmtn, energy, dscr|C_{\max}$ сводится к одномерной задаче о рюкзаке:

$$\sum_{j \in \mathcal{J}} b_j y_j \to \max,$$

$$\sum_{j \in \mathcal{J}} a_j y_j \leq m,$$

$$y_j \in \{0, 1\},\ j \in \mathcal{J},$$

где $b_j = \left(\frac{v_j \lambda_j}{W_j} - v_j^\alpha size_j \mu\right)$ и $a_j = size_j$ для всех $j \in \mathcal{J}$. Представленная задача может быть решена с помощью алгоритма динамического программирования за время, полиномиальное от m и размера входа задачи.

Лемма 9 *Для задачи $P|size_j, pmtn, energy, dscr|C_{\max}$ можно построить расписание длины не более $OPT + \varepsilon$ за время, полиномиальное от $\log\frac{1}{\varepsilon}$, m и размера входа задачи.*

В случае задачи $P|fix_j, pmtn, energy, dscr|C_{\max}$ с $|fix_j| = 2$, отделяющий оракул сводится к задаче поиска максимального взвешенного паросочетания на мультиграфе, где работа j задается ребром $\{fix_{j1}, fix_{j2}\}$ с весом $\left(\frac{v_j \lambda_j}{W_j} - 2\mu v_j^\alpha\right)$, где $fix_{j1}, fix_{j2} \in fix_j$. Набор ребер, формирующий паросочетание с максимальным суммарным весом, соответствует требуемой конфигурации работ и может быть найден за время полиномиальное от n.

Лемма 10 *Для задачи $P|fix_j, pmtn, energy, dscr|C_{\max}$ с $|fix_j| = 2$ можно построить расписание с длиной не более $OPT + \varepsilon$ за время, полиномиальное от $\log\frac{1}{\varepsilon}$ и размера входа индивидуальной задачи.*

Непрерывное множество скоростей. В случае непрерывного множества скоростей процессоров мощность $\mathcal{C}$ не ограничена, поскольку скорость процессора может принимать любые действительные значения. Выполним дискретизацию возможных значений для скорости и рассмотрим только конечное число скоростей, с которыми могут работать процессоры. То есть задача с непрерывным множеством скоростей сводится к задаче с дискретным набором скоростей процессоров и рациональными интенсивностями потребления энергии.

Для дискретизации скоростей найдем нижнюю и верхнюю границы на скорости процессоров. Для получения нижней границы S_{LB} предположим, что единственная работа с суммарным объемом $\sum_{j\in\mathcal{J}} W_j m_j$ потребляет весь доступный объем энергии E, тогда

$$s_{LB} = \left(\frac{E}{\sum_{j\in J} W_j m_j}\right)^{\frac{1}{\alpha-1}}. \tag{90}$$

Верхняя граница на скорость в свою очередь равна

$$s_{UB} = \left(\frac{E}{\min_{j\in J} W_j m_j}\right)^{\frac{1}{\alpha-1}}, \tag{91}$$

что соответствует случаю назначения всего объема энергии E работе с наименьшим значением $j' := \operatorname{argmin}_{j\in\mathcal{J}} W_j m_j$. Можно предположить, что нижняя и верхняя границы являются рациональными числами (могут быть вычислены одним из известных численных методов, см., например, метод Ньютона или метод дихотомии в [167]).

Рассмотрим указанные нижнюю и верхнюю границы и рациональную константу $\delta > 0$. Будем учитывать только скорости из набора $S_\delta = \{S_{LB}, (1+\delta)S_{LB}, (1+2\delta)S_{LB}, \ldots, (1+k\delta)S_{LB}\}$, где k – наименьшее целое число, такое что $(1+k\delta)S_{LB} \geq S_{UB}$. Следовательно, общее число скоростей ограничено величиной $k \leq \frac{S_{UB}}{\delta S_{LB}}$, что имеет полиномиальную зависимость от $\frac{1}{\delta}$ и экспоненциальную зависимость от размера входа.

Положим δ равным $\left(1+\frac{\theta}{E}\right)^{\frac{1}{\alpha+1}} - 1$, это рациональное число. Далее рассмотрим оптимальное расписание и округлим скорости процессоров до ближайших дискретных значений и увеличим интенсивности потреб-

ления энергии для округленных скоростей в $(1+\delta)$ раз. Общее потребление энергии увеличится, но не будет превышать $(1+\delta)^{\alpha+1}E = E+\theta$. Длительности работ и длина расписания не изменятся. Итак, рассматриваем набор скоростей S_δ и решаем заачу линейного программирования (81)-(84) с рациональными входными данными, где в неравенстве (83):

1. вместо E используется $(E+\theta)$ при ограничении на потребление энергии;

2. мгновенная мощность E_C конфигурации $C \in \mathcal{C}$ заменяется рациональным значением из интервала $[E_C, (1+\delta)E_C)$, вычисляемым за время, полиномиальное от $\log\frac{1}{\theta}$ и размера входа задачи с использованием одного из известных численных методов (см., например, метод Ньютона или метод дихотомии в [167]).

В результате строится расписание с потреблением энергии не больше $E+\theta$ и длины не больше $OPT+\varepsilon$ ввиду аддитивной ошибки ε метода эллипсоидов.

Теорема 16 *Для задачи $P|size_j, pmtn, energy|C_{\max}$ расписание с потреблением энергии не более $E+\theta$ и длительностью не более $OPT+\varepsilon$ можно найти за время, полиномиальное от m, $\log 1/\varepsilon$, $1/\theta$ и размера входа индивидуальной задачи.*

Теорема 17 *Для задачи $P|fix_j, pmtn, energy|C_{\max}$ с $|fix_j| = 2$ расписание с потреблением энергии не более $E+\theta$ и длительностью не более $OPT+\varepsilon$ можно найти за время, полиномиальное от $\log 1/\varepsilon$, $1/\theta$ и и размера входа индивидуальной задачи.*

Обобщения. Теоремы 16 и 17 можно обобщить на случай работ с неравномерным распределением объема между процессорами. В прямой задаче линейного программирования ограничение (82) для жестких работ заменяется на

$$\sum_{C\in\mathcal{C}:\ j\in C} \frac{S_{j,l,C}x_C}{W_{jl}} \geq 1,\ j \in \mathcal{J},\ l = 1,\ldots,size_j,$$

где $S_{j,l,C}$ — скорость операции l работы j в соответствии с конфигурацией C, а потребление энергии в единицу времени $E_C = \sum_{j\in J}\sum_{l=1}^{size_j}(S_{j,l,C})^\alpha$.

Аналогично, ограничение (82) и E_C модифицируются для одномодальных работ, состоящих из двух операций с индивидуальными объемами и заданными для выполнения процессорами. Двойственная задача строится и решается аналогично задаче с равномерным распределением объема работы между процессорами. В результате имеет место

Следствие 4 *Для задач* $Pm|size_j, pmtn, energy|C_{\max}$ *и* $P|fix_j, |fix_j| = 2, pmtn, energy|C_{\max}$ *с неравномерным распределением объема работ между процессорами можно построить расписание с потреблением энергии не более* $E + \theta$ *и длительностью не более* $OPT + \varepsilon$ *за время, полиномиальное от* $\log 1/\varepsilon$, $1/\theta$ *и размера входа индивидуальной задачи.*

Алгоритм можно также обобщить на задачи, где каждый процессор имеет индивидуальную функцию расчета мгновенной мощности s^{α_i}. Прямая задача линейного программирования для этого случая совпадает с (81)–(84) за исключением расчета $E_C = \sum_{j \in C} \left(\sum_{i \in I_{j,C}} (S_{j,C})^{\alpha_i} \right)$. Здесь $I_{j,C}$ – подмножество процессоров, используемых работой j в конфигурации C.

Следствие 5 *Для задач оптимизации* $P|fix_j, pmtn, energy, \alpha_i|C_{\max}$ *и* $P|size_j, pmtn, energy, \alpha_i|C_{\max}$ *с* $|fix_j| = 2$ *можно построить расписание с потреблением энергии не более* $E + \theta$ *и длительностью не более* $OPT + \varepsilon$ *за время, полиномиальное от* $\log 1/\varepsilon$, $1/\theta$ *и размера входа индивидуальной задачи.*

2.1.5 Нижняя оценка на целевую функцию

Построим алгоритм для поиска нижней оценки на длину расписания по модели (47)-(51). Рассмотрим следующие две подзадачи.

Подзадача (P1)

$$\frac{1}{m} \sum_{j \in \mathcal{J}} size_j p_j \to \min,$$

$$\sum_{j \in \mathcal{J}} size_j W_j^{\alpha} p_j^{1-\alpha} \le E,$$

$$p_j \ge 0, \ j \in \mathcal{J}.$$

Подзадача (P2)

$$\max_{j\in\mathcal{J}} p_j \to \min,$$

$$\sum_{j\in\mathcal{J}} size_j W_j^{\alpha} p_j^{1-\alpha} \le E,$$

$$p_j \ge 0,\ j \in \mathcal{J}.$$

Задачи (P1) и (P2) являются упрощенными версиями задачи (47)-(51). Поэтому, если оптимальное решение (P1) или (P2) является допустимым решением для (47)-(51), то оно также является оптимальным решением для (47)-(51).

Алгоритм для (47)-(51) состоит из трех шагов. На первых двух шагах решаются задачи (P1) и (P2), а на третьем шаге формируется агрегированная версия задач (P1) и (P2), полученная путем приравнивания значений целевых функций.

Первый шаг.

Перед решением задачи (P1) проверим следующее условие

$$\max_{j\in\mathcal{J}} W_j \le \frac{1}{m}\sum_{j\in\mathcal{J}} size_j W_j. \tag{92}$$

Если неравенство (92) имеет место, то оптимальное решение задачи (P1) является оптимальным решением задачи (47)-(51), иначе переходим ко второму шагу. Действительно, задача (P1) соответствует случаю, когда каждая жесткая работа j заменяется $size_j$ однопроцессорными работами той же длительности p_j, и требуется составить расписание для них на одном процессоре. Таким образом, в оптимальном решении (P1) все работы имеют одинаковую скорость s, которую можно найти с помощью уравнения

$$\sum_{j\in\mathcal{J}} size_j W_j s^{\alpha-1} = E.$$

Тогда длительности работ равны

$$p_i = \frac{W_i}{s} = \frac{W_i \left(\sum_{j\in\mathcal{J}} size_j W_j\right)^{\frac{1}{\alpha-1}}}{E^{\frac{1}{\alpha-1}}},\ i \in \mathcal{J}. \tag{93}$$

Условие (92) гарантирует, что неравенство $\max_{j\in\mathcal{J}} p_j \leq \frac{1}{m}\sum_{j\in\mathcal{J}} size_j p_j$ выполняется для длительностей работ (93).

Второй шаг.

Сначала проверяем условие

$$\sum_{j\in\mathcal{J}} size_j \leq m, \tag{94}$$

которое показывает, что оптимальное решение задачи (P2) является оптимальным решением задачи (47)-(51). Задача (P2) соответствует одновременному выполнению работ. В таком случае все работы должны иметь одинаковую длительность

$$p_i = \frac{\left(\sum_{j\in\mathcal{J}} size_j W_j^{\alpha}\right)^{\frac{1}{\alpha-1}}}{E^{\frac{1}{\alpha-1}}},\ i \in \mathcal{J}. \tag{95}$$

Условие (94) гарантирует, что $\frac{1}{m}\sum_{j\in\mathcal{J}} size_j p_j \leq \max_{j\in\mathcal{J}} p_j$ для длительностей работ (95). Однако, если общее количество процессоров, требуемых всеми работами, больше m, то переходим к третьему шагу.

Третий шаг.

Если ни неравенство (92), ни неравенство (94) не выполняются, то согласно необходимым и достаточным условиям Каруша-Куна-Таккера, для оптимального решения задачи (47)-(51) должно выполняться следующее равенство:

$$\frac{1}{m}\sum_{j\in\mathcal{J}} size_j p_j = \max_{j\in\mathcal{J}} p_j.$$

В результате получаем задачу (P3):

$$\sum_{j\in\mathcal{J}} size_j p_j \to \min,$$

$$\max_{j\in\mathcal{J}} p_j - \frac{1}{m}\sum_{j\in\mathcal{J}} size_j p_j = 0,$$

$$\sum_{j\in\mathcal{J}} size_j W_j^{\alpha} p_j^{1-\alpha} \le E,$$

$$p_j \ge 0,\ j \in \mathcal{J}.$$

Решение задачи (P3) начинается с оптимального решения (P1), где длительности работ равны

$$p_i = \frac{W_i \left(\sum_{j\in\mathcal{J}} size_j W_j\right)^{\frac{1}{\alpha-1}}}{E^{\frac{1}{\alpha-1}}},\ i \in \mathcal{J}.$$

Отметим, что длительности работ p_i пропорциональны объемам W_i. Упорядочим работы по невозрастанию объемов $W_1 \ge W_2 \ge \dots W_n$. Пусть l обозначает работу с минимальным объемом, таким что

$$W_l > \frac{1}{m}\sum_{j\in\mathcal{J}} size_j W_j. \tag{96}$$

Это условие эквивалентно условию $p_l > \frac{1}{m}\sum_{j\in\mathcal{J}} size_j p_j$.

Поскольку целевая функция в задаче (P3) является линейной, необходимым условием для оптимальности решения (P3) является

$$p_j = \frac{1}{m}\sum_{i\in\mathcal{J}} size_i p_i,\ j = 1,\dots,l.$$

Затем формулируем новую задачу (P4):

$$\sum_{j\in\mathcal{J}} size_j p_j \to \min,$$

$$p_j = \frac{1}{m}\sum_{i\in\mathcal{J}} size_i p_i,\ j = 1,\dots,l,$$

$$\sum_{j\in\mathcal{J}} size_j W_j^{\alpha} p_j^{1-\alpha} \le E,$$

$$p_j \ge 0,\ j \in \mathcal{J}.$$

Для нахождения оптимального решения задачи (P4) используем метод Лагранжа, построив функцию Лагранжа

$$L(p_j, \lambda_j) = \sum_{j\in\mathcal{J}} size_j p_j + \sum_{j=1}^{l} \lambda_j \left\{ p_j - \sum_{i\in\mathcal{J}} \frac{size_i p_i}{m} \right\} +$$

$$\lambda_{l+1} \left(\sum_{j\in\mathcal{J}} size_j W_j^{\alpha} p_j^{1-\alpha} - E \right)$$

Вычисляем частные производные, приравниваем их к нулю и находим длительности работ.

$$p_j = \frac{\sum_{i=l+1}^{n} size_i W_i}{(m - \sum_{i=1}^{l} size_i)} C,\ j = 1, \ldots, l, \tag{97}$$

$$p_j = W_j C,\ j = l+1, \ldots, n, \tag{98}$$

где

$$C = \left(\frac{E}{\sum_{j=1}^{l} \frac{W_j^{\alpha} size_j (\sum_{i=l+1}^{n} W_i size_i)^{1-\alpha}}{(m - \sum_{i=1}^{l} size_i)^{1-\alpha}} + \sum_{j=l+1}^{n} W_j size_j} \right)^{\frac{1}{1-\alpha}}.$$

Отметим, что $m > \sum_{i=1}^{l} size_i$ из-за условия (96) для выбора индекса l.

Если для полученных длительностей работ $l+1, \ldots, n$ выполняется следующее неравенство

$$\max_{j=l+1,\ldots,n} p_j \le \frac{1}{m} \sum_{j\in\mathcal{J}} size_j p_j, \tag{99}$$

то имеем оптимальное решение задачи (47)-(51). В противном случае переходим к решению задачи (P4) с новым значением l.

Неравенство (99) можно переписать в следующем виде, используя

полученные длительности работ (97), (98)

$$\max_{j=l+1,\dots,n} W_j = W_{l+1} \le \frac{1}{(m - \sum_{i=1}^{l} size_i)} \sum_{j=l+1}^{n} size_j W_j. \qquad (100)$$

Таким образом, вместо того, чтобы решать задачу (P4) несколько раз, необходимо только найти такое значение l, для которого выполняется условие (100). Это можно сделать за $O(\min\{n, m\})$ шагов. После этого, используя формулы (97), (98) для расчета длительностей, получаем оптимальное решение исходной задачи (47)-(51).

На каждом из трех шагов формулы для длительностей включают операции возведения в степень и извлечения корня. Поэтому $p_j,\ j \in \mathcal{J}$, может быть иррациональным и вычисляться с любой заданной точностью $\varepsilon' > 0$ за время $O\left(n\log\left(\frac{1}{\varepsilon'} nmW_{\max}\right)\right)$, где $W_{\max} = \max_{j\in\mathcal{J}} W_j$ (см. метод Ньютона или метод дихотомии в [167]). Если задать $\varepsilon' = \frac{\varepsilon}{n}$, то получим решение $OPT + \varepsilon$ задачи (47)-(51), поскольку критерий вычисляется как взвешенная сумма длительностей работ.

2.2 Сумма моментов завершения

Рассмотрим задачу составления расписаний с учетом расхода энергии и критерием минимизации суммы моментов завершения. Работы из множества $\mathcal{J} = \{1, \dots, n\}$ должны быть выполнены на m процессорах. Прерывания выполнения работ запрещены. Пусть V_j обозначает объем работы $j \in \mathcal{J}$. Рассматривается два основных варианта: жесткие работы ($size_j$) и одномодальные многопроцессорные работы (fix_j). Также исследуются постановки с гибкими (any) и мягкими (var) работами [90].

Исследуется однородная модель с непрерывным множеством скоростей процессоров. Если процессор работает со скоростью s, то потребление энергии в единицу времени (мгновенная мощность) равно s^α, где $\alpha > 1$ – заданная константа. Общее потребление энергии – это мгновенная мощность, интегрированная по времени. Процессор может выполнять работы с разной скоростью. Когда m_j процессоров используются для выполнения работы $j \in \mathcal{J}$, общий объем V_j равномерно распределяется между процессорами, которые работают с одинаковой скоростью

при выполнении этой работы. В таком случае $W_j := \frac{V_j}{m_j}$ обозначает объем работы j на одном процессоре. Задача заключается в составлении расписания (определение моментов начала и окончания выполнения работ, а также используемых ими процессоров), при котором минимизируется сумма моментов завершения выполнения работ $\sum C_j$ и суммарное потребление энергии не превышает бюджет E.

Используются следующие стандартные обозначения из теории расписаний $P|size_j, energy|\sum C_j$, $P|any, \delta_j, energy|\sum C_j$, $P|fix_j, energy|\sum C_j$ и $P|var, \delta_j, energy|\sum C_j$ для задач составления расписаний с ограничением на потребление энергии для жестких, гибких, одномодальных и мягких работ, соответственно. Здесь δ_j обозначает верхнюю границу на степень распараллеливания для работы $j \in J$.

2.2.1 Предыдущие исследования

Pruhs и др. [200] исследовали задачу минимизации среднего времени нахождения работ в системе на одном процессоре при ограничении на расход энергии и произвольных моментах поступления работ. Для случая работ единичных объемов предложен полиномиальный алгоритм, который строит расписание с минимальным средним временем нахождения работ в системе для различных уровней расхода энергии. В [62] алгоритм обобщен на многопроцессорный случай. Для составления расписаний с произвольными объемами работ на одном процессоре был разработан алгоритм с оценкой точности $O(1)$ при условии, что расход энергии может быть превышен не более чем в $(1+\varepsilon)$ раз. Albers и Fujiwara [20] рассматривали онлайн и оффлайн версии задачи на одном процессоре, где минимизируется энергопотребление в сумме со средним временем нахождения работ в системе. Для работ единичного объема были разработаны детерминированный онлайн алгоритм с константной оценкой точности и оффлайн алгоритм динамического программирования. В [39] авторы предложили онлайн алгоритм для задачи с учетом расхода энергии и работ произвольного объема. Алгоритм является $O(1)$-приближенным для агрегированного критерия, суммирующего взвешенное время нахождения работ в системе и расход энергии. На первом шаге решается упрощенная версия задачи с целевой функцией, суммирующей «дробное» взвешенное

время нахождения работ в системе и расход энергии. Затем решение, полученное на первом шаге, преобразуется в допустимое решение исходной задачи с потерей точности.

Shabtay и др. [216] исследовали близкую по постановке задачу составления расписаний однопроцессорных работ на идентичных параллельных процессорах, где длительности работ p_j зависят от потребления ресурса по правилу: $p_j(R_j) = \left(\frac{W_j}{R_j}\right)^{\kappa}$. Здесь W_j — объём работы работы j, R_j — объем ресурса, выделенного для выполнения работы j, $0 < \kappa \leq 1$ – заданная положительная константа. Авторами был предложен точный полиномиальный алгоритм для многопроцессорного случая без прерываний с критерием $\sum_j C_j$. Данный алгоритм может быть применён для решения аналогичной задачи с учетом расхода энергии при однопроцессорных работах.

Теперь представим известные результаты для распараллеливаемых и многопроцессорных работ с заданными длительностями и без учёта расхода энергии. Задача для жёстких работ без прерываний является NP-трудной в сильном смысле даже в случае двух процессоров [169], и NP-трудной для случая с прерываниями, когда количество процессоров является частью входа [89]. Для задачи без прерываний были предложены приближенные алгоритмы с гарантированными константными оценками точности. Это алгоритмы списочного типа [230], использующие стратегию минимизации среднего времени отклика (SMART) [214]. Задача составления расписаний для одномодальных работ без прерываний на двух процессорах является NP-трудной [132] и NP-трудной в сильном смысле для двухпроцессорных работ [164]. Преобразование расписания с прерываниями в расписание без прерываний позволяет получить 2-приближенное решение для случая двух процессоров [65], а метод типа «First Fit Coloring» гарантирует 2-приближённое решение для составления расписаний работ единичного объема на двух процессорах [116].

Как отмечалось ранее условия Каруша-Куна-Таккера (ККТ) являются необходимыми и достаточными для того, чтобы решения задач выпуклой оптимизации были оптимальными [57, 166]. Условия Каруша-Куна-Таккера часто применяются в алгоритмических структурах для задач составления расписаний при выпуклых функциях потребления ре-

сурсов. Более того, условия Каруша-Куна-Таккера позволяют выявить структурные свойства оптимальных расписаний и разработать комбинаторные алгоритмы малой трудоемкости. Примеры использования можно найти в [24,31,35,37,202,216] для задач с выпуклыми функциями потребления ресурсов и различными критериями. В настоящем разделе условия Каруша-Куна-Таккера используются для решения вспомогательных подзадач выпуклого программирования при вычислении верхних и нижних оценок для критерия при доказательстве NP-трудности и построении приближенных алгоритмов.

Основные результаты раздела. Доказывается NP-трудность задач составления расписаний $P|size_j \leq \frac{m}{2}, energy|\sum C_j$ и $P|fix_j,\ |fix_j| = 2, energy|\sum C_j$, и разрабатываются двухэтапные приближенные или точные алгоритмы для следующих частных случаев:

- жесткие работы,
- гибкие работы,
- ограниченное число параллельных и специализированных процессоров,
- однопроцессорный случай,
- полностью распараллеливаемые работы,
- несовместные работы.

На первом этапе вычисляется нижняя оценка на суммарное время завершения выполнения работ, отыскивается последовательность и длительности работ с помощью решения вспомогательной задачи выпуклого программирования. Затем, на втором этапе, задача формулируется в виде классической задачи составления расписаний без возможности варьирования скоростей процессоров и для ее решения применяются алгоритмы списочного типа или более сложные подходы.

2.2.2 Жесткие работы

Рассмотрим случай жестких работ. Докажем, что задача является NP-трудной и построим приближенный алгоритм для случаев, когда раз-

меры и объемы работ согласованы.

NP-трудность.

Теорема 18 *Задача* $P|size_j \le \frac{m}{2}, W_j = 1, energy|\sum C_j$ *является NP-трудной в сильном смысле.*

Доказательство. Докажем, что NP-полная в сильном смысле задача *3-Разбиение* [111] может быть сведена к распознавательной версии задачи $P|size_j \le \frac{m}{2}, W_j = 1, energy|\sum C_j$.

Рассмотрим индивидуальную задачу 3-Разбиение: Дано множество из $3q$ элементов с весами $a_j,\ j = 1, \ldots, 3q$, где $\sum_{j=1}^{3q} a_j = Bq$ и $\frac{B}{4} < a_j < \frac{B}{2}$. Можно ли разбить множество на q подмножеств $A_1, \ldots, A_q$ так, что $\sum_{j \in A_i} a_j = B$?

Построим индивидуальную задачу $P|size_j, energy|\sum C_j$. Пусть число работ $n = 3q$, число процессоров $m = B$ и граница на расход энергии $E = Bq$. Для каждого a_j создаем работу $j,\ j = 1, \ldots, 3q$. Положим $W_j = 1$, $size_j = a_j$, $V_j = a_j$ для $j \in \mathcal{J}$. В распознавательной версии $P|size_j, W_j = 1, energy|\sum C_j$ требуется ответить на вопрос: существует ли расписание со значением критерия $\sum C_j$, не превышающим заданный порог T?

Для определения значения T решаем вспомогательную задачу с $\sum_{j=1}^{3q} a_j$ однопроцессорными работами единичных объемов, т.е. каждая жесткая работа заменяется $size_j$ однопроцессорными работами. Такая задача имеет единственное оптимальное решение (с точностью до размещения работ на процессорах и перестановки работ на каждом процессоре), где каждый процессор выполняет q работ и потребляет q единиц энергии. Найдем оптимальные длительности работ на каждом процессоре, используя следующую задачу выпуклого программирования:

$$\sum_{j=1}^{q} p_j(q - j + 1) \to \min, \tag{101}$$

$$\sum_{j=1}^{q} p_j^{1-\alpha} = q, \tag{102}$$

$$p_j \ge 0,\ j = 1, \ldots, q. \tag{103}$$

Здесь p_j – длительность j-й работы на процессоре, $j = 1, \ldots, q$.

Составляем функцию Лагранжа

$$L(p_j, \lambda) = \sum_{j=1}^{q} p_j(q - j + 1) +$$

$$\lambda \left(\sum_{j=1}^{q} p_j^{1-\alpha} - q \right).$$

Приравниваем частные производные к нулю:

$$\frac{\partial L}{\partial p_j} = (q - j + 1) + \lambda(1 - \alpha)p_j^{-\alpha} = 0,\ p_j = \left(\frac{\lambda(\alpha - 1)}{q - j + 1} \right)^{1/\alpha},\ j = 1, \ldots, q.$$

Подставляем полученное выражение в (102) и получаем

$$\sum_{j=1}^{q} \left(\frac{\lambda(\alpha - 1)}{q - j + 1} \right)^{1-\alpha/\alpha} = q,\ (\lambda(\alpha-1))^{1/\alpha} = \left(\frac{\sum_{j=1}^{q}(q - j + 1)^{\alpha-1/\alpha}}{q} \right)^{1/\alpha-1}.$$

Затем вычисляем оптимальное решение:

$$p_j^* = \frac{\left(\sum_{j=1}^{q}(q - j + 1)^{\frac{\alpha-1}{\alpha}} \right)^{\frac{1}{\alpha-1}}}{q^{\frac{1}{\alpha-1}}(q - j + 1)^{\frac{1}{\alpha}}},\ j = 1, \ldots, q,$$

$$\sum C_j^* = \sum_{j=1}^{q} p_j(q - j + 1) = \left(\sum_{j=1}^{q}(n - j + 1)^{\frac{\alpha-1}{\alpha}} \right)^{\frac{\alpha}{\alpha-1}} q^{\frac{1}{1-\alpha}}.$$

Отметим, что каждая следующая работа имеет большую длительность, чем предыдущая. В оптимальном расписании процессоры не простаивают. Оптимальное суммарное время завершения для всех процессоров равно $m \sum C_j^*$. Положим пороговое значение $T := 3 \sum C_j^*$, так как одновременно может выполняться не более трех жестких работ.

Покажем, что положительный ответ в построенной задаче составления расписаний $P|size_j, energy| \sum C_j$ с $\sum C_j \leq T$ приводит к положительному ответу в задаче 3-Разбиение и наоборот. Действительно,

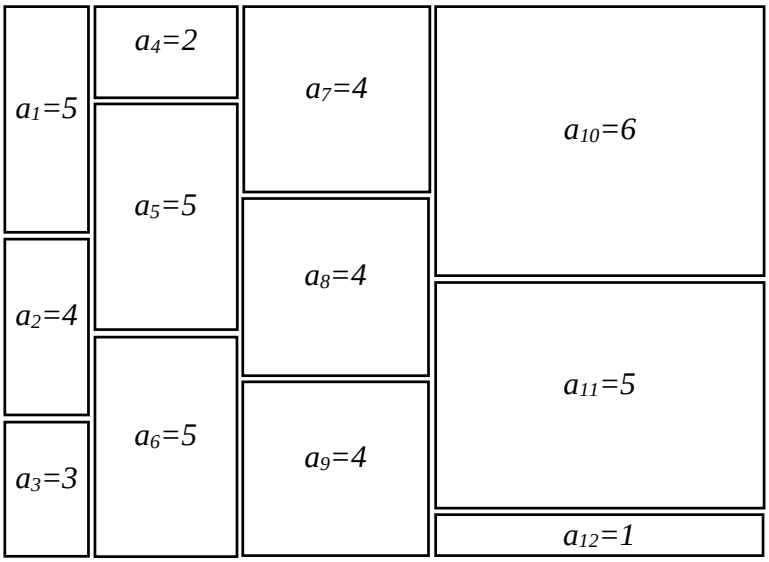


Рисунок 8 — Допустимое решение, соответствующее положительному ответу в задаче 3-Разбиение: $A_1 = \{1, 2, 3\}$, $A_2 = \{4, 5, 6\}$, $A_3 = \{7, 8, 9\}$, $A_4 = \{10, 11, 12\}$, $\sum_{j \in A_i} a_j = 12$.

$\sum C_j \geq 3 \sum C_j^*$ для любого допустимого расписания в указанной индивидуальной задаче ранее представленного упрощения при переходе к однопроцессорным работам с тем же общим объемом. Таким образом, допустимое расписание с $\sum C_j = T$ существует тогда и только тогда, когда все процессоры выполняют q работ, и каждые три работы, выполняемые параллельно и использующие все доступные процессоры, соответствуют трем элементам, образующим множество A_i, такое что $\sum_{j \in A_i} a_j = B$, $i = 1, \ldots, q$. Это расписание согласуется с оптимальным расписанием соответствующей задачи с однопроцессорными работами (см. пример на рис. 8).

Представленная сводимость является полиномиальной. Таким образом, задача $P|size_j \leq \frac{m}{2}, W_j = 1, energy| \sum C_j$ является NP-трудной в сильном смысле. □

Далее построим приближенный алгоритм для рассматриваемой задачи с жесткими работами, удовлетворяющими специальным свойствам по объемам и размерам. С помощью вспомогательной задачи выпуклого программирования вычислим скорости работ. Расписание с гарантированной оценкой точности будет строиться алгоритмом списочного типа.

Полиномиальный приближенный алгоритм. Порядок выполнения работ и, как следствие, время их завершения играют важную роль в задачах с критерием минимизации суммарного времени завершения. Рассмотрим сначала случай, когда порядок моментов начала работ задан, и вычислим нижнюю границу на сумму моментов завершения, игнорируя «прямоугольную» природу работ (длительность-размер). Предположим, что работы начинаются в расписании в соответствии с перестановкой $\pi = (\pi_1, \ldots, \pi_n)$. Используя нижнюю оценку на целевую функцию, представленную в [230] для жестких работ с заданными длительностями, построим задачу выпуклого программирования:

$$LB(\pi) = \frac{1}{m}\sum_{j=1}^{n}\sum_{i=1}^{j} size_{\pi_i} p_{\pi_i} + \frac{1}{2}\sum_{j=1}^{n} p_{\pi_j} - \frac{1}{2m}\sum_{j=1}^{n} size_{\pi_j} p_{\pi_j} \to \min, \quad (104)$$

$$\sum_{j=1}^{n} W_j^{\alpha} p_j^{1-\alpha} size_j \le E. \quad (105)$$

$$p_j \ge 0, \ j \in \mathcal{J}. \quad (106)$$

Здесь π_i – i-я работа в соответствии с перестановкой π, p_{π_i} – время выполнения работы π_i на каждом из используемых процессоров. Отметим, что для жестких работ $W_i = \frac{V_i}{size_i}$.

Лемма 11 *Задача выпуклого программирования (104)-(106) при заданной перестановке работ π полиномиально разрешима и оптимальное значение целевой функции равно*

$$LB^*(\pi) = \frac{E^{\frac{1}{1-\alpha}}}{m}\left(\sum_{i=1}^{n} W_{\pi_i} size_{\pi_i}\left(n - i + 0.5 + \frac{0.5m}{size_{\pi_i}}\right)^{\frac{\alpha-1}{\alpha}}\right)^{\frac{\alpha}{\alpha-1}}. \quad (107)$$

Доказательство. Решаем задачу (104)-(106) с помощью метода Лагранжа. Выписываем функцию Лагранжа $L(p_{\pi_j}, \lambda)$

$$L(p_{\pi_j}, \lambda) = \frac{1}{m}\sum_{j=1}^{n}\sum_{i=1}^{j} size_{\pi_i} p_{\pi_i} + \frac{1}{2}\sum_{j=1}^{n}\left(1 - \frac{size_{\pi_j}}{m}\right) p_{\pi_j} +$$

$$\lambda \left(\sum_{j=1}^{n} W_{\pi_j}^{\alpha} p_{\pi_j}^{1-\alpha} size_{\pi_j} - E \right).$$

Поскольку целевая функция (104) является линейной, а ограничение (105) представлено выпуклой функцией, необходимые и достаточные условия для оптимальности решения имеют вид (частные производные равны нулю) [57, 166]:

$$\frac{\partial L}{\partial p_{\pi_i}} = \frac{1}{m} size_{\pi_i}(n-i+1) + \frac{1}{2}\left(1 - \frac{size_{\pi_i}}{m}\right) + \lambda W_{\pi_i}^{\alpha} size_{\pi_i}(1-\alpha) p_{\pi_i}^{-\alpha} = 0,$$

$$i = 1, \ldots, n.$$

Переписывая выражения, получаем

$$p_{\pi_i} = ((\alpha-1)\lambda m)^{\frac{1}{\alpha}} \frac{W_{\pi_i}}{\left(n - i + 0.5 + \frac{0.5m}{size_{\pi_i}}\right)^{\frac{1}{\alpha}}}, \tag{108}$$

$$i = 1, \ldots, n.$$

Длительности (108) подставляем в уравнение

$$\frac{\partial L}{\partial \lambda} = \sum_{j=1}^{n} W_j^{\alpha} p_j^{1-\alpha} size_j - E = 0.$$

Это приводит к следующему уравнению

$$((\alpha-1)\lambda m)^{\frac{1-\alpha}{\alpha}} \sum_{j=1}^{n} W_{\pi_j} size_{\pi_j} \left(n - j + 0.5 + \frac{0.5m}{size_{\pi_j}}\right)^{\frac{\alpha-1}{\alpha}} = E.$$

В результате вычисляем значение

$$((\alpha-1)\lambda m)^{\frac{1}{\alpha}} = E^{\frac{1}{1-\alpha}} \left(\sum_{j=1}^{n} W_{\pi_j} size_{\pi_j} \left(n - j + 0.5 + \frac{0.5m}{size_{\pi_j}}\right)^{\frac{\alpha-1}{\alpha}} \right)^{\frac{1}{\alpha-1}}$$

и длительность работ из (108) выражаем как

$$p_{\pi_i} = \frac{E^{\frac{1}{1-\alpha}} W_{\pi_i}}{\left(n - i + 0.5 + \frac{0.5m}{size_{\pi_i}}\right)^{\frac{1}{\alpha}}} \cdot \left(\sum_{j=1}^{n} \frac{W_{\pi_j} size_{\pi_j}}{\left(n - j + 0.5 + \frac{0.5m}{size_{\pi_j}}\right)^{\frac{1-\alpha}{\alpha}}} \right)^{\frac{1}{\alpha-1}}, \quad (109)$$

$$i = 1, \ldots, n.$$

Полученные значения длительностей подставляются в выражение (104) и нижняя оценка на $\sum C_j$ для произвольной перестановки работ π вычисляется следующим образом

$$LB^*(\pi) = \frac{E^{\frac{1}{1-\alpha}}}{m} \left(\sum_{i=1}^{n} W_{\pi_i} size_{\pi_i} \left(n - i + 0.5 + \frac{0.5m}{size_{\pi_i}} \right)^{\frac{\alpha-1}{\alpha}} \right)^{\frac{\alpha}{\alpha-1}}.$$

□

Лемма 12 *Пусть выполняется условие о том, что упорядочение работ по неубыванию объемов V_j соответствует упорядочению работ по неубыванию размеров $size_j$ (то есть $V_i < V_j$ соответствует $size_i \leq size_j$ для $i \neq j \in \mathcal{J}$). Тогда минимальное значение $LB^*(\pi)$ достигается на перестановке, в которой работы упорядочены по неубыванию размеров $size_j$.*

Доказательство. Минимизация (107) эквивалентна минимизации

$$G(\pi) = \sum_{i=1}^{n} V_{\pi_i} \left(n - i + 0.5 + \frac{0.5m}{size_{\pi_i}} \right)^{\frac{\alpha-1}{\alpha}}.$$

среди всех возможных перестановок π. Пусть π^* задает перестановку работ по неубыванию размеров $size_i$. Рассмотрим две последовательные работы в π^*, например, π_i^* и π_{i+1}^*, для которых $size_{\pi_i^*} = x$, $size_{\pi_{i+1}^*} = y$, $V_{\pi_i^*} = V_x$, $V_{\pi_{i+1}^*} = V_y$ и $x \leq y$, $V_x \leq V_y$. Эти работы приводят к

$$g(\pi^*, i) = V_x \left(n - i + 0.5 + \frac{0.5m}{x} \right)^{\frac{\alpha-1}{\alpha}} + V_y \left(n - i - 0.5 + \frac{0.5m}{y} \right)^{\frac{\alpha-1}{\alpha}}$$

в $G(\pi^*)$. Пусть $\pi' = (\pi_1^*, \ldots, \pi_{i-1}^*, \pi_{i+1}^*, \pi_i^*, \pi_{i+2}^*, \ldots, \pi_n^*)$ и

$$g(\pi', i) = V_y \left(n - i + 0,5 + \frac{0,5m}{y} \right)^{\frac{\alpha-1}{\alpha}} + V_x \left(n - i - 0,5 + \frac{0,5m}{x} \right)^{\frac{\alpha-1}{\alpha}}.$$

Тогда

$$g(\pi^*, i) - g(\pi', i) =$$

$$V_x \left(\left(n - i + 0,5 + \frac{0,5m}{x} \right)^{\frac{\alpha-1}{\alpha}} - \left(n - i - 0,5 + \frac{0,5m}{x} \right)^{\frac{\alpha-1}{\alpha}} \right)$$

$$-V_y \left(\left(n - i + 0.5 + \frac{0.5m}{y} \right)^{\frac{\alpha-1}{\alpha}} - \left(n - i - 0.5 + \frac{0.5m}{y} \right)^{\frac{\alpha-1}{\alpha}} \right) \leq$$

$$(V_x - V_y) \left(\left(n - i + 0.5 + \frac{0.5m}{y} \right)^{\frac{\alpha-1}{\alpha}} - \left(n - i - 0.5 + \frac{0.5m}{y} \right)^{\frac{\alpha-1}{\alpha}} \right) \leq 0.$$

Первое неравенство следует из леммы 13. Следовательно, $G(\pi^*) \leq G(\pi')$. $\square$

Лемма 13 *Функция* $f(x) = \left(a + \frac{b}{x}\right)^{\frac{\alpha-1}{\alpha}} - \left(a - 1 + \frac{b}{x}\right)^{\frac{\alpha-1}{\alpha}}$ *является возрастающей при* $x \geq 1$, $\alpha > 1$, $a > 1$ *и* $b > 0$.

Доказательство. Вычисляем производную

$$f'(x) = -\frac{b}{x^2} \cdot \frac{\alpha - 1}{\alpha} \left(a + \frac{b}{x} \right)^{\frac{-1}{\alpha}} + \frac{b}{x^2} \cdot \frac{\alpha - 1}{\alpha} \left(a - 1 + \frac{b}{x} \right)^{\frac{-1}{\alpha}} =$$

$$\frac{b}{x^2} \cdot \frac{\alpha - 1}{\alpha} \left(\left(a - 1 + \frac{b}{x} \right)^{\frac{-1}{\alpha}} - \left(a + \frac{b}{x} \right)^{\frac{-1}{\alpha}} \right).$$

В результате имеем $f'(x) > 0$. $\square$

Далее будем полагать, что упорядочение работ по неубыванию V_j соответствует их упорядочению по неубыванию $size_j$ (т. е. объемы и размеры размеры согласованы). Пусть работы упорядочены по неубываюнию размеров $size_j$. Обозначим через $\bar{p}_j$ длительности работ, а через $LB(\bar{p}_j) = \sum_{j=1}^{n} \left(\frac{1}{m} \sum_{i=1}^{j-1} size_i \bar{p}_i + \frac{1}{2} \bar{p}_j + \frac{1}{2m} size_j \bar{p}_j \right)$ нижнюю оценку, соответствующую оптимальному решению задачи (104)-(106).

Расписание строится с помощью алгоритма списочного типа: Первая работа ставится на выполнение в момент времени 0. Следующая работа

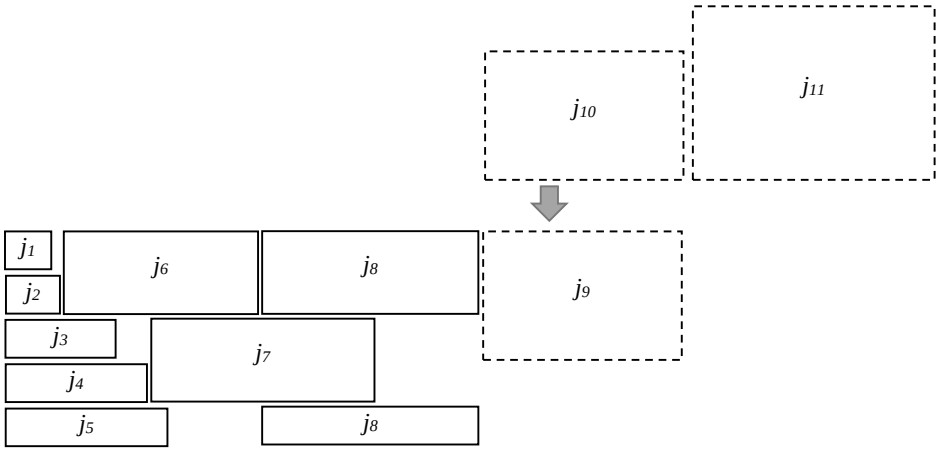


Рисунок 9 — Часть расписания, построенного алгоритмом списочного типа.

ставится в расписание в самый ранний момент, когда появляется достаточно процессоров для ее выполнения (см. пример на рис. 9). Трудоемкость алгоритма составляет $O(n^2)$.

Лемма 14 *Пусть выполняется условие, что из* $V_i < V_j$ *следует* $size_i \leq size_j$ *для* $i \neq j \in \mathcal{J}$. *Алгоритм списочного типа строит для задачи* $P|size_j \leq \frac{m}{2}, energy|\sum C_j$ *допустимое расписание* S *с суммарным временем завершения не превосходящим* $2LB(\bar{p}_j)$.

Доказательство. Каждая работа задействует при выполнении не более половины процессоров. Таким образом, время начала работы j не превышает $2\left(\frac{1}{m}\sum_{i=1}^{j-1} size_i\bar{p}_i\right)$, так как в каждый момент времени в расписании S задействовано не менее $\frac{m}{2}$ процессоров. Время завершения работы j удовлетворяет условию $C_j \leq \frac{2}{m}\sum_{i=1}^{j-1} size_i\bar{p}_i + \bar{p}_j$. В сумме имеем

$$\sum_{j\in\mathcal{J}} C_j \leq 2\left(\frac{1}{m}\sum_{j=1}^{n}\sum_{i=1}^{j-1} size_i\bar{p}_i + \frac{1}{2}\sum_{j=1}^{n}\bar{p}_j\right).$$

Сравниваем это значение с нижней границей $LB(\bar{p}_j)$ и получаем $\sum_{j\in\mathcal{J}} C_j \leq 2LB(\bar{p}_j)$. □

Лемма 15 *Пусть выполняется условие, что из* $W_i < W_j$ *следует* $size_i \leq size_j$ *для* $i \neq j \in \mathcal{J}$. *Алгоритм списочного типа строит для задачи* $P|size_j, energy|\sum C_j$ *допустимое расписание* S *с общим временем завершения не более* $2LB(\bar{p}_j)$.

Доказательство. Отметим, что длительности работ $\bar{p}_j$ в указанном случае не уменьшаются (см. формулу (109)). Если в каждый момент времени в S занято не менее $\frac{m}{2}$ процессоров, то можно воспользоваться теми же свойствами, что и в предыдущей лемме.

В противном случае предположим, что l — последняя работа, требующая менее $\frac{m}{2}$ процессоров. По построению в S менее $\frac{m}{2}$ процессоров может быть занято только в интервале $I := [C_l - \bar{p}_l, C_{l+1} - \bar{p}_{l+1})$. Обозначим длину интервала I через Δ, и пусть h будет минимальным числом процессоров, используемым в I. Отметим, что $size_l \leq h < \frac{m}{2}$, поскольку работа l выполняется в течение всего интервала I.

Можно показать, что в каждый момент времени в интервале $[0, C_l - p_l)$ согласно расписанию S используется не менее $m - size_l + 1 \geq m - h + 1$ процессоров. (В противном случае работа l должна быть назначена на выполнение раньше.) Более того, не менее $m - h + 1$ процессоров используется в интервале $[C_{l+1} - \bar{p}_{l+1}, C_n)$, поскольку работа $(l+1)$ не выполняется в интервале I и $size_j \geq size_{l+1}$ для всех $j > l + 1$.

Рассмотрим работу $j > l$. Длительность этой работы не больше, чем

$$\frac{1}{m-h+1}\left(\sum_{i=1}^{j-1} size_i \bar{p}_i - \Delta h\right) + \Delta + \bar{p}_j.$$

Теперь докажем, что $\Delta - \frac{h}{m-h+1}\Delta + \bar{p}_j \leq 2\left(\frac{1}{2}\bar{p}_j + \frac{size_j \bar{p}_j}{2m}\right)$. Действительно, ввиду $\Delta \leq \bar{p}_l < \bar{p}_j$, $size_j + h > m$ и $m > 2h$ имеем

$$\Delta - \frac{h}{m-h+1}\Delta + \bar{p}_j < \left(\frac{m-2h+1}{m-h+1} + 1\right)\bar{p}_j = \frac{2m-3h+2}{m-h+1}\bar{p}_j =$$

$$\frac{(2m-3h+2)(2m)}{(m-h+1)(2m-h)} \cdot \frac{(2m-h)}{(2m)}\bar{p}_j =$$

$$\frac{4m^2-6mh+4m}{2m^2-3mh+2m+(h^2-h)}\left(\frac{1}{2} + \frac{m-h}{2m}\right)\bar{p}_j \leq$$

$$2\left(\frac{1}{2} + \frac{m-h}{2m}\right)\bar{p}_j < 2\left(\frac{1}{2}\bar{p}_j + \frac{size_j \bar{p}_j}{2m}\right).$$

Следовательно, время завершения работы $j > l$ меньше

$$2\left(\frac{1}{m}\sum_{i=1}^{j-1} size_i\bar{p}_i + \frac{1}{2}\bar{p}_j + \frac{size_j\bar{p}_j}{2m}\right).$$

Более того, время завершения работы $j \leq l$ не больше, чем $\frac{2}{m}\sum_{i=1}^{j-1} size_i\bar{p}_i + \bar{p}_j$, так как в интервале $[0, C_l - p_l)$ задействовано не менее $m - size_l + 1 > \frac{m}{2}$. Сумма моментов завершения работ удовлетворяет условиям

$$\sum_{j\in\mathcal{J}} C_j < 2\sum_{j=1}^{n}\left(\frac{1}{m}\sum_{i=1}^{j-1} size_i\bar{p}_i + \frac{1}{2}\bar{p}_j + \frac{size_j\bar{p}_j}{2m}\right).$$

□

Таким образом, имеют место теоремы.

Теорема 19 *Для задачи* $P|size_j \leq \frac{m}{2}, energy|\sum C_j$ *можно построить* 2*-приближенное расписание за полиномиальное время при условии, что неубывающему порядку* V_j *соответствует неубывающий порядок* $size_j$.

Теорема 20 *Для задачи* $P|size_j, energy|\sum C_j$ *можно построить* 2*-приближенное расписание за полиномиальное время при условии, в что неубывающему порядку* $W_j = \frac{V_j}{size_j}$ *соответствует неубывающий порядок* $size_j$.

Если $size_i \leq size_j$ предполагает $W_i < W_j$, то имеет место $V_i = W_i size_i < V_j = size_j W_j$, но обратное утверждение в общем случае неверно. Таким образом, условие на входные данные в теореме 20 гарантирует выполнение условий в теореме 19, но не наоборот.

2.2.3 Двухпроцессорный случай

Перейдем к рассмотрению двухпроцессорной задачи составления расписаний $P2|size_j, energy|\sum C_j$ с произвольными объемами работ и построим 2-приближенный алгоритм. Алгоритм основан на следующей идее. Для индивидуальной задачи $P2|size_j, energy|\sum C_j$ строится однопроцессорная задача $1|energy|\sum C_j$, такая, что оптимальное значение целевой функции является нижней оценкой на целевую функцию для задачи $P2|size_j, energy|\sum C_j$. Затем формируется допустимое решение для

исходной двухпроцессорной задачи, используя тот же порядок работ, что и в оптимальной последовательности для $1|energy|\sum C_j$.

Пусть $(P2)$ обозначает исходную задачу $P2|size_j, energy|\sum C_j$ с объемами работ W_j и бюджетом на расход энергии E. Сформулируем однопроцессорную задачу $(P1)$ с объемами работ W'_j и бюджетом на расход энергии E', построенную из $(P2)$, положив $W'_j = \frac{W_j}{2}$ для однопроцессорных работ, $W'_j = W_j$ для двухпроцессорных работ и $E' = \frac{E}{2}$. Поскольку $(P1)$ включает в себя только однопроцессорные работы, то упорядочение работ по неубыванию объемов соответствует оптимальному решению, которое можно найти за полиномиальное время (см. раздел 2.2.6).

Следующая лемма показывает, что оптимальное значение целевой функции $\sum C_j^*(P1)$ для $(P1)$ является нижней оценкой на оптимальное значение $\sum C_j^*(P2)$ для $(P2)$.

Лемма 16 $\sum C_j^*(P1) \leq \sum C_j^*(P2)$.

Доказательство. Пусть $S2$ есть произвольное решение $(P2)$. Построим решение $S1$ $(P1)$, уменьшая длительность однопроцессорных работ в два раза и не изменяя время выполнения двухпроцессорных работ. В таком случае потребление энергии будет не больше, чем $E/2 = E'$, и все работы будут выполнены в заданных объемах. Более того, $S1$ строится таким образом, что время завершения $C_j(S1)$ любой работы j в $S1$ не больше, чем $C_j(S2)$ в $S2$. Следовательно, $\sum C_j^*(P1) \leq \sum_{j\in\mathcal{J}} C_j(S1) \leq \sum_{j\in\mathcal{J}} C_j(S2)$ для любого расписания $S2$, в частности, для оптимального решения $(P2)$. В итоге $\sum C_j^*(P1) \leq \sum C_j^*(P2)$. □

Приближенный алгоритм $\mathcal{A}$.

Шаг 1: По индивидуальной задаче $(P2)$ строится пример $(P1)$, как описано выше. Работы перенумеровываются по неубыванию объемов W'_j и вычисляются оптимальные длительности p'_j.

Шаг 2: Определяются длительности работ в задаче $(P2)$: $p_j = 2p'_j$ для однопроцессорных работ и $p_j = p'_j$ для двухпроцессорных работ. Работа j назначается первому доступному процессору, если требует для выполнения один процессор, или двум процессорам, когда оба они доступны, если работа требует два процессора, с сохранением порядка работ, который был получен на шаге 1.

Лемма 17 $\sum_{j\in\mathcal{J}} C_j(\mathcal{A}) \le 2\sum C_j^*(P2)$.

Доказательство. Пусть $C_j(P1)$ обозначает время завершения работы j для $(P1)$ на шаге 1, а $C_j(P2)$ – время завершения работы j для $(P2)$ на шаге 2. По лемме 16 имеем $\sum_{j=1}^{n} C_j(P1) = \sum C_j^*(P1) \le \sum C_j^*(P2)$. Поскольку на шаге 2 сохраняется тот же порядок работ, что и на шаге 1, то $C_j(P2) \le \sum_{i=1}^{j} p_i$ для каждой работы j. Отметим, что $C_j(P1) = \sum_{i=1}^{j} p_i' \ge \sum_{i=1}^{j} \frac{p_i}{2} \ge \frac{1}{2} C_j(P2)$. Таким образом, $\sum_{j=1}^{n} C_j(P2) \le 2\sum_{j=1}^{n} C_j(P1) \le 2\sum C_j^*(P2)$. □

Теорема 21 *Для задачи* $P2|size_j, energy|\sum C_j$ *можно построить* 2*-приближенное расписание за время* $O(n\log n)$.

2.2.4 Гибкие работы

В этом разделе рассмотрим случай построения расписания для гибких работ. Напомним, что V_j обозначает общий объем работы j, то есть время выполнения этой работы на одном процессоре с единичной скоростью. Пусть $\delta_j \le m$ обозначает верхнюю границу на число процессоров, которые может использовать работа j. Исследуется линейный случай, когда время выполнения работы линейно уменьшается с числом назначенных ей процессоров.

Предположим, что упорядочение работ по неубыванию V_j соответствует их упорядочению по неубыванию δ_j (т. е. из $V_i < V_j$ следует $\delta_i \le \delta_j$). Чтобы получить нижнюю оценку на сумму моментов завершения, построим следующую задачу выпуклого программирования в случае заданной последовательности работ π:

$$\frac{1}{m}\sum_{j=1}^{n}\sum_{i=1}^{j} p_{\pi_i} + \frac{1}{2}\sum_{j=1}^{n}\frac{p_{\pi_j}}{\delta_{\pi_j}} - \frac{1}{2m}\sum_{j=1}^{n} p_{\pi_j} \to \min, \tag{110}$$

$$\sum_{j=1}^{n} V_j^{\alpha} p_j^{1-\alpha} \le E, \tag{111}$$

$$p_j \ge 0,\ j \in \mathcal{J}. \tag{112}$$

Здесь p_j – длительность работы j на одном процессоре в объеме V_j. Длительность работы j на $m_j \le \delta_j$ процессорах будет равна $\frac{p_j}{m_j}$.

Используя те же свойства, что и в лемме 12 из раздела 2.2.2, можно показать, что минимум (110), обозначаемый через

$$LB(\pi) := \frac{E^{1/1-\alpha}}{m} \left(\sum_{j=1}^{n} V_{\pi_j} \left(n - j + 0.5 + \frac{0.5m}{\delta_{\pi_j}} \right)^{\alpha-1/\alpha} \right)^{\alpha/\alpha-1},$$

достигается на перестановке π^*, где работы упорядочены по неубыванию верхней границы на число используемых процессоров $\delta_j,\ j \in \mathcal{J}$, Тогда длительность работы на одном процессоре в полном объеме будет равна

$$p^*_{\pi^*_i} = \frac{E^{\frac{1}{1-\alpha}} V_{\pi^*_i}}{\left(n - i + 0.5 + \frac{0.5m}{\delta_{\pi^*_i}} \right)^{\frac{1}{\alpha}}} \cdot \left(\sum_{j=1}^{n} \frac{V^*_{\pi_j}}{\left(n - j + 0.5 + \frac{0.5m}{\delta_{\pi^*_j}} \right)^{\frac{1-\alpha}{\alpha}}} \right)^{\frac{1}{\alpha-1}}, \quad (113)$$

$$i = 1, \ldots, n.$$

Предположим, что упорядочение работ по неубыванию $\frac{V_j}{\delta_j}$ соответствует их упорядочению по неубыванию δ_j или упорядочение работ по неубыванию V_j соответствует их упорядочению по неубыванию δ_j и все $\delta_j \leq \frac{m}{2}$. Тогда при назначении количества используемых процессоров $m_j := \delta_j$ для всех работ $j \in \mathcal{J}$ и построении допустимого расписания с помощью алгоритма списочного типа при последовательности работ π^* будет получено 2-приближенное решение, как и в случае жестких работ.

Теперь покажем, что 2-приближенное решение может быть найдено для более общего случая, когда упорядочение работ по неубыванию V_j соответствует их упорядочению по неубыванию δ_j и δ_j произвольны. Будем назначать количество процессоров m_j для работ по правилу

$$m_j = \begin{cases} \delta_j & \text{if } \delta_j < \left\lceil \frac{m}{2} \right\rceil, \\ \left\lceil \frac{m}{2} \right\rceil & \text{if } \delta_j \geq \left\lceil \frac{m}{2} \right\rceil, \ j \in J. \end{cases}$$

Построим расписание с помощью алгоритма списочного типа на основе порядка работ по неубыванию $\delta_j,\ j \in \mathcal{J}$. Докажем, что суммарное время завершения $\sum C_j(p^*_j) \leq 2LB(\pi^*)$. Действительно, по крайней мере

$\frac{m}{2}$ процессоров используются в расписании в каждый момент времени, поэтому

$$\sum C_j(p_j^*) \le \frac{2}{m}\sum_{j=1}^{n}\sum_{i=1}^{j-1} p^*_{\pi_i^*} + \sum_{j=1}^{n}\frac{p^*_{\pi_j^*}}{m_j} = \frac{2}{m}\sum_{j=1}^{n}\sum_{i=1}^{j} p^*_{\pi_i^*} + \sum_{j=1}^{n}\frac{p^*_{\pi_j^*}}{m_j} - \frac{2}{m}\sum_{j=1}^{n} p^*_{\pi_j^*} =$$

$$\frac{2}{m}\sum_{j=1}^{n}\sum_{i=1}^{j} p^*_{\pi_i^*} + \sum_{j=1}^{n} p^*_{\pi_j^*}\left(\frac{1}{m_j} - \frac{2}{m}\right) \le$$

$$\frac{2}{m}\sum_{j=1}^{n}\sum_{i=1}^{j} p^*_{\pi_i^*} + \sum_{j=1}^{n} p^*_{\pi_j^*}\left(\frac{1}{\delta_j} - \frac{1}{m}\right) \le 2LB(\pi^*).$$

Следовательно, имеет место

Теорема 22 *Для задачи $P|any, \delta_j, energy|\sum C_j$ можно найти 2-приближенное расписание за полиномиальное время в случае, когда упорядочение работ по неубыванию V_j соответствует их упорядочению по неубыванию δ_j.*

Отметим, что вопрос о вычислительной сложности задачи $P|any, V_j = V, \delta_j, energy|\sum C_j$ является открытым.

2.2.5 Одномодальные многопроцессорные работы

Рассмотрим одномодальные многопроцессорные работы. Докажем, что задача в такой постановке NP-трудна, а также построим приближенный алгоритм с гарантированной константной оценкой точности для случая двух процессоров.

Теорема 23 *Задача $P|fix_j,\ |fix_j| = 2, V_j = 2, energy|\sum C_j$ является NP-трудной в сильном смысле.*

Доказательство. Доказательство строится на идее, аналогичной доказательству теоремы 18, но основано на полиномиальной сводимости NP-полной в сильном смысле задаче о реберной раскраске для кубических графов [131].

Хроматический индекс графа – это минимальное количество цветов, необходимое для раскраски ребер графа (образуют цветовые классы) таким образом, чтобы никакие два смежных ребра не были раскрашены в один цвет. Рассмотрим пример задачи о реберной раскраске в кубическом графе $G = (V, A)$, где необходимо определить равен ли хроматический индекс $\chi'(G)$ трем. Известно, что $\chi'(G) = 3$ или 4, а $|V|$ четно. Более того, $\chi'(G) = 3$ тогда и только тогда, когда каждый цветовой класс имеет ровно $\frac{1}{2}|V|$ ребер.

Построим индивидуальную задачу $P|fix_j, energy|\sum C_j$ следующим образом. Пусть число работ $n = |A|$, число процессоров $m = |V|$ и ограничение на потребление энергии $E = 2|A| = 3|V|$. Вершины соответствуют процессорам. Каждому ребру $\{u_j, v_j\}$ ставится в соответствие работа j с $fix_j = \{u_j, v_j\}$, $j = 1, \ldots, |A|$. Полагаем $V_j = 2$ и $W_j = \frac{V_j}{|fix_j|} = 1$ для всех $j = 1, \ldots, n$. В распознавательной версии $P|fix_j, W_j = 1, energy|\sum C_j$ требуется ответить на вопрос: существует ли расписание, в котором суммарное время завершения не превышает заданный порог T?

Для определения значения T решим вспомогательную задачу с $2n$ однопроцессорными работами, т. е. каждая двухпроцессорная работа заменяется двумя однопроцессорными работами. Такая задача имеет единственное оптимальное решение (с точностью до перестановки работ на процессорах), где каждый процессор выполняет три работы и потребляет 3 единицы энергии. Найдем оптимальные длительности работ на каждом процессоре, используя следующую задачу выпуклого программирования:

$$p_1 + 2p_2 + p_3 \to \min, \tag{114}$$

$$\sum_{j=1}^{3} p_j^{1-\alpha} = 3, \tag{115}$$

$$p_j \geq 0,\ j = 1, \ldots, n. \tag{116}$$

Здесь p_j – длительность j-й работы на процессоре.

Составляем функцию Лагранжа

$$L(p_j, \lambda) = (3p_1 + 2p_2 + p_3) + \lambda \left(p_1^{1-\alpha} + p_2^{1-\alpha} + p_3^{1-\alpha} - 3\right)$$

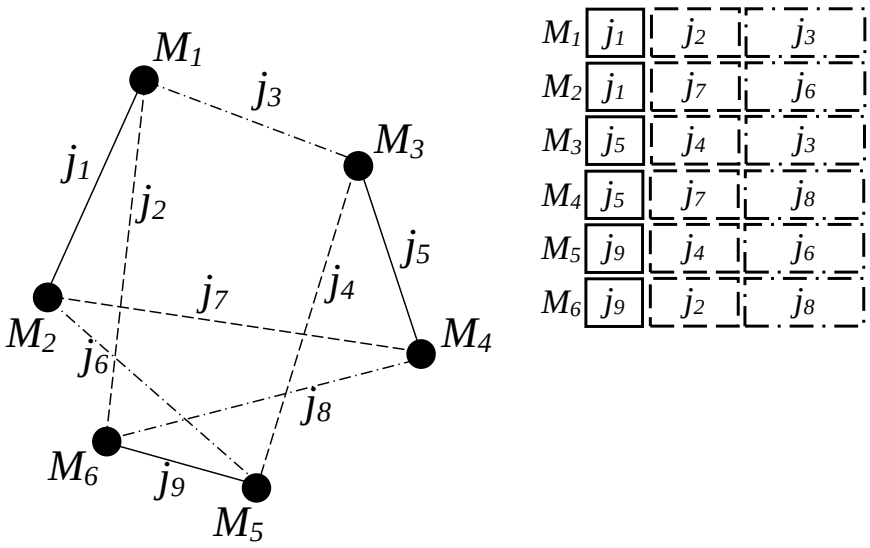


Рисунок 10 — Пример кубического графа с $\chi'(G) = 3$ (различные цветовые классы обозначены различными типами линий) и соответствующее расписание для $P|fix_j, energy|\sum C_j$.

и вычисляем

$$p_j^* = \frac{3^{1/1-\alpha}}{(4-j)^{1/\alpha}} \left(3^{\alpha-1/\alpha} + 2^{\alpha-1/\alpha} + 1\right)^{1/\alpha-1}, \; j = 1, 2, 3, \tag{117}$$

$$\sum C_j^* = \left(\frac{\left(3^{\alpha-1/\alpha} + 2^{\alpha-1/\alpha} + 1\right)^{\alpha}}{3}\right)^{1/\alpha-1}. \tag{118}$$

Здесь применяются условия Каруша-Куна-Таккера, как и в доказательстве теоремы 18. Сумма моментов завершения работ на всех процессорах равна $m\sum C_j^*$. Оптимальное расписание не содержит простоев. Установим пороговое значение $T := \frac{m}{2}\sum C_j^*$, поскольку не более $\frac{m}{2}$ двухпроцессорных работ могут выполняться одновременно. Докажем, что положительный ответ в задаче о реберной раскраске ведет к положительному ответу в построенной распознавательной версии задачи $P|fix_j, W_j = 1, energy|\sum C_j$ и наоборот.

Предположим, что ответ в задаче о реберной раскраске положительный. Тогда существует допустимое расписание, в котором $\frac{m}{2}$ работ, соответствующих $\frac{m}{2}$ рёбрам, образующим один цветовой класс, выполняются параллельно (см. пример на рис. 10). Это расписание схоже с оптималь-

ным расписанием для соответствующей задачи с однопроцессорными работами. Значение критерия равно $\frac{m}{2}\sum C_j^*$.

Покажем, что если сумма моментов завершения работ не превышает $T = \frac{m}{2}\sum C_j^*$, то соответствующее расписание позволит получить положительный ответ в задаче о реберной раскраске графа. Отметим, что каждый процессор должен выполнить 3 работы единичного объема. Таким образом, минимальная сумма моментов завершения работ на каждом процессоре равна (118) и достигается, когда бюджет по энергии $3|V|$ распределяется равномерно между процессорами. В этом случае расписание для двухпроцессорных работ идентично оптимальному расписанию для соответствующих однопроцессорных работ ввиду свойств порогового значения T. Таким образом, три подмножества работ, выполняемых одновременно, задают три цветовых класса для задачи о реберной раскраске графа.

Представленная сводимость является полиномиальной. Таким образом, задача составления расписаний $P|fix_j, W_j = 1, energy|\sum C_j$ является NP-трудной в сильном смысле. □

Построим приближенный алгоритм с гарантированной константной оценкой точности для задачи с двумя процессорами (обозначаемой через P). Пусть $\mathcal{J}_i$ задает подмножество работ, для которых требуется только процессор $i = 1, 2$, а $\mathcal{J}_{12}$ определяет подмножеством работ с $fix_j = \{1, 2\}$, т. е. $\mathcal{J} = \mathcal{J}_1 \cup \mathcal{J}_2 \cup \mathcal{J}_{12}$. Выделим две подзадачи: первая подзадача P_1 включает работы из $\mathcal{J}' = \mathcal{J}_1 \cup \mathcal{J}_{12},\ |J'| = n'$, а вторая подзадача P_2 включает только работы из $\mathcal{J}'' = \mathcal{J}_2 \cup \mathcal{J}_{12},\ |J''| = n''$.

Оптимум $\sum C_j^1$ ($\sum C_j^2$) P_1 (P_2) достигается при перестановке, где работы упорядочены по неубыванию значений $W_i|fix_i|^{1/\alpha},\ i \in \mathcal{J}'$ ($W_i|fix_i|^{1/\alpha},\ i \in \mathcal{J}''$), см. раздел 2.2.6. Значение $\max\{\sum C_j^1,\ \sum C_j^2\}$ является нижней оценкой на целевую функцию, обозначаемую через $LB(P)$, для задачи P.

Уменьшим бюджет на расход энергии в обеих подзадачах в два раза и получим для них $2^{\frac{1}{\alpha-1}}$-приближенные решения S' и S'' с теми же последовательностями работ, что и в случае бюджета E (см. формулу (123) в разделе 2.2.6). Пусть C'_j и C''_j (p'_j и p''_j) обозначают моменты завершения (длительности) работы j в S' и S'' соответственно. Отметим, что подпоследовательности двухпроцессорных работ идентичны в оптимальных

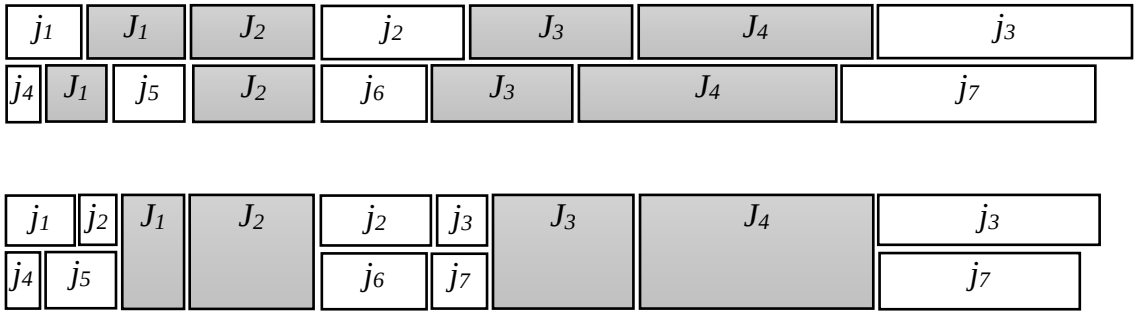


Рисунок 11 — Пример однопроцессорных расписаний и результирующего двухпроцессорного расписания с прерываниями, построенного алгоритмом $\mathcal{A}^{pr}$.

решениях обеих подзадач. Далее построим расписание с прерываниями для исходной задачи (алгоритм $\mathcal{A}^{pr}$), а затем преобразуем его в расписание без прерываний (алгоритм $\mathcal{A}^{npr}$).

Алгоритм 3 Приближенный алгоритм $\mathcal{A}^{pr}$ для задачи с прерываниями.

1: Ставим двухпроцессорные работы j в расписание без прерываний в интервалах времени $(\max\{C'_j, C''_j\} - \min\{p'_j, p''_j\},\ \max\{C'_j, C''_j\}]$.
2: Однопроцессорные работы ставим на выполнение без простоев в той же последовательности, что и в S' и S'' (длительности не изменяются). Двухпроцессорные работы могут вызывать прерывания однопроцессорных работ.

Отметим, что интервалы выполнения двухпроцессорных работ не пересекаются, поэтому построенное расписание с прерываниями является допустимым (см. пример на рис. 11). Суммарное время завершения $\sum C_j(\mathcal{A}^{pr}) \leq 2 \cdot 2^{1/\alpha - 1} LB(P)$, так как моменты завершения работ не превышают своих значений в S' и S'' по построению.

Перейдем к построению допустимого расписания без прерываний. Полученное расписание с прерываниями может быть преобразовано без увеличения времени завершения работ таким образом, чтобы каждая двухпроцессорная работа прерывала не более одной однопроцессорной работы (если двухпроцессорная работа j прерывает две однопроцессорных работы i и i', то изменение времени начала j на максимальное из времен

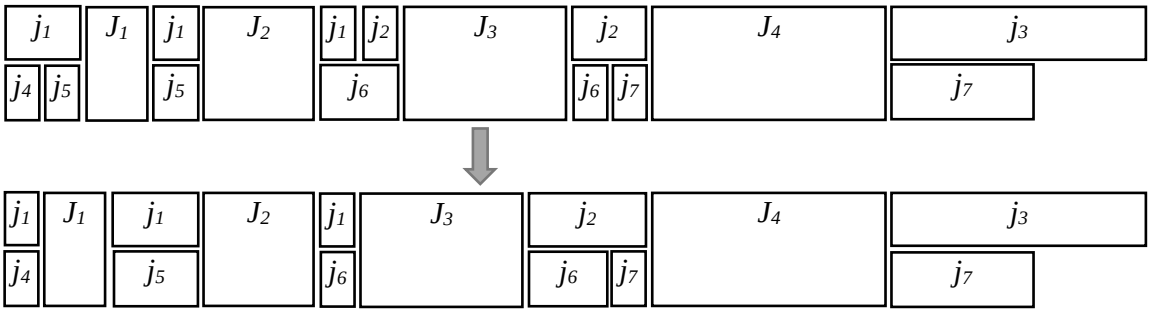


Рисунок 12 — Корректировка расписания с прерываниями.

начала i и i' и перемещение соответствующих частей i и i' на более поздний срок уменьшит время завершения j, не изменяя время завершения других работ, см. рис. 12). Пусть $S'(j)$ и $C'(j)$ обозначают время начала и время окончания, соответственно, работы j в модифицированном расписании с прерываниями.

Лемма 18 *Для задачи* $P2|fix_j, energy|\sum C_j$ *алгоритм* $\mathcal{A}^{npr}$ *строит допустимое расписание с суммой моментов завершения выполнения работ* $\sum C_j(\mathcal{A}^{npr}) \le 2\sum C_j(\mathcal{A}^{pr}) \le 2^{2\alpha-1/\alpha-1}LB(P)$.

Доказательство. На шаге l (см. рис. 13) вставка периода простоя увеличивает суммарное время завершения в расписании на $g(j_{i_l}) \cdot h(j_{i_l})$, и в общей сложности после всех шагов расписание без прерываний будет иметь значение целевой функции

$$\sum C_j(\mathcal{A}^{npr}) \le \sum C_j(\mathcal{A}^{pr}) + \sum_{l=1}^{k} g(j_{i_l}) \cdot h(j_{i_l}).$$

Поскольку каждая двухпроцессорная работа прерывает не более одной однопроцессорной работы, то временные интервалы $(S'(j_{i_1}), C'(F(j_{i_1})]$, $(S'(j_{i_2}), C'(F(j_{i_2})]$,..., $(S'(j_{i_k}), C'(F(j_{i_k})]$ не пересекаются друг с другом. Поэтому,

$$\sum C_j(\mathcal{A}^{pr}) \ge \sum_{l=1}^{k} (C'(F(j_{i_l})) - S'(j_{i_l})) \cdot h(j_{i_l}) > \sum_{l=1}^{k} g(j_{i_l}) \cdot h(j_{i_l}).$$

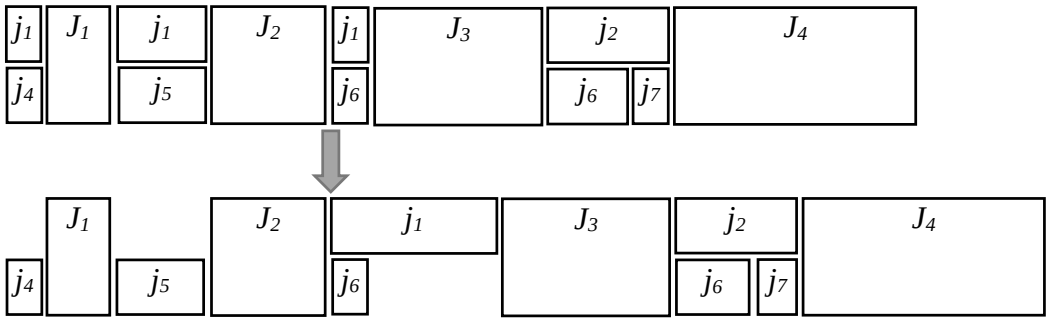


Рисунок 13 — Шаг $l = 1$ в приближенном алгоритме $\mathcal{A}^{npr}$.

В результате имеем следующую нижнюю оценку на суммарное время завершения работ:

$$\sum C_j(\mathcal{A}^{npr}) \leq \sum C_j(\mathcal{A}^{pr}) + \sum_{l=1}^{k} g(j_{i_l}) \cdot h(j_{i_l})$$

$$< 2 \sum C_j(\mathcal{A}^{pr}) \leq 2^{2\alpha - 1/\alpha - 1} LB(P).$$

□

Алгоритм 4 Приближенный алгоритм $\mathcal{A}^{npr}$ для задачи без прерываний.

1: Идентифицировать однопроцессорные работы j_{i_1}, $j_{i_2}, \ldots, j_{i_k}$, которые прерываются некоторыми двухпроцессорными работами. Предположим, что эти работы упорядочены по возрастанию моментов начала $S'(j_{i_1}) < \cdots < S'(j_{i_k})$, и, следовательно, моменты завершения удовлетворяют свойству $C'(j_{i_1}) < \cdots < C'(j_{i_k})$.

2: Основная процедура построения расписания состоит из k шагов.
На шаге l перемещается работа j_{i_l}. Пусть $F(j_{i_l})$ обозначает последнюю двухпроцессорную работу, которая прерывает j_{i_l}, а $g(j_{i_l})$ есть длительность j_{i_l} до времени начала $F(j_{i_l})$, $h(j_{i_l})$ обозначает– число работ, которые завершаются после $F(j_{i_l})$, $l = 1, \ldots, k$. Заменим время начала j_{i_l} на время завершения $F(j_{i_l})$ и увеличим время начала последующих работ на $g(j_{i_l})$.

Теорема 24 *Для задачи теории расписаний* $P2|fix_j, energy|\sum C_j$ *мож-*

но построить $2^{2\alpha-1/\alpha-1}$*-приближенное расписание за полиномиальное время.*

Теорема 25 *Для задачи теории расписаний* $P2|fix_j, pmtn, energy|\sum C_j$ *можно построить* $2^{\alpha/\alpha-1}$*-приближенное расписание за полиномиальное время.*

Статус сложности задачи с прерываниями и без прерываний в случае двух процессоров является открытым вопросом.

2.2.6 Полиномиально разрешимые случаи

Построим полиномиальный точный алгоритм для простых частных случаев, когда существует оптимальное решение, в котором в каждый момент времени выполняется не более одной работы.

Рассмотрим следующую задачу выпуклого программирования, соответствующую случаю, когда все работы выполняются последовательно в соответствии с перестановкой π:

$$F(\pi) = \sum_{i=1}^{n}(n-i+1)p_{\pi_i} \to \min, \tag{119}$$

$$\sum_{i=1}^{n}(B_{\pi_i})^{\alpha}p_{\pi_i}^{1-\alpha} = E, \tag{120}$$

$$p_j \geq 0,\ j \in \mathcal{J}. \tag{121}$$

Здесь B_{π_i} есть некоторый параметр работы π_i, включающий, например, объем и количество используемых процессоров.

Для нахождения оптимального решения задачи (119)-(121) используем метод Лагранжа, где функция Лагранжа имеет вид

$$L(p_j, \lambda) = \sum_{i=1}^{n}(n-i+1)p_{\pi_i} + \lambda\left(\sum_{i=1}^{n}(B_{\pi_i})^{\alpha}p_{\pi_i}^{1-\alpha} - E\right).$$

Вычисляем частные производные, приравниваем их к нулю и находим длительности работ

$$p_{\pi_i} = \frac{B_{\pi_i}}{(n-i+1)^{\frac{1}{\alpha}}} \left(\sum_{j=1}^{n} B_{\pi_j} (n-j+1)^{\frac{\alpha-1}{\alpha}} \right)^{\frac{1}{\alpha-1}} (E)^{\frac{1}{1-\alpha}}, \qquad (122)$$

$$i = 1, \ldots, n.$$

Тогда оптимальное значение целевой функции для последовательности π равно

$$F^*(\pi) = \left(\sum_{i=1}^{n} B_{\pi_i} (n-i+1)^{\frac{\alpha-1}{\alpha}} \right)^{\frac{\alpha}{\alpha-1}} (E)^{\frac{1}{1-\alpha}}. \qquad (123)$$

Минимизация (123) эквивалентна минимизации

$$f(\pi) = \sum_{i=1}^{n} B_{\pi_i} (n-i+1)^{\frac{\alpha-1}{\alpha}}$$

по всем возможным перестановкам π. Определим n-мерные векторы $B_\pi = (B_{\pi_1}, B_{\pi_2}, \ldots, B_{\pi_n})$ и $N = \left(n^{\frac{\alpha-1}{\alpha}}, (n-1)^{\frac{\alpha-1}{\alpha}}, \ldots, 1^{\frac{\alpha-1}{\alpha}} \right)$.

Тогда $f(\pi)$ можно записать в виде скалярного произведения $B_\pi \bullet (N)^{\mathrm{T}}$. Утверждается, что минимум $B_\pi \bullet (N)^{\mathrm{T}}$ достигается на перестановке, где работы упорядочены по неубыванию значений $B_j,\ j \in \mathcal{J}$. Действительно, в этом случае первый вектор упорядочен по неубыванию, второй – по убыванию, и скалярное произведение минимально.

Однопроцессорный случай. Задача составления расписаний на одном процессоре может быть сформулирована как следующая задача выпуклого программирования в случае, когда задана последовательность работ π:

$$\sum_{j \in \mathcal{J}} C_j(\pi) = \sum_{i=1}^{n} (n-i+1) p_{\pi_i} \to \min,$$

$$\sum_{i=1}^{n} (W_{\pi_i})^{\alpha} p_{\pi_i}^{1-\alpha} = E,$$

$$p_j \geq 0,\ j \in \mathcal{J}.$$

Минимальная сумма моментов завершения достигается на перестановке, где работы упорядочены по неубыванию объемов $W_j,\ j \in \mathcal{J}$.

Теорема 26 *Оптимальное расписание для задачи* $1|energy|\sum C_j$ *можно найти за полиномиальное время.*

Полностью распараллеливаемые работы. Покажем полиномиальную разрешимость задачи с мягкими работами в случае, когда все работы могут использовать до m процессоров (полностью распараллеливаемые работы). Рассмотрим линейный случай, когда время выполнения работы линейно уменьшается с числом назначенных ей процессоров. Очевидно, что существует оптимальное расписание, при котором каждая работа задействует m процессоров (см., например, [230]). Такая задача идентична задаче минимизации суммы моментов завершения работ на одном процессоре. Пусть работы упорядочены в соответствии с перестановкой π и $W_j = \frac{V_j}{m}$, тогда имеем следующую задачу выпуклого программирования

$$\sum_{i=1}^{n}\sum_{j=1}^{i} p_{\pi_j} = \sum_{i=1}^{n}(n-i+1)p_{\pi_i} \to \min,$$

$$\sum_{i=1}^{n} W_i^{\alpha} p_i^{1-\alpha} \le \frac{E}{m},$$

$$p_j \ge 0,\ j \in \mathcal{J}.$$

Минимум суммы моментов завершения достигается на перестановке, где работы упорядочены по неубыванию объемов $W_i,\ i \in \mathcal{J}$.

Теорема 27 *Оптимальное решение для задачи составления расписаний* $P|var, \delta_j = m, energy|\sum C_j$ *можно найти за полиномиальное время.*

Несовместные работы. Работы называются несовместимыми, когда никакие две работы не могут выполняться одновременно. В случае одномодальных работ это означает, что $fix_j \cap fix_i \neq \emptyset$ для всех $i \neq j \in \mathcal{J}$. Жесткие работы несовместимы, когда все $size_j > \frac{m}{2},\ j \in \mathcal{J}$. Такая

задача также аналогична задаче с одним процессором, поскольку в каждый момент времени может выполняться не более одной работы. Предположим, что работы упорядочены согласно некоторой перестановке π. Чтобы найти оптимальные длительности работ и целевую функцию для перестановки π, решим следующую задачу:

$$\sum_{j=1}^{n} p_{\pi_j}(n-j+1) \to \min,$$

$$\sum_{j=1}^{n} m_j p_j \left(\frac{W_j}{p_j}\right)^{\alpha} = E,$$

$$p_j \geq 0,\ j = 1, \ldots, n.$$

Здесь $m_j = |fix_j|$ для одномодальных многопроцессорных работ и $m_j = size_j$ для жестких работ. Оптимальное расписание может быть построено из оптимального решения представленной модели, соответствующей перестановке, где работы упорядочены по неубыванию $W_j m_j^{1/\alpha}$.

Теорема 28 *Задача $P|fix_j, energy|\sum C_j$ полиномиально разрешима в случае, когда $fix_j \cap fix_i \neq \emptyset$ для всех $i \neq j,\ i, j \in \mathcal{J}$.*

Теорема 29 *Задача $P|size_j, energy|\sum C_j$ полиномиально разрешима в случае, когда $size_j > \frac{m}{2}$ для всех $i \neq j,\ i, j \in \mathcal{J}$.*

2.3 Распараллеливаемые работы

Исследуется задача составления расписаний множества работ $\mathcal{J} = \{1, \ldots, n\}$ на параллельных m процессорах с возможностью варьирования скоростей. Каждая работа $j \in \mathcal{J}$ характеризуется объемом W_j и верхней границей $\delta_j \leq m$ на степень распараллеливания (максимальное число используемых процессоров). Напомним, что такие работы называются гибкими (moldable), если набор процессоров не меняется при выполнении работы, и мягкими (malleable), если набор процессоров может меняться при выполнении работы [90]. Рассматривается линейный случай, когда время выполнения работы линейно уменьшается с числом назначенных ей процессоров, а W_j — это время выполнения работы j на одном процессоре с единичной скоростью. Предполагается, что все

работы поступают в момент времени 0. Прерывание работ, миграция и отношения предшествования могут быть разрешены или запрещены в исследуемых частных случаях.

Рассматривается стандартная однородная модель учета потребления энергии при настройке скоростей процессоров. Когда процессор работает со скоростью s, потребление энергии в единицу времени (*мгновенная мощность*), составляет s^α, где $\alpha > 1$ — константа. Каждый из m процессоров может менять скорость при выполнении работ. Предполагается, что общий объем W_j работы $j \in \mathcal{J}$ должен быть равномерно распределен между используемыми процессорами, т. е. если работа j использует m_j процессоров, то объемы равны для всех m_j процессоров, и все эти процессоры работают с одной скоростью. Множество скоростей процессоров непрерывно.

Требуется построить расписание с указанием моментов начала и окончания работ, а также используемых ими процессоров, при котором минимизируется длина расписания (makespan, $C_{\max}$) с учетом общего ограничения на потребление энергии равного E. Задача составления расписаний гибких и мягких работ с учетом ограничения на потребление энергии обозначается через $P|any, \delta_j, energy|C_{\max}$ и $P|var, \delta_j, energy|C_{\max}$, соответственно.

Разрабатываются двухэтапные алгоритмы для следующих случаев:

- мягкие работы,
- гибкие работы,
- гибкие работы с частичным порядком.

На первом этапе вычисляется нижняя оценка на длину расписания и оцениваются длительности работ. Затем, на втором этапе, работам назначается количество используемых процессоров, и строится допустимое расписание алгоритмом списочного типа. Строятся гарантированные оценки точности для разработанных алгоритмов. В таблице 1 представлены полученные результаты и открытые вопросы (отмечены “?”) для различных типов работ.

Таблица 1 — Оценки точности алгоритмов при различных типах работ и критерии минимизации длины расписания

Тип работ	Прерывания и миграция	Без прерываний	Част. порядок
Жесткие	OPT+ε (§ 2.1.4)	$2 - \frac{1}{m}$ (§ 2.1.2)	$\frac{2-q}{1-q}$ (§ 2.1.2)
Гибкие	?	$2 - \frac{2}{m+1}$ (§ 2.3.3)	$\frac{3+\sqrt{5}}{2}$ (§ 2.3.4)
Мягкие	полином (§ 2.3.2)	–	?

2.3.1 Предыдущие исследования

Представим уже известные результаты для классической задачи минимизации длины расписания в случае гибких и мягких работ с заданными длительностями и без ограничений на ресурсы. Задача без прерываний с гибкими работами является NP-трудной. Для случаев $P2|any|C_{\max}$ и $P3|any|C_{\max}$ были предложены псевдополиномиальные алгоритмы, основанные на динамическом программировании [91]. Известна PTAS для случая $Pm|any|C_{\max}$ [135]. Приближенные алгоритмы списочного типа были предложены для задачи составления расписаний $P|any, \delta_j, prec|C_{\max}$ с гибкими работами, линейным ускорением от числа используемых процессоров, верхней границей на степень распараллеливания δ_j и отношениями предшествования [236].

Приближенные алгоритмы с константной оценкой точности были также получены для более общего случая с неубывающими сублинейными функциями ускорения работ от количества используемых процессоров (NdSub). Алгоритм с оценкой точности $\frac{2}{1-\frac{1}{m}}$ для $P|any, NdSub, \delta_j|C_{\max}$ решает задачу так, что сначала все работы рассматриваются как однопроцессорные и строится расписание по правилу «Работы с наибольшей длительностью выполнять раньше» (Largest Processing Time First). Затем алгоритм итеративно увеличивает количество используемых процессоров для работ «большой» длительности [42]. Аналогичные подходы использовались в работах [117, 161]. Для задачи с отношениями предшествования известен двухэтапный приближенный алгоритм [171]. На первом этапе назначение процессоров работам реализуется с помощью 2-приближенного алгоритма, учитывающего самый длинный путь в графе частичного порядка и общую загрузку процессоров. На втором этапе строится допустимое расписание работ с определенным на первом этапе

числом процессоров для каждой работы. Строится расписание алгоритмом списочного типа с учетом частичного порядка.

Задача с мягкими независимыми работами и линейным ускорением от числа процессоров разрешима за линейное время [88]. Задача с отношениями предшествования в виде набора цепей полиномиально разрешима, когда количество процессоров ограничено константой, и NP-трудна, когда количество процессоров является частью входа [88].

Основные результаты раздела. Методы решения задач настоящего раздела основаны на формулировках упрощенных версий задач в виде моделей выпуклого программирования, которые решаются методом эллипсоидов и с помощью условий Каруша-Куна-Таккера. Это позволяет получить нижние оценки на целевую функцию и вспомогательные оценочные длительности работ. Затем строятся допустимые расписания с помощью различных жадных стратегий.

2.3.2 Мягкие работы

Рассмотрим случай мягких работ, которые могут выполняться более чем одним процессором параллельно, тем самым уменьшая общее время выполнения. Допускаются прерывания, миграция и изменение количества используемых процессоров. Предполагается, что длительность работы j на k процессорах составляет $\frac{p_j}{k}$, где p_j – длительность работы на одном процессоре.

Рассматриваемая задача с мягкими работами при учете расхода энергии полиномиально разрешима. Действительно, нижнюю оценку на целевую функцию можно вычислить с помощью решения следующей задачи выпуклого программирования:

$$LB \to \min, \tag{124}$$

$$\frac{1}{m}\sum_{j=1}^{n} p_j \le LB, \tag{125}$$

$$\frac{p_j}{\delta_j} \le LB,\ j \in \mathcal{J}, \tag{126}$$

$$\sum_{j \in \mathcal{J}} p_j \left(\frac{W_j}{p_j} \right)^{\alpha} \leq E, \tag{127}$$

$$p_j \geq 0, \ j \in \mathcal{J}. \tag{128}$$

Здесь p_j обозначает длительность работы j на одном процессоре. Неравенство (125) показывает, что общая загрузка всех процессоров не превышает $LB \cdot m$. Ограничения (126) гарантируют, что длительность любой работы j не превышает LB при использовании максимально возможного числа δ_j процессоров. Неравенство (127) гарантирует, что общее потребление энергии не превышает верхнюю границу E. Задача (124)-(128) может быть решена за полиномиальное время с помощью условий Каруша-Куна-Таккера [166].

Далее строится оптимальное расписание, используя полученные длительности работ и алгоритм из [184]: Работы назначаются последовательно, одна за другой, на процессоры, начиная с первого. Если интервал обработки некоторой работы на некотором процессоре i превысит LB, то работа разбивается на части:

- первая часть, заканчивающаяся в момент LB на процессоре i;
- несколько частей, которые выполняются в течение интервала $[0, LB)$ на процессорах $i+1, \ldots, i+l-1$;
- последняя часть, начинающаяся в момент времени 0 на процессоре $i+l$.

Эта процедура повторяется для всех работ. Временная сложность составляет $O(n)$.

Теорема 30 *Для задачи составления расписаний* $P|var, \delta_j, energy|C_{\max}$ *оптимальное расписание можно найти за полиномиальное время.*

Отметим, что оптимальное расписание для мягких работ с $\delta_j = m, \ j \in \mathcal{J}$, можно легко найти за линейное время, назначив m процессоров для каждой работы и произвольно упорядочив их. В этом случае длительности работ равны $p_j = \frac{W_j}{m} \left(\frac{E}{\sum_{i \in \mathcal{J}} W_i} \right)^{1/(1-\alpha)}$ и все процессоры работают с одинаковой постоянной скоростью.

2.3.3 Гибкие работы

В случае гибких работ количество требуемых процессоров выбирается перед запуском работы и не изменяется до ее завершения. Предположим, что гибкая работа j может быть выполнена на любом числе процессоров вплоть до $\delta_j \leq m$. Прерывания и миграция не допускаются. Задача является NP-трудной даже в случае $\delta_j = 1$ для всех $j \in \mathcal{J}$ и $m = 2$ (к ней сводится известная задача Разбиение). Построим полиномиальный приближенный алгоритм.

Нижнюю оценку LB на длину расписания и оценку длительностей работ p_j можно найти за полиномиальное время с помощью решения задачи (124)–(128) для мягких работ. Затем, если назначить количество используемых процессоров $m_j := \delta_j$ для всех работ $j \in \mathcal{J}$, и построить допустимое расписание с помощью алгоритма списочного типа, то будет получено $\left(2 - \frac{1}{m}\right)$-приближенное решение задачи, как и в случае жестких работ.

Построим алгоритм с гарантированной оценкой точности $\left(2 - \frac{2}{m+1}\right)$ (см. алгоритм 5). Например, имеет место $\frac{3}{2}$-приближенный алгоритм для жестких работ и $\frac{4}{3}$-приближенный алгоритм для гибких работ в случае двух процессоров.

В алгоритме 5 все работы изначально рассматриваются как однопроцессорные, и допустимое расписание строится с использованием стратегии, когда работы добавляются в расписание в порядке неубывания длительностей. Затем на каждой итерации идентифицируем работу, которая определяет длину расписания, и увеличиваем количество используемых процессоров для нее, пока не будет получено приближенное решение с указанной оценкой точности. Трудоемкость каждой итерации составляет $O(n\log m)$, поскольку подмножество последовательных процессоров должно быть назначено для распараллеливаемых работ и один процессор – для каждой однопроцессорной работы. Общая временная сложность алгоритма 5 равна $O(n^2\log m)$.

Напомним, что $LB \geq \max\{\mathcal{A}; \max\limits_{j\in\mathcal{J}} \frac{p_j}{\delta_j}\}$, где $\mathcal{A} := \frac{1}{m}\sum\limits_{j\in\mathcal{J}} p_j$. Покажем, что представленный алгоритм имеет оценку точности $\left(2 - \frac{2}{m+1}\right)$. Сначала приведем свойства построенного расписания, а затем докажем основ-

ной результат для гибких работ.

Алгоритм 5 Алгоритм списочного типа для гибких работ

1: Отсортировать работы в порядке невозрастания длительностей p_j. Работы назначаются процессорам таким образом, чтобы следующая работа размещалась на процессоре с текущим минимальным временем завершения. Установить количество используемых процессоров $m_j = 1$ для всех $j \in \mathcal{J}$. Обозначить полученное расписание через S, а его длину через $C_{\max}(S)$.

2: Найти работу $i = \operatorname{argmax}_{j\in\mathcal{J}} \frac{p_j}{m_j}$ и положить $h = \max_{j\in\mathcal{J}} \frac{p_j}{m_j}$.

3: Если либо $C_{\max}(S) \neq h$, либо $m_i > 1$, либо $\delta_i = 1$, то перейти к шагу 5. Если $C_{\max}(S) = h$, но $h \leq \left(2 - \frac{2}{m+1}\right) \cdot LB$, то также переходим к шагу 5.

4: Полагаем $m_i := \left\lceil \frac{p_i}{\left(2-\frac{2}{m+1}\right)\cdot LB} \right\rceil$. Строим расписание S по правилу. Работы $j \in \mathcal{J}$ с $m_j > 1$ размещаются на требуемом количестве процессоров, начиная с момента 0 (общее количество используемых процессоров не превысит m). Для остальных работ строим расписание так же, как на шаге 1. Если $C_{\max}(S') > h$, то полагаем $m_i := 1$ и переходим к шагу 5, иначе полагаем $S := S'$ и переходим к шагу 2.

5: В качестве результата возвращаем расписание S.

Лемма 19 *Пусть $m \geq 2$ и S — расписание, полученное на некотором шаге алгоритма 5. Тогда $\frac{p_i}{m_i} > \left(1 - \frac{1}{m+1}\right) \cdot LB$ для всех $i \in \mathcal{J}$ таких, что $m_i > 1$.*

Доказательство. Предположим, что $\frac{p_i}{m_i} \leq \left(1 - \frac{1}{m+1}\right) \cdot LB$ для некоторой работы i, использующей более одного процессора, т. е. $p_i \leq \left(1 - \frac{1}{m+1}\right) \cdot LB \cdot m_i$. Тогда имеем

$$\frac{p_i}{m_i - 1} \leq \frac{\left(1 - \frac{1}{m+1}\right) \cdot LB \cdot m_i}{m_i - 1} \leq 2\left(1 - \frac{1}{m+1}\right) \cdot LB = \left(2 - \frac{2}{m+1}\right) \cdot LB.$$

Это противоречит выбору m_i на шаге 4. □

Лемма 20 *На шаге 4 алгоритма 5 имеем $\sum_{j\in\mathcal{J}:\ m_j>1} m_j \leq m$ после назначения $m_i := \left\lceil \frac{p_i}{\left(2-\frac{2}{m+1}\right)\cdot LB} \right\rceil$.*

Доказательство. Предположим, что $\sum\limits_{j:\ m_j>1} m_j \geq m+1$, тогда из леммы 19 общая загрузка процессоров работами с $m_j > 1$ будет больше, чем

$$(m+1)\left(1-\frac{1}{m+1}\right)\cdot LB = m\cdot LB \geq \sum_{j\in\mathcal{J}} p_j.$$

Это приводит к противоречию.
□

Лемма 21 *Пусть S есть расписание, полученное на шаге 1 или 4 алгоритма 5, а y – время выполнения первой работы, которая начинается в момент времени $t > 0$ (если такой работы нет, то $y = 0$). Тогда $C_{\max}(S) \leq \max\{h(S); \mathcal{A} + y\left(1-\frac{1}{m}\right)\}$, где $h(S) = \max_{j\in\mathcal{J}} \frac{p_j}{m_j}$ – длительность самой большой работы в расписании S.*

Доказательство. Если $C_{\max}(S)$ соответствует времени завершения работы, которая начинается в момент времени 0, то $C_{\max}(S) = h(S)$. В противном случае $C_{\max}(S)$ равно времени завершения работы i, которая началась позже (см. рис. 14). Легко видеть, что $m_i = 1$ и $p_i \leq y$. Работа i начинается в момент времени $t' = (C_{\max}(S) - p_i)$. По построению S в алгоритме 5, ни один из процессоров не завершает выполнение до момента времени t'. Положим $\mathcal{A}' := \frac{1}{m}\sum_{j\in\mathcal{J}'} p_j$, где $\mathcal{J}'$ – подмножество работ, которые начинаются до момента t'. Тогда $t' \leq \mathcal{A}' \leq \mathcal{A} - \frac{p_i}{m}$. В результате имеем

$$C_{\max}(S) - p_i \leq \mathcal{A} - \frac{p_i}{m}$$

и

$$C_{\max}(S) \leq \mathcal{A} + p_i\left(1-\frac{1}{m}\right) \leq \mathcal{A} + y\left(1-\frac{1}{m}\right).$$

□

Лемма 22 *Пусть S есть расписание, полученное на шаге 1 или 4 алгоритма 5, а y – это время выполнения первой работы, которая начинается в момент времени $t > 0$ (если такой работы нет, то $y = 0$). Тогда*

1. *$y \leq \frac{p_i}{m_i}$ для всех $i \in \mathcal{J}$ таких, что $m_i > 1$;*

2. *$C_{\max}(S) \leq \max\{h(S); \frac{2\mathcal{A}}{\left(1+\frac{1}{m}\right)}\}$.*

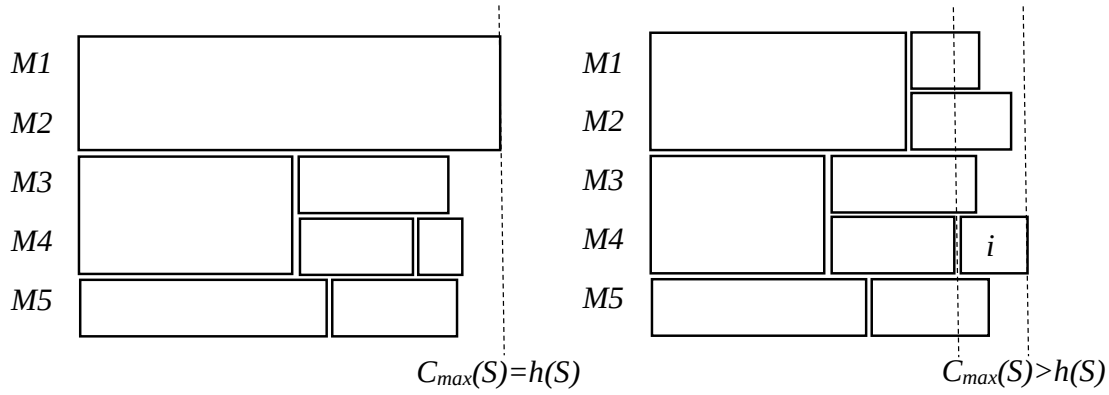


Рисунок 14 — Типы расписаний, построенных алгоритмом 5.

Доказательство. **1.** Из леммы 19 имеем $\frac{p_i}{m_i} > \left(1 - \frac{1}{m+1}\right) \cdot LB$ для всех работ $i \in \mathcal{J}$, использующих более одного процессора. Предположим, что $y \geq \left(1 - \frac{1}{m+1}\right) \cdot LB$, тогда все однопроцессорные работы, начинающиеся в момент времени 0, имеют длительность $p_j \geq \left(1 - \frac{1}{m+1}\right) \cdot LB$. Итак, общая загрузка m процессоров работами, начинающимися в момент времени 0, и работой, соответствующей y, будет больше, чем

$$(m+1)\left(1 - \frac{1}{m+1}\right) \cdot LB = m \cdot LB \geq \sum_{j \in \mathcal{J}} p_j.$$

Это приводит к противоречию. Поэтому $y < \left(1 - \frac{1}{m+1}\right) \cdot LB$ и $y < \frac{p_i}{m_i}$ для всех $i \in \mathcal{J}$ таких, что $m_i > 1$.

2. Длительность работы, соответствующей y, не больше длительности работ, начинающихся в момент времени 0. Итак, имеем $\mathcal{A} \geq m\frac{y}{m} + \frac{y}{m}$. Другими словами, $y \leq \frac{\mathcal{A}}{\left(1+\frac{1}{m}\right)}$. Подставляя это выражение в результат леммы 21, получаем $C_{\max}(S) \leq \max\left\{h(S); \frac{2\mathcal{A}}{\left(1+\frac{1}{m}\right)}\right\}$. □

Теорема 31 *Пусть S есть расписание, полученное на шаге 5 алгоритма 5. Тогда $C_{\max}(S) \leq \left(2 - \frac{2}{m+1}\right) \cdot LB$.*

Доказательство. Возможны два случая (см. рис. 14).

Случай 1. $C_{\max}(S) \neq h(S) := \max_{j \in \mathcal{J}} \frac{p_j}{m_j}$. Легко проверить, что $C_{\max}(S) \geq \max\{h(S); \mathcal{A}\}$. Тогда $C_{\max}(S) > h(S)$, и из леммы 22 имеем $C_{\max}(S) \leq \left(2 - \frac{2}{m+1}\right)\mathcal{A} \leq \left(2 - \frac{2}{m+1}\right) \cdot LB$.

Случай 2. $C_{\max}(S) = h(S)$. Рассмотрим различные возможные случаи остановки алгоритма 5.

Алгоритм 5 завершился на шаге 3, так как
1) $m_i > 1$, поэтому, в силу правила выбора значения m_i имеем $h(S) = \frac{p_i}{m_i} \leq \left(2 - \frac{2}{m+1}\right) \cdot LB$;
2) $\delta_i = 1$, следовательно, $p_i = \frac{p_i}{\delta_i} \leq LB$;
3) $h(S) \leq \left(2 - \frac{2}{m+1}\right) \cdot LB$, таким образом, $C_{\max}(S) \leq \left(2 - \frac{2}{m+1}\right) \cdot LB$.

Алгоритм 5 завершился на шаге 4, так как построенное расписание S' имеет $C_{\max}(S') > h(S)$. Аналогично доказательству леммы 22 можно показать

$$C_{\max}(S') \leq \max\left\{h(S'); \frac{2\mathcal{A}}{\left(1 + \frac{1}{m}\right)}\right\}.$$

Из предположения $C_{\max}(S') > h(S) \geq h(S')$, поэтому $C_{\max}(S') \leq \frac{2\mathcal{A}}{\left(1+\frac{1}{m}\right)}$. Однако, $C_{\max}(S) = h(S) < C_{\max}(S') \leq \frac{2\mathcal{A}}{\left(1+\frac{1}{m}\right)} \leq \left(2 - \frac{2}{m+1}\right) \cdot LB$. $\square$

В результате имеем

Теорема 32 *Для задачи составления расписаний $P|any, \delta_j, energy|C_{\max}$ можно построить $\left(2 - \frac{2}{m+1}\right)$-приближенное расписание за полиномиальное время.*

2.3.4 Гибкие работы с частичным порядком

Рассмотрим случай гибких работ без прерываний при наличии частичного порядка между работами. Если работа j предшествует работе j' (обозначается $j \prec j'$), то j' не может начаться, пока j не будет завершена. Отношения предшествования представлены ориентированным ациклическим графом $G = (J, A)$, где дуга $(j, j') \in A$ тогда и только тогда, когда работа j предшествует работе j'.

Сформулируем следующую задачу выпуклого программирования для построения нижней оценки на оптимальную длину расписания:

$$T \to \min, \tag{129}$$

$$\frac{1}{m}\sum_{j \in \mathcal{J}} p_j \leq T, \tag{130}$$

$$C_j \leq T,\ j \in \mathcal{J}, \tag{131}$$

$$\frac{p_j}{\delta_j} \le C_j,\ j \in \mathcal{J}, \tag{132}$$

$$C_j + \frac{p_{j'}}{\delta_{j'}} \le C_{j'},\ (j, j') \in A, \tag{133}$$

$$\sum_{j \in \mathcal{J}} W_j^{\alpha} p_j^{1-\alpha} \le E, \tag{134}$$

$$C_j \ge 0,\ p_j \ge 0,\ j \in \mathcal{J}. \tag{135}$$

Здесь переменная p_j (C_j) соотвествует длительности работы $j \in \mathcal{J}$ на одном процессоре (оценка времени завершения j). Неравенства (130), (131) и (132), (134) имеют тот же смысл, что и неравенства (125), (126), (127) в модели для задачи без ограничений предшествования (см. раздел 2.3.2). Неравенства (133) позволяют оценить моменты завершения работ с учетом отношений предшествования. Задача выпуклого программирования (129)-(135) позволит получить полиномиальный алгоритм для вычисления нижней оценки, поскольку задачи выпуклого программирования могут быть решены с произвольной точностью с помощью мотода эллипсоидов (см., например, [193]).

После решения задачи (129)-(135) можно определить критический путь в графе $G^{\delta} = (J, A, P^{\delta})$ с длинами вершин-работ, равными $\frac{p_j}{\delta_j}$, т. е. $P^{\delta} = \left(\frac{p_1}{\delta_1}, \ldots, \frac{p_n}{\delta_n}\right)$. Обозначим длину этого пути как L^{δ}, а среднюю длительность работ как $\mathcal{A} := \frac{1}{m} \sum_{j \in \mathcal{J}} p_j$. Тогда оптимальная длина расписания $C^*_{\max} \ge \max\{L^{\delta};\ \mathcal{A}\}$.

Пусть $\mu \le \frac{m+1}{2}$ обозначает максимальное количество процессоров, которое может быть использовано любой работой в расписании, построенном с помощью алгоритма 6. Здесь для каждой работы количество процессоров назначается на первом шаге, а сами работы ставятся в расписание с учетом отношений предшествования за время $O(n^3)$ на втором шаге.

Покажем, что алгоритм 6 имеет оценку точности $\max\left\{\frac{m}{\mu}; \frac{2m-\mu}{m-\mu+1}\right\}$. Установим свойства построенного расписания, а затем получим константную оценку точности. Пусть $C^{apx}_{\max}$ обозначает длину расписания, построенного алгоритмом 6. Тогда, $C^{apx}_{\max} \ge \max\{L^{apx};\ \mathcal{A}\}$, где L^{apx} – длина критического пути в графе $G^{apx} = (J, A, P^{apx})$ с вектором длин работ, равным $P^{apx} = \left(\frac{p_1}{m_1}, \ldots, \frac{p_n}{m_n}\right)$.

Алгоритм 6 Алгоритм списочного типа для гибких работ с отношением предшествования

1: Назначить работе $j \in \mathcal{J}$ количество процессоров равным $m_j := \min\{\delta_j, \mu\}$. Это определяет длительность $\bar{p}_j = \frac{p_j}{m_j}$ работы $j \in \mathcal{J}$ на каждом из используемых процессоров.

2: Повторять следующую процедуру, пока все работы не будут поставлены в расписание:

Пусть $\bar{\mathcal{J}}$ обозначает подмножество еще не размещенных в расписании работ, все предшественники которых уже стоят в расписании. Для каждой работы из $\bar{\mathcal{J}}$ вычислим самый ранний возможный момент начала. Добавим работу с самым ранним моментом начала в расписание.

Построенное расписание содержит три типа интервалов $I_1,\ I_2,\ I_3$ (см. рис. 15), таких, что

- I_1 – множество интервалов, где используется не более $\mu - 1$ процессоров;
- I_2 – множество интервалов, где используется не менее μ, но не более $m - \mu$ процессоров;
- I_3 – множество интервалов, где используется не менее $m - \mu + 1$ процессоров.

Следовательно,

$$C_{\max}^{apx} = |I_1| + |I_2| + |I_3|, \tag{136}$$

$$m\mathcal{A} = \sum_{j \in \mathcal{J}} p_j \geq |I_1| + \mu|I_2| + (m - \mu + 1)|I_3|. \tag{137}$$

Если умножить (136) на $(m - \mu + 1)$ и вычесть (137), то получим

$$(m - \mu + 1)C_{\max}^{apx} \leq m\mathcal{A} + (m - \mu)|I_1| + (m - 2\mu + 1)|I_2|. \tag{138}$$

Установим связь между длиной L^δ критического пути в G^δ и длительностями интервалов из $I_1,\ I_2$.

Лемма 23 *Неравенство $|I_1| + \frac{\mu}{m}|I_2| \leq L^\delta$ выполняется для расписания, построенного алгоритмом 6.*

Доказательство. Находим критическую цепочку работ в расписании, построенном с помощью алгоритма 6. Пусть j_1 есть любая работа, которая завершается в момент времени $C^{apx}_{\max}$. После идентификаци работ $j_i \to j_{i-1} \to \cdots \to j_1$, следующая работа j_{i+1} выбирается по правилу: Рассмотрим последний временной интервал I в $I_1 \cup I_2$, который идет до момента начала j_i. Определим множество $\mathcal{J}'$ работ, которое включает j_i и все работы, которые ей предшествуют и начинаются после интервала I. Так как в течение интервала I задействовано не более $(m-\mu)$ процессоров, и так как алгоритм 6 назначает не более μ процессоров для каждой работы, то все работы из $\mathcal{J}'$ не могут быть выполнены в течение интервала I из-за некоторых предшествующих работ. В качестве следующей работы j_{i+1} выбирается любая предшествующая работа j_i, которая выполняется в интервале I и завершается в конце I или позже.

Эта процедура завершается, когда критическая цепочка содержит работу, которая начинается до всех интервалов в $I_1 \cup I_2$. По построению работы критической цепочки покрывают все интервалы в $I_1 \cup I_2$.

Теперь рассмотрим любую работу j полученной критической цепи. Если $m_j = \delta_j \le \mu$, то эта работа имеет одну и ту же длину и в расписании и в графе G^δ из нижней оценки. Работа может быть выполнена в I_1 и I_2. Если $m_j = \mu < \delta_j$, то эта работа имеет длительность $\bar{p}_j = \frac{p_j}{\mu}$ в расписании и длину $\frac{p_j}{\delta_j}$ в графе G^δ. Таким образом, $\frac{p_j}{\delta_j} \ge \frac{\mu}{m}\bar{p}_j$ и работа может быть выполнена только в I_2. Нижняя оценка на длину критической цепи в терминах графа G^δ равна $|I_1| + \frac{\mu}{m}|I_2|$. В то же время эта нижняя оценка $|I_1| + \frac{\mu}{m}|I_2| \le L^\delta$, поскольку L^δ является длиной критического пути в G^δ. $\square$

Лемма 24 *Пусть* $\frac{m}{\mu} \le \frac{2m-\mu}{m-\mu+1}$, *тогда* $C^{apx}_{\max} \le \frac{2m-\mu}{m-\mu+1} C^*_{\max}$.

Доказательство. Из неравенства в предположении леммы имеем $(m-\mu+1) \le \frac{\mu}{m}(2m-\mu) = \frac{\mu}{m}(m-\mu)+\mu$, следовательно, $(m-2\mu+1) \le \frac{\mu}{m}(m-\mu)$. Тогда, используя (138) и лемму 23, получаем

$$(m-\mu+1)C^{apx}_{\max} \le m\mathcal{A} + (m-\mu)|I_1| + (m-2\mu+1)|I_2| \le$$

$$m\mathcal{A} + (m-\mu)|I_1| + \frac{\mu}{m}(m-\mu)|I_2| \le m\mathcal{A} + (m-\mu)L^\delta.$$

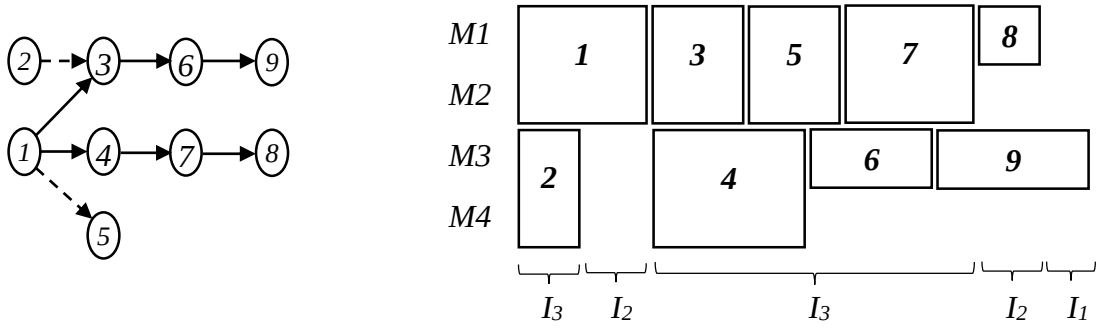


Рисунок 15 — Расписание S, построенное алгоритмом 6, $L^\delta = 13$, $C_{\max} = 18$. Здесь $n = 9$, $m = 4$, $\mu = 2$, $p = (8, 4, 6, 10, 6, 4, 8, 2, 5)$, $\delta = (4, 2, 3, 2, 3, 1, 2, 1, 1)$, $P^\delta = (2, 2, 2, 5, 2, 4, 4, 2, 5)$, $P^{apx} = (4, 2, 3, 5, 3, 4, 4, 2, 5)$.

Поскольку $\max\{\mathcal{A};\ L^\delta\} \le C^*_{\max}$, то требуемое неравенство имеет место:

$$(m - \mu + 1)C^{apx}_{\max} \le mC^*_{\max} + (m - \mu)C^*_{\max} = (2m - \mu)C^*_{\max}.$$

□

Лемма 25 *Пусть* $\frac{m}{\mu} \ge \frac{2m-\mu}{m-\mu+1}$, *тогда* $C^{apx}_{\max} \le \frac{m}{\mu}C^*_{\max}$.

Доказательство. Неравенство из условия леммы эквивалентно $(m-\mu) \le \frac{m}{\mu}(m - 2\mu + 1)$. Следовательно, из (138) и леммы 23 имеем

$$(m - \mu + 1)C^{apx}_{\max} \le m\mathcal{A} + (m - \mu)|I_1| + (m - 2\mu + 1)|I_2| \le$$

$$m\mathcal{A} + \frac{m}{\mu}(m - 2\mu + 1)|I_1| + (m - 2\mu + 1)|I_2| \le m\mathcal{A} + \frac{m}{\mu}(m - 2\mu + 1)L^\delta.$$

Ввиду $\max\{\mathcal{A};\ L^\delta\} \le C^*_{\max}$ получаем:

$$(m - \mu + 1)C^{apx}_{\max} \le mC^*_{\max} + \frac{m}{\mu}(m - 2\mu + 1)C^*_{\max} = \frac{m}{\mu}(m - \mu + 1)C^*_{\max}.$$

□

Лемма 26 *Для каждого $m \geq 2$ имеем*

$$\min_{1 \leq \mu \leq \frac{m+1}{2}} \max \left\{ \frac{m}{\mu}; \frac{2m-\mu}{m-\mu+1} \right\} < \frac{3+\sqrt{5}}{2} \approx 2.62.$$

Минимум достигается на одном из ближайших целых значений $\frac{1}{2}(3m - \sqrt{5m^2-4m})$.

Доказательство. Функция $f_1(\mu) = \frac{m}{\mu}$ убывает, а функция $f_2(\mu) = \frac{2m-\mu}{m-\mu+1}$ возрастает. Поэтому равенство $f_1(\mu) = f_2(\mu)$ достигает минимум на дробном значении $\tilde{\mu} = \frac{1}{2}(3m - \sqrt{5m^2-4m})$ (это корень уравнения $\mu^2 - 3m\mu + m^2 + m = 0$). В этом случае имеем

$$\frac{m}{\tilde{\mu}} = \frac{m}{(3m - \sqrt{5m^2-4m})/2} = \frac{m(3m + \sqrt{5m^2-4m})}{(2m^2+2m)}$$

$$< \frac{3m + \sqrt{5}m}{2m} = \frac{3+\sqrt{5}}{2}.$$

Отметим, что $f_1(\mu) = \frac{3+\sqrt{5}}{2}$ для $\mu = \frac{3-\sqrt{5}}{2}m$ и $f_2(\mu) = \frac{3+\sqrt{5}}{2}$ для $\mu = \frac{3-\sqrt{5}}{2}m + \frac{\sqrt{5}+1}{2}$. Длина интервала $\left[\frac{3-\sqrt{5}}{2}m,\ \frac{3-\sqrt{5}}{2}m + \frac{\sqrt{5}+1}{2}\right]$ больше единицы, и этот интервал содержит значение $\tilde{\mu}$. Таким образом, минимум $\max\left\{\frac{m}{\mu}; \frac{2m-\mu}{m-\mu+1}\right\} < \frac{3+\sqrt{5}}{2}$ и достигается на одном из ближайших целочисленных значений к $\tilde{\mu}$ в случае целочисленных значений μ. □

В результате справедлива следующая теорема.

Теорема 33 *Для задачи $P|any, \delta_j, prec, energy|C_{\max}$ можно построить $\left(\frac{3+\sqrt{5}}{2}\right)$-приближенное расписание за полиномиальное время.*

Рассмотрим случай, когда $\delta_j \leq qm$ для всех $j \in \mathcal{J},\ 0 < q < 1$. Если назначить число используемых процессоров $m_j = \delta_j$ для всех работ, и построить допустимое расписание с помощью алгоритма списочного типа при учете частичного порядка, то будет получено $\left(\frac{2-q}{1-q}\right)$-приближенное решение задачи, как и для жестких работ. Легко видеть, что $\frac{2-q}{1-q} > \frac{3+\sqrt{5}}{2}$ для всех $q > \frac{\sqrt{5}-1}{\sqrt{5}+1} \approx 0.38$.

Статус сложности задачи с мягкими работами при наличии частичного порядка является открытым вопросом.

2.3.5 Нижняя оценка на целевую функцию

Построим полиномиальный алгоритм для решения задачи поиска нижней оценки на целевую функцию (124)-(128). Рассмотрим две следующие подзадачи.

Подзадача (P1)

$$\frac{1}{m}\sum_{j\in\mathcal{J}} p_j \to \min,$$

$$\sum_{j\in\mathcal{J}} W_j^{\alpha} p_j^{1-\alpha} \le E,$$

$$p_j \ge 0,\ j \in \mathcal{J}.$$

Подзадача (P2)

$$\max_{j\in\mathcal{J}} \frac{p_j}{\delta_j} \to \min,$$

$$\sum_{j\in\mathcal{J}} W_j^{\alpha} p_j^{1-\alpha} \le E,$$

$$p_j \ge 0,\ j \in \mathcal{J}.$$

Задачи (P1) и (P2) являются упрощенными версиями задачи (124)-(128). Поэтому, если оптимальное решение (P1) или (P2) является допустимым решением (124)-(128), то оно также является оптимальным решением (124)-(128).

Алгоритм для (124)-(128) состоит из трех шагов. На первых двух шагах решаются задачи (P1) и (P2), а на третьем шаге формируется комбинация (P1) и (P2) путем приравнивания значений целевых функций.

Первый шаг. Перед решением задачи (P1), проверим условие

$$\max_{j\in\mathcal{J}} \frac{W_j}{\delta_j} \le \frac{1}{m}\sum_{j\in\mathcal{J}} W_j. \tag{139}$$

Если имеет место неравенство (139), то оптимальное решение задачи (P1) является оптимальным решением задачи (124)-(128), иначе переходим ко второму шагу. Действительно, задача (P1) соответствует слу-

чаю, когда требуется составить расписание выполнения работ на одном процессоре. Поэтому в оптимальном решении (P1) все работы имеют одинаковую скорость s, которую можно найти из уравнения

$$\sum_{j\in\mathcal{J}} W_j s^{\alpha-1} = E.$$

Длительности работ тогда равны

$$p_i = \frac{W_i}{s} = \frac{W_i \left(\sum_{j\in\mathcal{J}} W_j\right)^{\frac{1}{\alpha-1}}}{E^{\frac{1}{\alpha-1}}},\ i \in \mathcal{J}. \tag{140}$$

Условие (139) гарантирует, что имеет место неравенство $\max_{j\in\mathcal{J}} \frac{p_j}{\delta_j} \leq \frac{1}{m}\sum_{j\in\mathcal{J}} p_j$ для длительностей работ (140).

Второй шаг. В первую очередь проверим условие

$$\sum_{j\in\mathcal{J}} \delta_j \leq m, \tag{141}$$

которое указывает на то, что оптимальное решение задачи (P2) является оптимальным решением задачи (124)-(128). Задача (P2) соответствует одновременному выполнению работ. В таком случае все работы должны иметь одинаковую длительность на всех используемых процессорах

$$\frac{p_i}{\delta_i} = \frac{\left(\sum_{j\in\mathcal{J}} \delta_j W_j^{\alpha}\right)^{\frac{1}{\alpha-1}}}{E^{\frac{1}{\alpha-1}}},\ i \in \mathcal{J}. \tag{142}$$

Условие (141) гарантирует, что $\frac{1}{m}\sum_{j\in\mathcal{J}} p_j \leq \max_{j\in\mathcal{J}} \frac{p_j}{\delta_j}$ для длительностей работ (142). Однако, если общее количество процессоров, требуемых всеми работами, больше m, то переходим к третьему шагу.

Третий шаг. Если ни неравенство (139), ни неравенство (141) не выполняются, то согласно необходимым и достаточным условиям Каруша-Куна-Таккера, для оптимального решения задачи (124)-(128) должно выполняться следующее равенство:

$$\frac{1}{m}\sum_{j\in\mathcal{J}} p_j = \max_{j\in\mathcal{J}} \frac{p_j}{\delta_j}.$$

В результате имеем задачу (P3):

$$\sum_{j\in\mathcal{J}} p_j \to \min,$$

$$\max_{j\in\mathcal{J}} \frac{p_j}{\delta_j} - \frac{1}{m}\sum_{j\in\mathcal{J}} p_j = 0,$$

$$\sum_{j\in\mathcal{J}} W_j^{\alpha} p_j^{1-\alpha} \le E,$$

$$p_j \ge 0,\ j \in \mathcal{J}.$$

Решение (P3) начинается с оптимального решения (P1), где длительности работ на одном процессоре равны

$$p_i = \frac{W_i \left(\sum_{j\in\mathcal{J}} W_j\right)^{\frac{1}{\alpha-1}}}{E^{\frac{1}{\alpha-1}}},\ i \in \mathcal{J},$$

а длительности работ на максимально возможном количестве процессоров равны $\frac{p_i}{\delta_i},\ i \in \mathcal{J}$.

В результате имеем, что длительности работ $\frac{p_j}{\delta_j}$ пропорциональны $\frac{W_j}{\delta_j}$. Упорядочиваем работы по невозрастанию $\frac{W_1}{\delta_1} \ge \frac{W_2}{\delta_2} \ge \ldots \frac{W_n}{\delta_n}$. Пусть l обозначает работу с минимальным отношением $\frac{W_l}{\delta_l}$ таким, что

$$\frac{W_l}{\delta_l} > \frac{1}{m}\sum_{j\in\mathcal{J}} W_j. \tag{143}$$

Это условие эквивалентно условию $\frac{p_l}{\delta_l} > \frac{1}{m}\sum_{j\in\mathcal{J}} p_j$.

Поскольку целевая функция задачи (P3) является линейной, то необходимым условием для оптимальности решения (P3) является

$$\frac{p_j}{\delta_j} = \frac{1}{m}\sum_{i\in\mathcal{J}} p_i,\ j = 1, \ldots, l.$$

Затем формулируем новую задачу (P4):

$$\sum_{j\in\mathcal{J}} p_j \to \min,$$

$$\frac{p_j}{\delta_j} = \frac{1}{m}\sum_{i\in\mathcal{J}} p_i,\ j = 1,\dots,l,$$

$$\sum_{j\in\mathcal{J}} W_j^{\alpha} p_j^{1-\alpha} \le E,$$

$$p_j \ge 0,\ j \in \mathcal{J}.$$

Для нахождения оптимального решения задачи (P4) используем метод Лагранжа, строя функцию Лагранжа

$$L(p_j,\lambda_j) = \sum_{j\in\mathcal{J}} p_j + \sum_{j=1}^{l}\lambda_j\left\{\frac{p_j}{\delta_j} - \sum_{i\in\mathcal{J}}\frac{p_i}{m}\right\} + \lambda_{l+1}\left(\sum_{j\in\mathcal{J}} W_j^{\alpha} p_j^{1-\alpha} - E\right).$$

Вычисляем частные производные, приравниваем их к нулю и находим длительности работ.

$$\frac{p_j}{\delta_j} = \frac{\sum_{i=l+1}^{n} W_i}{(m - \sum_{i=1}^{l}\delta_i)}C,\ j = 1,\dots,l, \tag{144}$$

$$\frac{p_j}{\delta_j} = \frac{W_j}{\delta_j}C,\ j = l+1,\dots,n, \tag{145}$$

где

$$C = \left(\frac{E}{\sum_{j=1}^{l}\frac{W_j^{\alpha}(\delta_j)^{1-\alpha}(\sum_{i=l+1}^{n} W_i)^{1-\alpha}}{(m-\sum_{i=1}^{l}\delta_i)^{1-\alpha}} + \sum_{j=l+1}^{n} W_j}\right)^{\frac{1}{1-\alpha}}.$$

Отметим, что $m > \sum_{i=1}^{l}\delta_i$ ввиду условия (143) для выбора индекса l.

Если для полученных длительностей работ $l+1,\dots,n$ выполняется следующее неравенство

$$\max_{j=l+1,\dots,n}\frac{p_j}{\delta_j} \le \frac{1}{m}\sum_{j\in\mathcal{J}} p_j, \tag{146}$$

то получаем оптимальное решение задачи (124)-(128). В противном случае переходим к решению задачи (P4) с новым значением l.

Трудоемкость алгоритма составляет $O(n\log(nm))$.

2.4 Задача open-shop

Цеховая задача составления расписаний имеет три основных варианта: одинаковые маршруты (flow-shop), различные маршруты (job-shop) и нефиксированные маршруты (open-shop) [13, 165]. В первых двух случаях для работ заданы жесткие маршруты прохождения по машинам, а в последнем случае работы могут обслуживаться на машинах в произвольном порядке. В текущем разделе рассматривается задача open-shop (цеховая задача открытого типа). Известно, что в случае, когда прерывания допускаются, задача полиномиально разрешима [119]. Задача без прерываний с двумя машинами полиномиальна разрешима, но является NP-трудной, начиная с трех машин [119]. Также широко известен 2-приближенный алгоритм для случая произвольного числа машин [41].

Современные производственные и компьютерные системы порождают новые варианты задачи open-shop, где в роли машин выступают процессоры и учитывается потребление ресурсов возобновимого и невозобновимого типов. Возобновимые ресурсы приводят к ограничениям в каждый момент времени с последующим возобновлением в следующий момент (например, кэш-память). Задача open-shop с одним возобновимым ресурсом полиномиально разрешима (алгоритм базируется на решении задачи о максимальном потоке специального вида) [139], а в случае двух таких ресурсов уже является NP-трудной (доказывается с помощью сводимости задачи Разбиение).

Невозобновимые ресурсы ограничиваются на некоторый период и распределяются между работами в различные моменты времени. Длительности работ могут зависеть от объемов потребления ресурсов. Исследование вычислительной сложности задачи open-shop с длительностями работ, зависящими от потребления ограниченного ресурса, представлено в работе [216]. Также в указанной статье предложено обобщение алгоритмов известных для классической задачи open-shop на случай наличия общего невозобновимого ресурса.

Будем рассматривать задачу open-shop с учетом расхода энергии в следующей постановке. Дано множество работ $J = \{1, \ldots, n\}$ и множе-

ство процессоров $P = \{1, \ldots, m\}$. Каждая работа j состоит из m операций $O_{j1}, \ldots, O_{jm}$, причем операция O_{ji} выполняется на процессоре i и имеет объем равный W_{ji}. В каждый момент времени каждый процессор может выполнять не более одной операции, а также в каждый момент времени не может выполняться более одной операции каждой работы. Операции могут выполняться с разной скоростью. Если операция O_{ji} выполняется со скоростью s_{ji}, то потребление энергии в единицу времени (мгновенная мощность) составляет s_{ji}^{α} ($\alpha > 1$ – константа). Множество возможных скоростей может быть дискретным или непрерывным. Динамическая составляющая потребления энергии равна суммарному потреблению энергии процессорами во время выполнения операций $\sum_{j \in J} \sum_{i \in P} s_{ji}^{\alpha} \left(\frac{W_{ji}}{s_{ji}} \right)$. Также имеет место статическая составляющая энергопотребления, равная $F_{\text{stat}} \cdot C_{\max}$, где $C_{\max}$ – общее время работы системы (длина расписания, makespan), $F_{\text{stat}} \geq 0$ – заданная константа (статическая мощность системы). Учет статической части характерен для систем, в которых допускается управление их выключением, и соответствует $F_{\text{stat}} > 0$. Если статическая составляющая не учитывается при оптимизации, например, когда сервер работает без перерывов или по заданному графику, то это задается через $F_{\text{stat}} = 0$.

Требуется построить допустимое расписание σ выполнения операций, при котором потребление энергии $\left(\sum_{j \in J} \sum_{i \in P} s_{ji}^{\alpha} \left(\frac{W_{ji}}{s_{ji}} \right) + mC_{\max}(\sigma) F_{\text{stat}} \right)$ не превышает заданной величины E и длина расписания $C_{\max}(\sigma)$ минимальна.

Задача исследуется как при допустимости прерывания операций (системы с общей памятью), так и при их запрете (системы с распределенной памятью). Также задача рассматривается в случае многопроцессорных операций [18], когда каждая работа состоит из m' операций и операция $i = 1, \ldots, m'$ задействует некоторое подмножество процессоров $fix_i \subseteq P$, $|fix_i| \geq 1$. Рассматриваемая задача будет обозначаться через $O|(fix_i),\ (pmtn),\ (discr),\ energy|C_{\max}$.

В работе [31] предложен 2-приближенный алгоритм для задачи openshop с учетом расхода энергии (динамическая составляющая). Также известны статьи, где предложены и экспериментально исследованы эвристические алгоритмы для одностадийных задач [147, 173]. Результаты

эксперимента показали, что хорошо зарекомендовали себя алгоритмы списочного типа с локальными улучшениями и алгоритмы, построенные по аналогии с известными жадными стратегиями типа first-fit, next-fit и best-fit для задачи об упаковке в контейнеры.

В литературе также представлены результаты по исследованию задач составления расписаний в производственных системах с оптимизацией расхода энергии на различных уровнях [109, 188]. Предложены модели частично целочисленного программирования, конструктивные эвристики и метаэвристики для решения задач цехового типа [129, 183, 211] с учетом специфики структуры решений, когда работы имеют дискретный набор вариантов выполнения с различными длительностями и энергопотреблением. Отметим, что близкие к исследуемой задаче постановки, где рассматривается два критерия – потребление энергии и длина расписания, также имеют место в современных исследованиях (см., например, [109, 183, 211, 213, 225]).

Основные результаты раздела. Исследуется вычислительная сложность и строятся алгоритмы с гарантированной оценкой точности для различных вариантов задачи open shop с учетом расхода энергии, возникающей в компьютерных системах. Предлагаемые методы решения позволяют обобщить известные результаты для классической задачи open-shop без учета ресурсных ограничений с критерием минимизации длины расписания на случай учета расхода энергии и получить новые результаты.

2.4.1 Однопроцессорные операции

Рассмотрим задачу open-shop с учетом ограничения на расход энергии для однопроцессорных операций при непрерывном и дискретном наборах скоростей. Выделим полиномиально разрешимые и NP-трудные частные случаи, построим полиномиальный алгоритм с гарантированной оценкой точности.

Непрерывное множество скоростей. В этом параграфе установим NP-трудность или полиномиальную разрешимость различных вариантов задачи open-shop при условии, что скорость может принимать произволь-

ное значение больше нуля.

Теорема 34 *Задача* $O3|energy|C_{\max}$ *является NP-трудной.*

Доказательство. В случае $m \geq 3$ задача является NP-трудной. Для доказательства используется подход, представленный в первой главе и основанный на ограничении длины расписания и общего потребления ресурса в распознавательном варианте задачи. Строим полиномиальную сводимость задачи Разбиение [111]: Дано множество элементов $A = \{a_1, a_2, \ldots, a_{2n_0}\}$ и вес e_j каждого элемента $a_j \in A$, такие что

$$\sum_{a_j \in A} e_j = 2B.$$

Требуется определить, может ли множество A быть разбито на подмножества A_1 и A_2 такие, что $\sum_{a_i \in A_1} e_i = \sum_{a_i \in A_2} e_i = B$. Построим пример задачи open-shop с ограничением на расход энергии: $n = 2n_0 + 1$, $m = 3$, $E = 18B$, $W_{ji} = e_j,\ j = 1, \ldots, 2n_0$, $W_{2n_0+1,i} = B$ для $i = 1, 2, 3$. Покажем, что расписание σ с $C_{\max}(\sigma) \leq 3B$ существует тогда и только тогда, когда в задаче Разбиение ответ «да».

Отметим, что общий объем работ составляет $\sum_{j=1}^{n} \sum_{i=1}^{3} W_{ji} = 9B$. С учетом выпуклости функции расчета мгновенной мощности, а также заданных ограничений на расход энергии и длину расписания, работы в допустимом расписании могут выполняться только с единичной скоростью. Причем как статическая, так и динамическая составляющие ограничены величиной $9B$.

Если в задаче Разбиение ответ «да», то есть существует разбиение A на два подмножества A_1 и A_2, то допустимое расписание строится так, что первый процессор выполняет операции согласно последовательности $(2n_0 + 1, \pi_{A_1}, \pi_{A_2})$, второй – $(\pi_{A_1}, 2n_0 + 1, \pi_{A_2})$ и третий – $(\pi_{A_2}, \pi_{A_1}, 2n_0 + 1)$. Перестановка π_{A_i} задает произвольное упорядочение элементов из $A_i,\ i = 1,\ 2$.

Если существует допустимое расписание, то ввиду загрузки все процессоры работают без простоев в интервале $(0, 3B]$. Причем работа $2n_0+1$ выполняется на одном из процессоров в каждый момент времени, так как длительность всякой ее операции равна B. Значит на одном из процес-

соров эта работа выполняется в интервале $(B, 2B]$, а остальные работы разбиваются на две равные части по объему и выполняются на этом процессоре в интервалах $(0, B]$ и $(2B, 3B]$. Эти части и формируют требуемые подмножества в задаче Разбиение, гарантируя ответ «да».

При отсутствии статической составляющей в оценке потребления энергии ($F_{\text{stat}} = 0$) задача также является NP-трудной (строится аналогичная сводимость, но верхняя граница на потребление энергии полагается равной $9B$). □

Теорема 35 *[31, теорема 5] Для задачи $O|energy|C_{\max}$ при учете только динамической составляющей потребления энергии ($F_{\text{stat}} = 0$) может быть найдено 2-приближенное решение за полиномиальное время.*

Доказательство. Для построения нижней оценки на целевую функцию будем использовать следующую модель выпуклого программирования [31]. Пусть переменные p_{ji} соответствуют длительностям операций $O_{ji},\ j \in J,\ i \in P$.

$$LB \to \min, \tag{147}$$

$$LB \geq \sum_{j=1}^{n} p_{ji},\ i \in P, \tag{148}$$

$$LB \geq \sum_{i=1}^{m} p_{ji},\ j \in J, \tag{149}$$

$$E \geq \sum_{j=1}^{n} \sum_{i=1}^{m} p_{ji} \left(\frac{W_{ji}}{p_{ji}} \right)^{\alpha}, \tag{150}$$

$$p_{ji} \geq 0,\ j \in J,\ i \in P. \tag{151}$$

Задача (147)–(151) может быть решена за полиномиальное время (см., например, [57, 193]). Далее применяем алгоритм списочного типа с найденными длительностями работ: Как только освобождается процессор $i \in P$ запускаем на нем операцию O_{ji} работы $j \in J$, которая в этот момент не выполняется ни на каком другом процессоре. Гарантированная оценка точности этого алгоритма равна двум по отношению к стандартной нижней оценке на длину расписания [41, 221]:

$$\max\left\{\max_{i\in P}\sum_{j=1}^{n}p_{ji};\ \max_{j\in J}\sum_{i=1}^{m}p_{ji}\right\}.$$

□

Теорема 36 *Задача* $O|pmtn, energy|C_{\max}$ *является полиномиально разрешимой.*

Доказательство. Задача с прерываниями полиномиально разрешима для произвольного числа процессоров. Сначала находится оптимальное решение по модели (147)–(149), (151) и

$$E \geq m \cdot LB \cdot F_{\text{stat}} + \sum_{j=1}^{n}\sum_{i=1}^{m} p_{ji}\left(\frac{W_{ji}}{p_{ji}}\right)^{\alpha}. \qquad (152)$$

Это решение гарантирует получение нижней оценки

$$LB = \max\left\{\max_{i\in P}\sum_{j=1}^{n}p_{ji};\ \max_{j\in J}\sum_{i=1}^{m}p_{ji}\right\}$$

и соответствующих длительностей работ. Далее для этих длительностей строится оптимальное расписание с прерываниями и значением целевой функции равным LB с помощью известного алгоритма поиска оптимального расписания задачи с прерываниями при заданных длительностях работ (см., например, [13, 119]). □

Теорема 37 *Задача* $O2|energy|C_{\max}$ *полиномиально разрешима.*

Доказательство. В случае двух процессоров задача без прерываний полиномиально разрешима. Сначала определим нижнюю оценку на целевую функцию и длительности работ по модели (147)–(149), (151)–(152) при $m = 2$.

Далее расписание с длиной LB отыскивается, например, по следующему правилу. Обозначим через J_1 работы, для которых $p_{j1} \leq p_{j2}$. Пусть $J_2 = J \setminus J_1$. Идентифицируем работу $k \in J_1$, для которой $p_{k2} \geq \max\{p_{j1},\ j \in J_1\}$, и работу $r \in J_2$, для которой $p_{r1} \geq \max\{p_{j2},\ j \in J_2\}$.

Рассмотрим расписания, задаваемые перестановками вида

$$(\sigma'; \pi_{J_1 \setminus \{k\}}; \pi_{J_2 \setminus \{r\}}; \sigma''),$$

где $\{\sigma',\ \sigma''\} = \{k,\ r\}$ (то есть работы $k,\ r$ стоят в начале или в конце расписания) и $\pi_{J'}$ – произвольная перестановка элементов из J'. Тогда одно из расписаний указанного вида будет иметь длину, равную LB [13]. Трудоемкость описанной процедуры равна $O(n)$. Известны также следующие алгоритмы решения задачи open shop с двумя машинами при фиксированных длительностях операций за время $O(n)$: алгоритм Гонзалеза Т., Сани С. [119], алгоритм Пинедо М., Шраге Л. [82], алгоритм де Верра Д. [240], алгоритм Сопера А. [222], алгоритм Храмовой А. [145].

Отметим, что если в задаче учитывается только динамическая составляющая ($F_{\text{stat}} = 0$), то она всегда имеет допустимое решение, а если учитываются обе составляющие (статическая и динамическая), то допустимого решения может не существовать. $\square$

Сейчас рассмотрим вариант задачи, в котором скорость должна быть постоянной и одинаковой для всех процессоров. В варианте задачи, когда учитывается только динамическая составляющая ($F_{\text{stat}} = 0$), скорость s находится из уравнения

$$s^{\alpha-1} \sum_{j=1}^{n} \sum_{i=1}^{m} W_{ji} = E,$$

и длительности операций $p_{ji} = \frac{W_{ji}}{s} = W_{ji} \left(\frac{\sum_{j=1}^{n} \sum_{i=1}^{m} W_{ji}}{E} \right)^{\frac{1}{\alpha-1}},\ j \in J,\ i \in P$. Далее с помощью описанных ранее подходов при фиксированных длительностях строится 2-приближенное расписание (задача без прерываний и произвольное число процессоров) или оптимальное решение (двухпроцессорная задача без прерываний или задача с прерываниями).

В варианте задачи, когда учитываются динамическая и статическая составляющие на расход энергии, будем рассматривать только случаи, когда при фиксированных длительностях операций длина оптимального расписания совпадает со стандартной нижней оценкой (двухпроцессорная задача без прерываний или задача с прерываниями). Обозначим

через

$$LBW := \max\left\{\max_{i\in P}\sum_{j=1}^{n} W_{ji};\ \max_{j\in J}\sum_{i=1}^{m} W_{ji}\right\},$$

тогда задача сводится к поиску минимальной скорости среди скоростей $s > 0$, для которых имеет место неравенство:

$$W \cdot s^{\alpha-1} + Vs^{-1} \le E,$$

или

$$W \cdot s^{\alpha} \le E \cdot s - V, \tag{153}$$

где $W := \sum_{j\in J}\sum_{i\in P} W_{ji}$, $V := LBW \cdot F_{\text{stat}} \cdot m$. Построим касательную к графику функции $f(s) = W \cdot s^{\alpha}$ (левая часть неравенства), параллельную прямой $g(s) = E \cdot s - V$ (правая часть неравенства). Тогда точка касания соответствует скорости $\bar{s} = \left(\frac{E}{W\alpha}\right)^{\frac{1}{\alpha-1}}$ и уравнение касательной $y(s) = E \cdot s - V'$, где $V' = E^{\frac{\alpha}{\alpha-1}} \cdot (\alpha W)^{\frac{1}{\alpha-1}} \left(1 - \frac{1}{\alpha}\right)$.

Следовательно, для $s > 0$ неравенство (153) не имеет решений при $V > E^{\frac{\alpha}{\alpha-1}} \cdot (\alpha W)^{\frac{1}{\alpha-1}} \left(1 - \frac{1}{\alpha}\right)$. В противном случае неравенство (153) имеет решения и единственное минимальное из них может быть найдено с заданной точностью с помощью численных методов (например, метод дихотомии, см. [167]).

Дискретный набор скоростей. Рассмотрим случай, когда имеется дискретный набор возможных скоростей и допускаются прерывания выполнения операций.

Теорема 38 *Задача* $O|discr,\ pmtn,\ energy|C_{\max}$ *полиномиально разрешима.*

Доказательство. Покажем, что для указанной задачи оптимальное решение может быть найдено за полиномиальное время. Обозначим через C множество допустимых конфигураций операций на процессорах, которые могут выполняться параллельно. Каждая конфигурация c задается набором операций O_c и скоростью $s_{j,i,c}$ каждой операции $O_{ji} \in O_c$. Тогда мгновенная мощность (потребление энергии в единицу времени)

для конфигурации $c \in C$ вычисляется как $E_c = \sum_{O_{ji} \in O_c} (s_{j,i,c})^{\alpha} + F_{\text{stat}} \cdot m$ (учитываются статическая и динамическая составляющие).

Пусть переменная x_c отвечает за длительность конфигурации c в расписании. Тогда для поиска оптимального решения можно использовать следующую модель выпуклого программирования.

$$\sum_{c \in C} x_c \to \min,$$

$$\sum_{c \in C:\ O_{ji} \in O_c} \frac{s_{j,i,c} x_c}{W_{ji}} \geq 1,\ O_{ji} \in O_c,$$

$$-\sum_{c \in C} E_c x_c \geq -E,$$

$$x_c \geq 0,\ c \in C.$$

Построим двойственную задачу

$$\sum_{O_{ji} \in O_c} \lambda_{ji} - E\mu \to \max, \tag{154}$$

$$\sum_{O_{ji} \in O_c} \frac{s_{j,i,c} \lambda_{ji}}{W_{ji}} - E_c \mu \leq 1,\ c \in C, \tag{155}$$

$$\lambda_{ji} \geq 0,\ O_{ji} \in O_c, \tag{156}$$

$$\mu \geq 0. \tag{157}$$

Двойственная задача (154)–(157) решается методом эллипсоидов с отделяющим оракулом [123], который при заданном решении (λ_{ji}, μ) определяет является ли решение допустимым, и если ответ отрицательный, то идентифицирует нарушенное ограничение.

Опишем как работает отделяющий оракул в рассматриваемом случае. Согласно ограничению (155) требуется найти максимум суммы по всем конфигурациям. Если он больше 1, то найдено нарушенное ограничение. В противном случае решение допустимо.

Перепишем ограничение (155) в виде:

$$\sum_{O_{ji} \in O_c} \left(\frac{s_{j,i,c} \lambda_{ji}}{W_{ji}} - (s_{j,i,c})^{\alpha} \mu \right) \leq F_{\text{stat}} \cdot m \cdot \mu + 1, \ c \in C.$$

Для каждой операции O_{ji} скорость v_{ji}, на которой достигается максимум $\frac{s_{j,i,c} \lambda_{ji}}{W_{ji}} - (s_{j,i,c})^{\alpha} \mu$, отыскивается как одна из ближайших к величине $\left(\frac{\lambda_{ji}}{W_{ji} \alpha \mu} \right)^{\frac{1}{\alpha - 1}}$. Далее для поиска максимума $\sum_{O_{ji} \in O_c} \left(\frac{v_{ji} \lambda_{ji}}{W_{ji}} - (v_{ji})^{\alpha} \mu \right)$ строим следующий двудольный граф $G = (J, P, E)$. Доли вершин J и P соответствуют работам и процессорам, соответственно. Ребро $(i, j) \in E$ соответствует операции O_{ji} и имеет вес $B_{ji} := \left(\frac{v_{ji} \lambda_{ji}}{W_{ji}} - (v_{ji})^{\alpha} \mu \right)$. Далее в отделяющем оракуле отыскивается паросочетание (независимое множество ребер) максимального веса за полиномиальное время [92]. Если вес найденного паросочетания больше $F_{\text{stat}} \cdot m \cdot \mu + 1$, то решение недопустимо и найдено нарушенное ограничение, иначе решение допустимо.

В результате двойственная задача разрешима за полиномиальное время, а значит и исходная задача разрешима за полиномиальное время [123]. □

В случае поиска одной дискретной скорости для всей системы можно выбирать одну из ближайших скоростей к оптимальной скорости, найденной для непрерывного случая.

2.4.2 Многопроцессорные операции и ограниченное число стадий

В данном разделе рассматривается задача open shop с ограничением на расход энергии, когда работа состоит из 2 или 3 стадий, но операции задействуют при своем выполнении несколько процессоров. Обозначим через fix_i множество процессоров, которые задействует операция $i = 1, 2, 3$. В этом случае будем считать W_{ji} объемом операции Q_{ji} на каждом из используемых процессоров, $i = 1, 2, 3, \ j \in J$.

При двух стадиях имеют место два случая: стадии совместны ($fix_1 \cap fix_2 = \emptyset$) и стадии несовместны ($fix_1 \cap fix_2 \neq \emptyset$).

Если стадии несовместны, то получаем случай, эквивалентный задаче составления расписаний на одном процессоре, где порядок следова-

ния операций может быть любым, а длительности p_{ji} находятся из модели выпуклого программирования с помощью условий Каруша-Куна-Таккера:

$$\sum_{j\in J}(p_{j1}+p_{j2})\to\min,$$

$$\sum_{j\in J}|fix_1|p_{j1}\left(\frac{W_{j1}}{p_{j1}}\right)^{\alpha}+\sum_{j\in J}|fix_2|p_{j2}\left(\frac{W_{j2}}{p_{j2}}\right)^{\alpha}$$

$$+\left(\sum_{j\in J}(p_{j1}+p_{j2})\right)\cdot F_{\text{stat}}\cdot m\le E.$$

Если стадии совместны, то задача сводится к случаю задачи open shop с двумя процессорами, где длительности операций находятся из модели выпуклого программирования:

$$LB\to\min, \tag{158}$$

$$LB\ge\sum_{j=1}^{n}p_{ji},\ i=1,\ 2, \tag{159}$$

$$LB\ge p_{j1}+p_{j2},\ j\in J, \tag{160}$$

$$E\ge\sum_{j=1}^{n}\sum_{i=1}^{2}|fix_i|\left(\frac{W_{ji}}{p_{ji}}\right)^{\alpha}p_{ji}+LB\cdot F_{stat}\cdot m, \tag{161}$$

$$p_{ji}\ge 0,\ j\in J,\ i=1,\ 2. \tag{162}$$

Теперь перейдем к рассмотрению случая задачи open shop с тремя стадиями при многопроцессорных операциях. Если все три стадии несовместны ($fix_1\cap fix_2\neq\emptyset$, $fix_1\cap fix_3\neq\emptyset$, $fix_2\cap fix_3\neq\emptyset$), то задача сводится к однопроцессорному случаю как и при двух несовместных стадиях из ранее представленных результатов.

Если одна из стадий не совместна с двумя другими (например, $fix_2\cap fix_1\neq\emptyset$, $fix_2\cap fix_3\neq\emptyset$, $fix_1\cap fix_3=\emptyset$), то операции второй стадии можно поставить в любом порядке в начале расписания, а для двух других стадий составить расписание как для случая двух процессоров. При

этом длительности операций могут быть найдены с помощью моделей выпуклого программирования, аналогичных модели (158)–(162).

Если две стадии не совместны друг с другом, а еще одна стадия совместна со всеми (например, $fix_1 \cap fix_2 \neq \emptyset$, $fix_1 \cap fix_3 = \emptyset$, $fix_2 \cap fix_3 = \emptyset$), то операции двух первых стадий можно объединить в одну, и для двух новых стадий (первая совместно со второй и третья) найти решение как для случая двух процессоров. В случае двухпроцессорной задачи open shop всякое расписание с прерываниями можно превратить в расписание без прерываний без увеличения его длины [119], поэтому возможным прерыванием между двумя первыми стадиями можно пренебречь. Модели выпуклого программирования для поиска длительностей операций и нижней границы на длину расписания имеют вид аналогичный модели (158)–(162), но с учетом особенностей комбинации стадий.

Глава 3

Задачи составления расписаний и маршрутизации с запретами работ: одностадийные и многостадийные системы

В третьей главе проводится анализ алгоритмической сложности класса задач составления расписаний и маршрутизации с запретами выполнения работ на определенных этапах в одностадийных и многостадийных системах.

1. Предлагается метод доказательства NP-трудности для задач составления расписаний с предписаниями (запретами) работ.

2. Разрабатывается точный метод решения для класса задач составления расписаний с предписаниями работ мощности не превосходящей 2, трудоемкость которого полиномиальна для «почти всех» систем предписаний; на основе этого подхода предлагаются модели целочисленного линейного программирования с меньшим числом переменных, чем дает классический подход.

3. Метод обобщается на случай предписаний произвольной мощности, основываясь на переборе совершенных паросочетаний в специальном двудольном графе.

Результаты третьей главы опубликованы в работах [5, 9, 49, 98, 100, 249].

3.1 Одностадийные системы

Рассматриваются задачи составления расписаний на перестановках с запретами работ в следующей постановке. Пусть $\mathcal{J}$, $|\mathcal{J}| = n$, — множество работ, а $\mathcal{M}$, $|\mathcal{M}| = m$, — множество машин. Исследуются одностадийные системы [61, 121], где каждая работа состоит из одной операции, и многостадийные системы типа flow-shop, job-shop и open-shop [13, 223],

где каждая работа состоит из нескольких операций. Обозначим через p_{ij} длительность (время обработки) операции i работы j. Для каждой машины $l \in \mathcal{M}$ задано множество позиций $N_l = \{1, \ldots, n_l\}$, в которые могут быть размещены работы (или их операции). Также задана система подмножеств работ $X = \{X^{i,l}\}_{i \in \mathcal{M},\ l \in N_l}$. Здесь $X^{i,l}$ — это подмножество работ (операций), которые могут выполняться в позиции $i \in N_l$ машины $l \in \mathcal{M}$. В результате имеем запреты на выполнение работ в некоторых позициях машин. Требуется назначить работы (или их операции) в позиции машин в соответствии с системой подмножеств работ X так, чтобы полиномиально вычислимый регулярный критерий принимал оптимальное значение. Такие задачи составления расписаний с дополнительными ограничениями на выполнение работ в позициях машин называются в литературе задачами с запретами работ [11] или задачами с предписаниями работ [100].

Очевидно, что при $|X^{i,l}| = n$ для всех позиций машин получаем классическую задачу составления расписаний. Отметим, что задача полиномиально разрешима во многих случаях, когда $|X^{i,l}| = 1$ для всех $l \in \mathcal{M}$, $l \in N_l$.

Задачи с запретами работ имеют приложения в компьютерных системах и производственных предприятиях, где запреты на выполнение работ в позициях машин возникают из-за технологических ограничений, фиксированных маршрутов [69, 105] и рабочих смен [104].

Например, рассмотрим задачу составления расписаний с маршрутизацией [69]. Задан неориентированный взвешенный граф. Вершины соответствуют узлам транспортной сети. Работы размещены в узлах. Веса рёбер соответствуют длительностям перемещения между узлами. Машина посещает узел не более одного раза. Работы выполняются машинами в узлах, где они размещены. Можно рассматривать модели, в которых работы связаны отношениями предшествования, состоят из нескольких стадий и другие ограничения. Цель – минимизировать некоторый регулярный критерий. Такие задачи можно решать в два этапа. Сначала строятся маршруты машин, а затем составляется расписание выполнения работ в позициях маршрутов [68, 69]. Однако, когда маршруты машин фиксированы, то получаем задачу составления расписаний с запре-

тами работ, определяемыми узлами, посещаемыми машинами в позициях маршрутов [68, 69, 105].

В то же время такие задачи играют важную роль в эволюционных алгоритмах, где в операторе кроссинговера (рекомбинации) решается задача оптимальной рекомбинации. Родительские решения обмениваются своими признаками при построении потомков в операторах кроссинговера эволюционных вычислений [29, 204, 244]. Для двух родительских решений задача оптимальной рекомбинации состоит в нахождении наилучшего возможного потомка, который может возникнуть в результате кроссинговера, где значение каждого компонента решения копируется из одного из значений этого компонента в родительских решениях. Такие операторы рекомбинации основаны на свойстве передачи генов [204].

Задача оптимальной рекомбинации в случае одной машины формулируется следующим образом [29, 98]:
Даны две родительские перестановки работ $\pi^1 = (\pi^1_1, \ldots, \pi^1_n)$ и $\pi^2 = (\pi^2_1, \ldots, \pi^2_n)$. Требуется найти перестановку потомка $\pi' = (\pi'_1, \ldots, \pi'_n)$ такую, что

(I) $\pi'_i = \pi^1_i$ или $\pi'_i = \pi^2_i$ для всех $i = 1, \ldots, n$;

(II) π' имеет оптимальное значение заданного критерия среди всех перестановок, удовлетворяющих условию (I).

Тогда работы π^1_i и π^2_i образуют подмножество $X^{i,1}$ для позиции i машины. Задача оптимальной рекомбинации формулируется аналогичным образом для случая нескольких машин.

Исследование вычислительной сложности оптимальной рекомбинации для классических задач маршрутизации и задач булева программирования, перестановочного варианта задачи flow-shop и задачи составления расписаний на одной машине с переналадками было проведено в [5, 9, 95]. Для решения таких задач используются различные подходы, такие как динамическое программирование [244], методы теории графов [9, 103, 229, 232] на основе рекурсии или декомпозиции, и другие. Экспериментальные исследования показали (см., например, [29, 70, 99, 103, 229, 244, 248, 250]), что оптимальная рекомбинация позволяет улучшить результаты эволюционных алгоритмов с рандомизированными операто-

рами. Здесь для решения NP-трудных задач оптимальной рекомбинации в эволюционном поиске используются ограниченные версии точных алгоритмов и жадные методы.

А.И. Сердюков [11] предложил точный алгоритм для задачи коммивояжёра с запретами вершин, основанный на перечислении совершенных паросочетаний в специальном двудольном графе. Этот алгоритм был улучшен в [100]. В [159, 210] исследовалась задача flow-shop на двух машинах с запрещёнными работами в первой или последней позициях. Авторы предложили алгоритм типа ветвей и границ для варианта flow-shop и построили полиномиальные алгоритмы решения.

Известны приближённые алгоритмы для задачи open-shop с маршрутизацией, где маршруты машин строятся на первом шаге, а расписания работ в узлах маршрутов формируются на втором шаге [68, 69]. Однако авторы рассматривают все работы, принадлежащие одному узлу, как одну работу в своих доказательствах. Таким образом, интересно исследовать запреты работ для позиций перестановок машин и поиск возможностей улучшения алгоритмов в этом контексте.

В качестве предыдущих исследований также отметим задачи типа раскраски графа со списками. Некоторые задачи составления расписаний (многостадийные системы, задачи с группировкой машин по технологиям, задачи распределения ресурсов) содержат как подзадачи задачу о раскраске графа, которая состоит в определении того, является ли граф k-раскрашиваемым или нет для некоторого натурального числа k. Известные модификации включают задачу о раскраске графа со списками, μ-раскраску и (λ, μ)-раскраску [50]. Здесь имеют место дополнительные ограничения на цвета, которые могут быть использованы для раскраски вершин или рёбер. Задача о хроматическом числе графа со списками является подзадачей задачи составления расписаний с однопроцессорными работами (соответствуют вершинам) и запретами работ, а задача о хроматическом индексе графа со списками также имеет интерпретацию как составление расписаний двухпроцессорных работ (соответствуют рёбрам) с запретами работ [90]. В задаче типа job-shop имеет место система X подмножеств для двухстадийных работ в контексте времени ожидания между операциями [203].

Основные результаты раздела. Проводятся методы анализа вычислительной сложности задачи с запретами работ для различных регулярных критериев, систем и мощностей подмножеств предписаний из X. Предлагается общий подход к доказательству NP-трудности указанного класса задач с использованием сведения задачи Упорядоченного Разбиения специального вида, когда строится семейство индивидуальных задач с оптимальными решениями без простоев машин. Разрабатываются методы решения задачи с предписаниями мощности, не превосходящей двух, и в общем случае.

Результаты данного раздела позволяют получить новые свойства задач составления расписаний с запретами работ при различных критериях и ограничениях. Эти свойства гарантируют NP-трудность в худшем случае и полиномиальную разрешимость в «почти всех» случаях для практически значимых семейств примеров. Это играет важную роль в эволюционных вычислениях и других итеративных методах, где подзадачи с запретами работ решаются на каждой итерации с помощью методов, основанных на целочисленном программировании или алгоритмах переборного типа.

3.1.1 Задачи на одной машине с запретами работ

Рассмотрим задачу составления расписаний без прерываний с запретами работ на одной машине. Каждая работа $j \in \mathcal{J}$ характеризуется моментом поступления r_j, директивным сроком d_j, длительностью (временем обработки) p_j и весом w_j. Прерывания работ не допускаются. Требуется найти последовательность работ, размещая на i-й позиции работу из заданного подмножества $X^i,\ i = 1, \ldots, n = |\mathcal{J}|$, и минимизируя некоторый регулярный критерий. Заметим, что раннее расписание, соответствующее перестановке работ, даёт наилучшее значение регулярного критерия для этой перестановки.

Напомним классические обозначения:
C_j — время завершения работы j;
$T_j = \max\{0; C_j - d_j\}$ — запаздывание работы j;
$L_j = C_j - d_j$ — временное смещение работы j;

$U_j = \chi_{C_j > d_j}$ — индикатор того, что работа j является запаздывающей:

$$\chi_{C_j>d_j} = \begin{cases} 1, & \text{если } C_j > d_j, \\ 0 & \text{иначе, } j = 1, \ldots, n. \end{cases}$$

Рассматриваются следующие случаи:
$1|r_j|\sum_j C_j$ (минимизируется суммарное время завершения работ);
$1|r_j|\sum_j U_j$ (минимизируется количество запаздывающих работ);
$1|r_j|\sum_j T_j$ (минимизируется суммарное запаздывание);
$1|C_j \le d_j|\sum_j w_j C_j$ (минимизируется взвешенное суммарное время завершения);
$1|r_j|\sum_j w_j U_j$ (минимизируется взвешенное количество запаздывающих работ);
$1|r_j|\sum_j w_j T_j$ (минимизируется взвешенное суммарное запаздывание);
$1|l_{ij}|C_{\max} = \max_j C_j$ (время окончания всех работ, здесь l_{ij} — время переналадки между работами i и j);
$2|d_j = d|\gamma$, $\gamma \in \{C_{\max}, L_{\max}, \sum_j U_j, \sum_j T_j\}$ (две машины)
с дополнительным ограничением на запреты работ.

Отметим, что задачи $1||\sum_j w_j U_j$, $1|C_j \le d_j|\sum_j w_j C_j$ являются NP-трудными в обычном смысле, а задачи $1|r_j|\sum_j C_j$, $1|r_j|L_{\max}$, $1|r_j|\sum_j U_j$, $1|r_j|\sum_j T_j$, $1|l_{ij}|C_{\max}$ являются NP-трудными в сильном смысле в постановке без запретов [61, 170].

NP-трудность.

Для доказательства NP-трудности рассматриваемых задач будем использовать полиномиальное сведение следующей NP-полной задачи Упорядоченное Разбиение: Дано упорядоченное множество $A = \{a_1, \ldots, a_{2n_0}\}$ и вес e_i каждого элемента $a_i \in A$ такой, что $\sum_{a_i \in A} e_i = 2E$ и $e_i < e_{i+1}$, $i = 1, \ldots, 2n_0 - 1$. Вопрос заключается в том, можно ли разбить A на два подмножества A_1 и A_2 так, чтобы

$$\sum_{a_i \in A_1} e_i = \sum_{a_i \in A_2} e_i = E,$$

и подмножество A_1 содержит только один элемент из каждой пары a_{2i-1}, a_{2i}, $i = 1, \ldots, n_0$. Заметим, что $|A_1| = |A_2| = n_0$ в случае существования такого разбиения.

Все представленные далее сводимости строятся к распознавательной версии соответствующей задачи составления расписаний, что приводит к NP-полноте распознавательной версии и NP-трудности оптимизационной версии задачи.

Утверждение 1 *Задача Упорядоченное Разбиение является NP-полной.*

Доказательство. Рассмотрим классическую задачу Разбиение [111] с множеством элементов $B = \{b_1, b_2, \ldots, b_{n_0}\}$. Вопрос заключается в том, можно ли разбить B на два подмножества B_1 и B_2 так, чтобы

$$\sum_{b_i \in B_1} b_i = \sum_{b_i \in B_2} b_i.$$

Положим

$$e_1 = 1,$$

$$e_{2i} = e_{2i-1} + b_i, \ i = 1, \ldots, n_0,$$

$$e_{2i+1} = e_{2i} + 1, \ i = 1, \ldots, n_0 - 1.$$

Рассмотрим произвольные подмножества A_1 и A_2, содержащие по одному элементу $a_i^1 \in A_1$ и $a_i^2 (\neq a_i^1) \in A_2$ из каждой пары $a_{2i-1}, \ a_{2i}, \ i = 1, \ldots, n_0$. Тогда

$$\sum_{i=1}^{n_0-k} e_i^1 < \sum_{i=1}^{n_0} e_i^2 + \sum_{i=n_0-k+1}^{n_0} e_i^1, \ k = 1, \ldots, n_0 - 1,$$

для построенной серии индивидуальных задач, так как

$$\sum_{i=1}^{n_0-k} (e_i^1 - e_i^2) \leq \sum_{i=1}^{n_0-k} b_i < \sum_{i=n_0-k+1}^{n_0} (e_i^1 + e_i^2)$$

$$= \sum_{i=n_0-k+1}^{n_0} (2b_1 + 2b_2 + \cdots + 2b_{i-1} + b_i + 2i),$$

$$k = 1, \ldots, n_0 - 1.$$

Таким образом, только n_0 элементов, выбранных из пар, могут дать общую сумму весов, равную E.

Кроме того,

$$\sum_{a_i \in A_1} e_i = \sum_{i=1}^{n_0} e_{2i-1} + \sum_{b_i \in B_1} b_i = \sum_{i=1}^{n_0} e_{2i-1} + \sum_{b_i \in B_2} b_i = \sum_{a_i \in A_2} e_i$$

тогда и только тогда, когда

$$\sum_{b_i \in B_1} b_i = \sum_{b_i \in B_2} b_i.$$

Будем рассматривать такой NP-полный случай в наших последующих сведениях. □

Пусть $r_{\min} = \min_{j \in J} r_j$ и $d_{\max} = \max_{j \in J} d_j$ или $d_{\max}$ равен некоторой иной верхней оценке длины расписания. Будем говорить, что расписание называется расписанием без простоев, если машина не простаивает в интервале $[r_{\min}, d_{\max}]$. Индивидуальная задача обладает свойством без простоев, если не существует допустимого расписания, которое не обладает свойством без простоев. Доказательства NP-трудности будут основаны на полиномиальном сведении задачи Упорядоченное Разбиение к распознавательной версии задачи составления расписаний с запретами работ. Полученное семейство задач будет обладать свойством без простоев. Кроме того, основные работы j будут соответствовать элементам a_j, а подмножества работ X^i будут содержать пары $a_{2i-1},\ a_{2i}$, но в некоторых случаях могут быть введены также вспомогательные работы. В то же время каждое сведение будет уникальным ввиду свойств целевой функции и дополнительных ограничений.

Теорема 39 *Задача* $1|d_j = d|\sum_j w_j U_j$ *с запретами работ является NP-трудной даже в случае системы подмножеств* $X = \{X^i\}$, *где* $|X^i| = 2,\ i = 1, \ldots, n$.

Доказательство. Пусть количество работ $n = 2n_0$. Положим $p_j = w_j = e_j$ и $d_j = E$ для работы $j \in \mathcal{J}$. Определим подмножества работ как $X^{i+n_0} = X^i = \{2i-1,\ 2i\},\ i = 1, \ldots, n_0$.

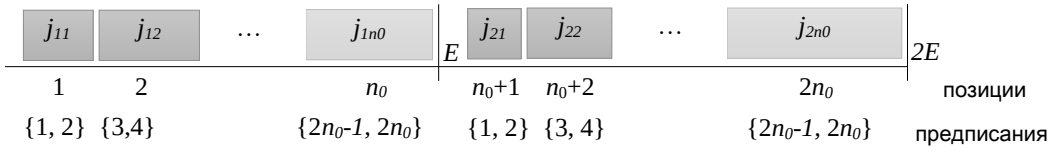


Рисунок 16 — Иллюстративный пример сведения задачи Упорядоченное Разбиение к $1|d_j = d|\sum_j w_j U_j$ с запретами работ. Здесь представлена схема расписания без простоев, в котором работы из $\{j_{11}, \ldots, j_{1n_0}\}$ завершаются до момента времени E, работы из $\{j_{21}, \ldots, j_{2n_0}\}$ начинаются после момента времени E, и ограничения по запретам работ выполнены. Такое расписание может быть построено тогда и только тогда, когда соответствующая индивидуальная задача Упорядоченное Разбиение имеет ответ «да». Только расписания такой структуры имеют значение целевой функции $\sum_j w_j U_j \leq E$, остальные расписания имеют большие значения целевой функции.

Перестановка π', удовлетворяющая ограничениям на запреты работ и имеющая $\sum_j w_j U_j(\pi') \leq E$, существует тогда и только тогда, когда задача Упорядоченногое Разбиение имеет ответ «да» (см. рис. 16). Действительно, общий директивный срок E для всех работ гарантирует, что $\sum_j w_j U_j(\pi') \leq E$ может выполняться только как равенство. Кроме того, подмножество J_1 из n_0 работ (содержащее по одной работе из каждой пары $2i-1,\ 2i$) с общей длительностью E, завершённое до директивного срока, может существовать только в случае, когда у нас есть два подмножества A_1, A_2 с суммой весов, равной E, в задаче Упорядоченногое Разбиение (элементы из A_1 соответствуют работам из J_1). □

Теорема 40 *Задача* $1|r_j|\gamma$ *с запретами работ является NP-трудной даже в случае системы подмножеств* $X = \{X^i\}$, *такой что* $|X^i| \leq 2,\ i = 1, \ldots, n$, *для* $\gamma \in \{L_{\max}; \sum_j U_j; \sum_j T_j\}$.

Доказательство. Положим количество работ $n = 2n_0 + 1$. Пусть $p_j = e_j$, $d_j = 2E + 1$, $r_j = 0$ для работ $j = 1, \ldots, 2n_0$. Положим $p_{2n_0+1} = 1$, $r_{2n_0+1} = E$ и директивный срок $d_{2n_0+1} = E + 1$ для работы $2n_0 + 1$. Определим систему X подмножеств работ как $X^{i+n_0+1} = X^i = \{2i - 1,\ 2i\},\ i = 1, \ldots, n_0$ и $X^{n_0+1} = \{2n_0 + 1\}$.

Предположим, что осуществляется поиск перестановки π', удовлетворяющей ограничениям на запреты работ и имеющей целевую функцию

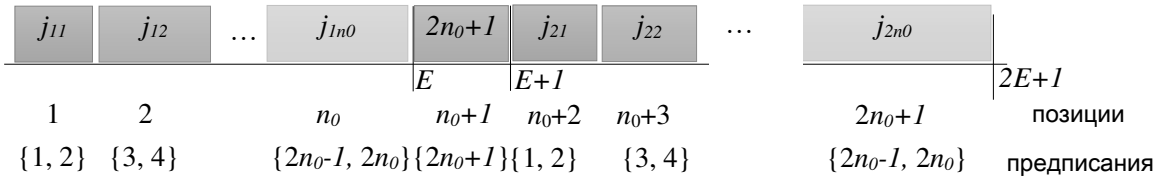


Рисунок 17 — Иллюстративный пример сведения задачи Упорядоченное Разбиение к $1|r_j|\gamma$, $\gamma \in \{L_{\max}; \sum_j U_j; \sum_j T_j\}$ и $1|C_j \le d_j| \sum_j w_j C_j$ с запретами работ. Здесь представлена схема расписания без простоев, где работа $(n_0 + 1)$ выполняется в интервале $[E, E + 1)$, работы из $\{j_{11}, \ldots, j_{1n_0}\}$ завершаются до момента времени E, работы из $\{j_{21}, \ldots, j_{2n_0}\}$ начинаются после момента времени $E + 1$ и завершаются до момента времени $2E+1$, и все работы поставлены в расписание в соответствии с системой подмножеств работ X. Эти расписания существуют тогда и только тогда, когда соответствующая индивидуальная задача Упорядоченное Разбиение имеет ответ «да». Только расписания такой структуры имеют значение целевой функции меньше заданной верхней границы.

$L_{\max}(\pi') \le 0$ ($\sum_j U_j(\pi') \le 0$ или $\sum_j T_j(\pi') \le 0$). Работа $2n_0 + 1$ поступает в систему в момент времени E и должна быть завершена к моменту $E+1$. Кроме того, остальные работы должны быть завершены до момента $2E + 1$. Это возможно тогда и только тогда, когда элементы в задаче Упорядоченное Разбиение разбиты на две части A_1, A_2 с равными суммарными весами (см. рис. 17). □

Теорема 41 *Задача* $1|C_j \le d_j| \sum_j w_j C_j$ *с запретами работ является NP-трудной даже в случае системы подмножеств* $X = \{X^i\}$, *где* $|X^i| \le 2,\ i = 1, \ldots, n,$.

Доказательство. Предположим, что количество работ $n = 2n_0+1$. Пусть $p_j = w_j = e_j$, $d_j = 2E + 1$ для работ $j = 1, \ldots, 2n_0$. Для работы $2n_0 + 1$ положим $p_{2n_0+1} = 1$, $w_{2n_0+1} = 0$ и $d_{2n_0+1} = E + 1$.

Определим систему подмножеств работ как $X^{i+n_0+1} = X^i = \{2i - 1,\ 2i\},\ i = 1, \ldots, n_0$ и $X^{n_0+1} = \{2n_0 + 1\}$.

Перестановка π', удовлетворяющая ограничениям на запреты работ и имеющая

$$\sum_j w_j C_j(\pi') \le \sum_{1 \le i \le j \le 2n_0} e_i e_j + E,$$

существует тогда и только тогда, когда задача Упорядоченное Разбиение имеет положительный ответ (см. рис. 17). Вес равен длительности для работ $i = 1, \ldots, 2n_0$. Таким образом, если работа $2n_0 + 1$ отсутствует, то $\sum_{j=1}^{2n_0} w_j C_j = \sum_{1 \le i \le j \le 2n_0} e_i e_j$ при любом допустимом упорядочении первых $2n_0$ работ. Работа $2n_0 + 1$ с длительностью 1 должна быть завершена до момента $E + 1$, следовательно, суммарное взвешенное время завершения $\sum_{j=1}^{2n_0+1} w_j C_j = \sum_{1 \le i \le j \le 2n_0} e_i e_j + \sum_{j \in J_2} e_j$, где J_2 – подмножество работ, выполняемых после работы $2n_0+1$. В результате неравенство $\sum_{j=1}^{2n_0+1} w_j C_j(\pi') \le \sum_{1 \le i \le j \le 2n_0} e_i e_j + E$ выполняется тогда и только тогда, когда подмножество J_2 содержит работы с общим весом E, и, следовательно, тогда и только тогда, когда задача Упорядоченное Разбиение с подмножествами A_1, A_2 имеет ответ «да». □

Теорема 42 *Задача* $1|r_j|\sum_j C_j$ *с запретами работ является NP-трудной даже в случае системы подмножеств* $X = \{X^i\}$, *такой что* $|X^i| \le 2,\ i = 1, \ldots, n$.

Доказательство. Предположим, что у нас есть $n = 2n_0 + 2$ работ. Пусть $p_j = e_j$, $r_j = 0$ для работ $j = 1, \ldots, 2n_0$, и $p_{2n_0+1} = p_{2n_0+2} = 1$, $r_{2n_0+1} = E,\ r_{2n_0+2} = 2E + 1$ для работ $2n_0 + 1,\ 2n_0 + 2$. Определим подмножества работ в системе X как $X^{i+n_0+1} = X^i = \{2i - 1,\ 2i\},\ i = 1, \ldots, n_0$ и $X^{n_0+1} = \{2n_0 + 1\}$, $X^{2n_0+2} = \{2n_0 + 2\}$.

Покажем, что существует перестановка π', удовлетворяющая ограничениям на запреты работ и имеющая

$$\sum_j C_j(\pi') \le L := (E + 1) + (2E + 2) +$$

$$\left(\sum_{j=1}^{n_0} (n_0 - j + 1)(e_{2j-1} + e_{2j}) + (E + 1)n_0 \right),$$

тогда и только тогда, когда задача Упорядоченное Разбиение имеет ответ «да» (см. рис. 18).

Пусть задача Упорядоченное Разбиение имеет ответ «да», и A_1 и A_2 – подмножества A такие, что $\sum_{a_i \in A_1} e_i = \sum_{a_i \in A_2} e_i = E,\ |A_1| = |A_2| = n_0$. Обозначим через $\pi(A_1)$ и $\pi(A_2)$ перестановки работ, соответствующие

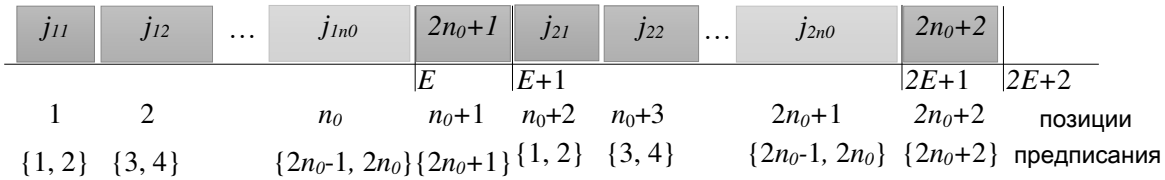


Рисунок 18 — Иллюстративный пример сведения задачи Упорядоченное Разбиение к $1|r_j|\sum_j C_j$ с запретами работ. Здесь представлена схема расписания без простоев, где работа $2n_0+1$ выполняется в интервале $[E, E+1)$, работа $2n_0+2$ выполняется в интервале $[2E+1, 2E+2)$, а остальные работы разбиты на две части $J_1 = \{j_{11}, \ldots, j_{1n_0}\}$, $J_2 = \{j_{21}, \ldots, j_{2n_0}\}$, удовлетворяющие ограничениям на запреты работ и имеющие общую длительность, равную E. Такое расписание может быть построено тогда и только тогда, когда соответствующая индивидуальная задача Упорядоченное Разбиение имеет ответ «да». Только расписания такой структуры имеют значение целевой функции $\sum_j C_j \leq L$, остальные расписания имеют большие значения целевой функции.

элементам из A_1 и A_2, где работы упорядочены по возрастанию их номеров. Тогда имеем $\sum_j C_j(\pi') = L$ для $\pi' = (\pi(A_1), 2n_0+1, \pi(A_2), 2n_0+2)$.

Пусть π' — перестановка, удовлетворяющая ограничениям на запреты работ и $\sum_j C_j(\pi') \leq L$. В этой перестановке работа $2n_0+1$ выполняется в позиции n_0+1, а последняя позиция содержит работу $2n_0+2$. Обозначим через J_1 подмножество работ, выполняемых до работы $2n_0+1$, а через J_2 – подмножество работ, выполняемых после работы $2n_0+1$ и до работы $2n_0+2$ ($|J_1| = |J_2| = n_0$). Требуется доказать, что общая длительность работ из J_1 равна E. Предположим, что $\sum_{j\in J_1} p_j < E$, тогда $\sum_{j\in J_2} p_j > E$ и $\sum_j C_j(\pi') \geq L+1$, так как работа $2n_0+2$ завершается в момент $2E+3$ или позже. Если общая длительность работ из J_1 больше E, то $\sum_j C_j(\pi') \geq L+1+(E+2)n_0-(E+1)n_0 = L+1+n_0$, поскольку работы из J_2 начинают выполняться после момента времени $(E+2)$, а работа $2n_0+1$ начинается не раньше момента времени $(E+1)$. Следовательно, $\sum_{j\in J_1} p_j = \sum_{j\in J_2} p_j = E$, и подмножества $A_1 = \{a_j : j \in J_1\}$, $A_2 = \{a_j : j \in J_2\}$ удовлетворяют условиям задачи Упорядоченное Разбиение. □

Теорема 43 *Задачи* $2||C_{\max}$, $2|d_j = d|\gamma$, $\gamma \in \{L_{\max}, \sum_j U_j, \sum_j T_j\}$, *с запретами работ являются NP-трудными для систем подмножеств*

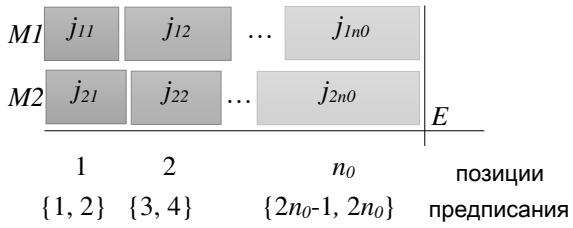


Рисунок 19 — Иллюстративный пример сведения задачи Упорядоченное Разбиение к $2||C_{\max},\ 2|d_j = d|\gamma,\ \gamma \in \{L_{\max}, \sum_j U_j, \sum_j T_j\}$, с запретами работ. Здесь представлена схема расписания без простоев, в котором последняя работа на обеих машинах завершается в момент E, и ограничения на запреты работ выполнены. Такое расписание может быть построено тогда и только тогда, когда соответствующая индивидуальная задача Упорядоченное Разбиение имеет ответ «да». Только расписания такой структуры (подмножества работ $J_1 = \{j_{11}, \ldots, j_{1n_0}\}$ и $J_2 = \{j_{21}, \ldots, j_{2n_0}\}$, выполняемые на машинах $M1$ и $M2$) имеют значение целевой функции не превосходящие заданное пороговое значение E.

работ $X = \{X^{i,l}\},\ |X^{i,l}| = 2,\ i \in N_l,\ l = 1, 2.$

Доказательство. Рассмотрим семейство индивидуальных задач составления расписаний с $n = 2n_0$ работами. Пусть длительности $p_j = e_j$ для работ $j = 1, \ldots, 2n_0$.

Определим систему X подмножеств работ как $X^{i,1} = X^{i,2} = \{2i - 1,\ 2i\},\ i = 1, \ldots, n_0$ и $X^{i,1} = X^{i,2} = \emptyset,\ i = 1 + n_0, \ldots, 2n_0$.

Легко видеть, что семейство перестановок $\{\pi'^l\}_{l \in M}$, удовлетворяющее ограничениям на запреты работ и определяющее длину расписания $C_{\max} \leq E$, существует тогда и только тогда, когда задача Упорядоченное Разбиение имеет ответ «да» (см. Рис. 19). Результат следует из свойства, что все работы должны быть завершены до момента времени E. Это возможно только в случае, когда элементы из A могут быть разделены на две части A_1, A_2 с равным общим весом в задаче Упорядоченное Разбиение. Для критериев $L_{\max}, \sum_j U_j, \sum_j T_j$ используется та же схема сведения, дополнительно полагая $r_j = 0,\ d_j = E$ и пороговое значение для целевой вункции, равное нулю. □

Также имеют место теоремы

Теорема 44 *Задача* $1|l_{ij}|C_{\max}$ *с запретами работ является NP-трудной*

для систем подмножеств работ $X = \{X^i\},\ |X^i| \le 2,\ i = 1, \ldots, n.$

Теорема 45 *Задача* $1||\sum_j w_j T_j$ *с запретами работ является NP-трудной для систем подмножеств работ* $X = \{X^i\},\ |X^i| \le 2,\ i = 1, \ldots, n.$

NP-трудность задач оптимальной рекомбинации для указанных задач была доказана в [98,248]. NP-трудность соответствующих задач составления расписаний с запретами работ доказывается аналогичным образом. В первом случае используется полиномиальное сведение задачи поиска максимальной клики минимального веса специального вида, как в работе Сердюкова А.И. [11]. Во втором случае для доказательства NP-трудности используется сведение специального случая задачи Разбиение [91].

Теорема 46 *Задача* $1|r_j = 0, d_j, w_j|\sum_j w_j T_j$ *с запретами работ является NP-трудной для систем подмножеств работ* $X = \{X^i\},\ |X^i| \le 2,\ i = 1, \ldots, n,$ *даже в случае, когда все работы, кроме двух, имеют одинаковые директивные сроки.*

Доказательство. Пусть количество работ $n = 2n_0 + 2$. Работы $j \in \{1, \ldots, 2n_0\}$ имеют длительности выполнения $p_j = e_j$, веса $w_j = e_j$ и директивные сроки $d_j = 0$. Две последние работы имеют длительность $p_{2n_0+1} = p_{2n_0+2} = 3E^2 + 1$, вес $w_{2n_0+1} = w_{2n_0+2} = 3(3E^2 + 1 + E)E + 1$, директивные сроки $d_{2n_0+1} = 3E^2 + 1$ и $d_{2n_0+2} = 2(3E^2 + 1) + E$.

Определим систему подмножеств работ X как $X^i = \{2i - 1,\ 2i\},\ i = 2, \ldots, n_0 + 1,\ n_0 + 3, \ldots, 2n_0 + 2$ и $X^1 = \{2n_0 + 1\},\ X^{n_0+2} = \{2n_0 + 2\}$.

Покажем, что перестановка π', которая согласуется с X и имеет

$$\sum_{j=1}^{n} w_{\pi'_j} T(\pi'_j) \le 3(3E^2 + 1 + E)E,$$

существует тогда и только тогда, когда задача Упорядоченное Разбиение имеет ответ «да» (см. Рис. 20).

Пусть задача Упорядоченное Разбиение имеет ответ «да» и подмножества A_1 и A_2 таковы, что $\sum_{a_i \in A_1} e_i = \sum_{a_i \in A_2} e_i = E,\ |A_1| = |A_2| = n_0$. Обозначим через $\pi(A_1)$ и $\pi(A_2)$ перестановки работ в порядке, соответствующем возрастанию их номеров из A_1 и A_2 соответственно. Тогда для

перестановки $\pi' = (2n_0 + 1, \pi(A_1), 2n_0 + 2, \pi(A_2))$ имеем

$$\sum_{j=1}^{n} w_{\pi'_j} T(\pi'_j) \leq (3E^2 + 1 + E)E + 2(3E^2 + 1 + E)E = 3(3E^2 + 1 + E)E,$$

так как все работы из $\pi(A_1)$ имеют время завершения не более $(3E^2+1+E)$, а работы из $\pi(A_2)$ завершаются до момента времени $2(3E^2 + 1 + E)$. Более того, суммарный вес работ из $\pi(A_1)$ (аналогично из $\pi(A_2)$) равен E, а работы $2n_0 + 1$ и $2n_0 + 2$ выполняются до директивного срока.

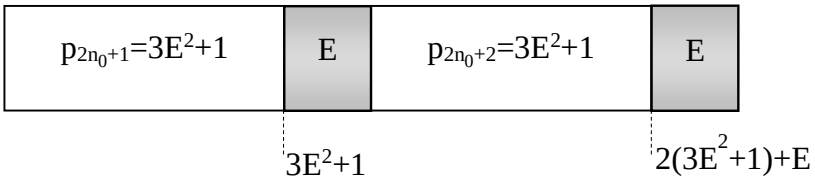


Рисунок 20 — Иллюстрация допустимого расписания для $1|r_j = 0, d_j, w_j| \sum_j w_j T_j$ с запретами работ.

Пусть перестановка π' согласуется с X и $\sum_{j=1}^{n} w_{\pi'_j} T(\pi'_j) \leq 3(3E^2 + 1 + E)E$. Тогда работы $2n_0 + 1$ и $2n_0 + 2$ выполняются до директивного срока, так как их веса больше $3(3E^2 + 1 + E)E$. Обозначим через W сумму весов работ, выполняемых после работы $2n_0 + 2$. Тогда

$$\sum_{j} w_j T_j(\pi') \geq (3E^2 + 1)(2E - W) + 2(3E^2 + 1)W = (3E^2 + 1)(2E + W).$$

Очевидно, что $W \geq E$, так как работа $2n_0 + 2$ должна быть завершена до момента времени $2(3E^2+1)+E$. Предположим, что $W \geq E+1$, тогда $\sum_{j=1}^{n} w_{\pi'_j} T(\pi'_j) \geq (3E^2 + 1)(2E + E + 1) = 3(3E^2 + 1)E + (3E^2 + 1) > 3(3E^2+1+E)E$, что приводит к противоречию. Следовательно, n_0 работ с суммарным весом E выполняются между работами $2n_0 + 1$ и $2n_0 + 2$. Таким образом, задача Упорядоченное Разбиение имеет ответ «да». □

Для доказательства NP-трудности задачи $1|r_j = 0, d_j| \sum_j T_j$ с запретами работ будем использовать следующий вариант задачи *Упорядоченное Разбиение* [91]: дано упорядоченное множество $A = \{a_1, \ldots, a_{2n_0}\}$ и

вес e_i каждого элемента $a_i \in A$, где

$$\sum_{a_i \in A} e_i = 2E,\ e_i > e_{i+1},\ i = 1, \ldots, 2n_0 - 1, \tag{163}$$

$$e_{2j} > e_{2j+1} + \delta,\ e_{2j-1} \leq e_{2j} + \delta, j = 1, \ldots, n_0. \tag{164}$$

Здесь $\delta = \frac{1}{2}\sum_{i=1}^{n_0}(e_{2i-1} - e_{2i})$. Заметим, что $E = \frac{1}{2}\sum_{i=1}^{2n_0} e_i = \sum_{i=1}^{n_0} e_{2i-1} - \delta = \sum_{i=1}^{n_0} e_{2i} + \delta$. Вопрос заключается в том, можно ли разбить множество A на два подмножества A_1 и A_2 так, что

$$\sum_{a_i \in A_1} e_i = \sum_{a_i \in A_2} e_i = E,\ |A_1| = |A_2| = n_0$$

и подмножество A_1 содержит только один элемент из каждой пары a_{2i-1}, a_{2i}.

Теорема 47 *Задача* $1|r_j = 0, d_j|\sum_j T_j$ *с запретами работ является NP-трудной для систем подмножеств работ* $X = \{X^i\}$, $|X^i| \leq 2$, $i = 1, \ldots, n$.

Доказательство. Пусть количество работ $n = 3n_0 + 1$. Работы $j \in \{1, \ldots, 2n_0\}$ будем называть S-работами: $S_1, \ldots, S_{2n_0}$, а остальные $(n_0 + 1)$ работ - B-работами: $B_1, \ldots, B_{n_0}, B_{n_0+1}$. Обозначим через b величину $(4n_0 + 1)\delta$.

Длительности выполнения S-работ равны $p(S_i) = e_i$, $i = 1, \ldots, 2n_0$, а длительности B-работ $p(B_i) = b$, $i = 1, \ldots, n_0 + 1$. Зададим директивные сроки работ по правилу (см. рис. 21)

$$d(S_i) = \begin{cases} (j-1)b + \delta + (e_2 + e_4 + \cdots + e_{2j}), & \text{если } i = 2j - 1, \\ d(S_{2j-1}) + 2(n_0 - j + 1)(e_{2j-1} - e_{2j}), & \text{если } i = 2j, \end{cases}$$

$j = 1, \ldots, n_0$.

$$d(B_i) = \begin{cases} ib + (e_2 + e_4 + \cdots + e_{2i}), & \text{если } i = 1, \ldots, n_0, \\ d(B_{n_0}) + \delta + b, & \text{если } i = n_0 + 1. \end{cases}$$

Определим систему подмножеств работ X как
$X^{2i-1} = \{S_{2i-1},\ S_{2i}\},\ i = 1, \ldots, n_0$;

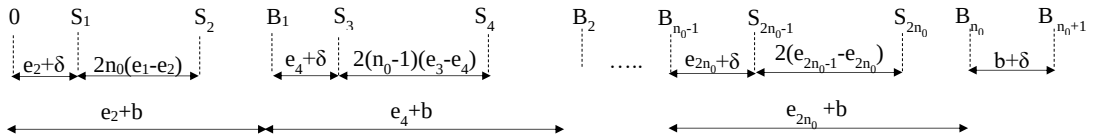


Рисунок 21 — Иллюстрация директивных сроков для задачи составления расписаний на одной машине $1|r_j = 0, d_j| \sum_j T_j$ с запретами работ.

$X^{2i} = \{B_i\},\ i = 1, \ldots, n_0; X^{2n_0+1} = \{B_{n_0+1}\};$
$X^{3n_0+2-i} = \{S_{2i-1},\ S_{2i}\},\ i = 1, \ldots, n_0.$

Покажем, что существует перестановка π', согласующаяся с системой подмножеств X и соответствующая $\sum_{j=1}^{n} T(\pi'_j) \leq T_0$, тогда и только тогда, когда задача Упорядоченное Разбиение имеет ответ «да». Здесь $T_0 := E + n_0 P - n_0(n_0 - 1)b/2 - n_0\delta - \sum_{i=1}^{n_0}(n_0 - i + 1)(e_{2i-1} + e_{2i})$, где $P := (n_0 + 1)b + 2E$ – суммарная длительность всех работ.

Начнём доказательство с того, что неравенство

$$\sum_{j=1}^{n} T(\pi'_j) \geq T_0$$

выполняется для перестановки
$\pi' := (S_{1,1}, B_1, S_{2,1}, B_2, \ldots, B_{n_0-1}, S_{n_0,1}, B_{n_0}, B_{n_0+1}, S_{n_0,2}, S_{n_0-1,2}, \ldots, S_{1,2})$
(здесь $\{S_{i,1},\ S_{i,2}\} = \{S_{2i-1},\ S_{2i}\}$). Более того, $\sum_{j=1}^{n} T(\pi'_j) = T_0$ тогда и только тогда, когда $\sum_{i=1}^{n_0} p(S_{i,1}) = \sum_{i=1}^{n_0} p(S_{i,2})$.

Вычислим суммарное запаздывание для работ $B_i,\ i = 1, \ldots, n_0$. Поскольку $e_{2j-1} > e_{2j}$, то $C(B_i) \geq d(B_i)$ для моментов завершения работ $B_i,\ i = 1, \ldots, n_0$. Кроме того, $C(B_i) = ib + \sum_{j=1}^{i} p(S_{i,1})$, $\sum_{i=1}^{n_0} d(B_i) = \frac{n_0(n_0+1)}{2} b + \sum_{i=1}^{n_0}(n_0 - i + 1)e_{2i}$. Следовательно,

$$\sum_{i=1}^{n_0} (C(B_i) - d(B_i)) = \sum_{i=1}^{n_0} C(B_i) - \sum_{i=1}^{n_0} d(B_i) =$$

$$\frac{n_0(n_0+1)}{2} b + \sum_{i=1}^{n_0}(n_0 - i + 1)p(S_{i,1}) - \frac{n_0(n_0+1)}{2} b - \sum_{i=1}^{n_0}(n_0 - i + 1)e_{2i} =$$

$$\sum_{i=1}^{n_0}(n_0-i+1)\left(p(S_{i,1})-e_{2i}\right).$$

Теперь вычислим суммарное запаздывание для работ $S_{i,2},\ i=1,\ldots,n_0$. Заметим, что для работ $S_{i,2},\ i=1,\ldots,n_0$, выполняется $C(S_{i,2})\geq d(S_{i,2})$, поскольку $C(S_{i,2})\geq (n_0+1)b+\sum_{j=1}^{n_0}a_{2j}$, но $d(S_{i,2})\leq (n_0-1)b+(2n_0+1)\delta+\sum_{j=1}^{n_0}a_{2j}$ из определения директивных сроков и длительностей выполнения. Более того, $C(S_{i,2})=P-\sum_{j=1}^{i-1}p(S_{j,2})$. Если $S_{i,2}=S_{2i-1}$, то $d(S_{i,2})=(i-1)b+\delta+\sum_{j=1}^{i}e_{2j}$, а если $S_{i,2}=S_{2i}$, то $d(S_{i,2})=(i-1)b+\delta+\sum_{j=1}^{i}e_{2j}+2(n_0-i+1)(e_{2i-1}-e_{2i})$. Следовательно, $d(S_{i,2})=(i-1)b+\delta+\sum_{j=1}^{i}e_{2j}+(n_0-i+1)(e_{2i-1}-e_{2i})+(n_0-i+1)\left(p(S_{i,1})-p(S_{i,2})\right)$. Таким образом,

$$\sum_{i=1}^{n_0}\left(C(S_{i,2})-d(S_{i,2})\right)=$$

$$n_0P-\sum_{i=1}^{n_0-1}(n_0-i)p(S_{i,2})-\frac{n_0(n_0-1)}{2}b-n_0\delta-$$

$$\sum_{i=1}^{n_0}(n_0-i+1)e_{2i-1}-\sum_{i=1}^{n_0}(n_0-i+1)\left(p(S_{i,1})-p(S_{i,2})\right)=$$

$$n_0P-\frac{n_0(n_0-1)}{2}b-n_0\delta-\sum_{i=1}^{n_0}(n_0-i+1)e_{2i-1}-\sum_{i=1}^{n_0}(n_0-i+1)p(S_{i,1})+\sum_{i=1}^{n_0}p(S_{i,2}).$$

Суммарное запаздывание $\sum_{j=1}^{n}T(\pi'_j)$ по всем работам равно

$$n_0P-\frac{n_0(n_0-1)}{2}b-n_0\delta-\sum_{i=1}^{n_0}(n_0-i+1)\left(e_{2i-1}+e_{2i}\right)+$$

$$\sum_{i=1}^{n_0}p(S_{i,2})+\sum_{i=1}^{n_0}T(S_{i,1})+T(B_{n_0+1}).$$

Если $\sum_{i=1}^{n_0}p(S_{i,1})\leq E=\sum_{i=1}^{n_0}e_{2i}+\delta$, то для любого $k=1,\ldots,n_0$ выполняется $\sum_{i=1}^{k}p(S_{i,1})\leq\sum_{i=1}^{k}e_{2i}+\delta$ (доказывается от противного). Из определений значений $d(S_i)$ и $d(B_{n_0+1})$ получаем $C(S_{i,1})\leq d(S_{i,1})$ и $T(S_{i,1})=0$; $C(B_{n_0+1})\leq d(B_{n_0+1})$ и $T(B_{n_0+1})=0$. Из определения T_0 и выражения для $\sum_{j=1}^{n}T(\pi'_j)$ делаем вывод, что $\sum_{i=1}^{n_0}p(S_{i,2})\geq E$ и

$\sum_{j=1}^{n} T(\pi'_j) \geq T_0$. Более того, равенство возможно только когда

$$\sum_{i=1}^{n_0} p(S_{i,1}) = \sum_{i=1}^{n_0} p(S_{i,2}) = E.$$

Если $\sum_{i=1}^{n_0} p(S_{i,1}) > E$, то

$$T(B_{n_0+1}) = \sum_{i=1}^{n_0} p(S_{i,1}) + (n_0+1)b - \left((n_0+1)b + \delta + \sum_{i=1}^{n_0} e_{2i} \right)$$

$$= \sum_{i=1}^{n_0} p(S_{i,1}) - E.$$

Более того, можно найти индекс l такой, что $\sum_{i=1}^{l} p(S_{i,1}) > \sum_{i=1}^{l} e_{2i} + \delta$. Пусть k – наименьший из таких индексов, тогда $S_{k,1} = S_{2k-1}$, $C(S_{k,1}) > d(S_{k,1}) = \sum_{i=1}^{k} e_{2i} + \delta + (k-1)b$ по определению и $T(S_{k,1}) > 0$. В результате имеем

$$\sum_{j=1}^{n} T(\pi'_j) \geq n_0 P - \frac{n_0(n_0-1)}{2} b - n_0\delta - \sum_{i=1}^{n_0} (n_0 - i + 1)\left(e_{2i-1} + e_{2i}\right) +$$

$$\left(\sum_{i=1}^{n_0} p(S_{i,2}) + \sum_{i=1}^{n_0} p(S_{i,1}) - E \right) + T(S_{k,1}) > T_0.$$

Таким образом, перестановка π', согласующаяся с системой подмножеств X и имеющая $\sum_{j=1}^{n} T(\pi'_j) \leq T_0$, существует тогда и только тогда, когда задача Упорядоченное Разбиение имеет положительный ответ. □

Теорема 48 *Задача $1|r_j, d_j|\sum_j T_j$ с запретами работ является NP-трудной для систем подмножеств работ $X = \{X^i\}$, $|X^i| \leq 2$, $i = 1, \ldots, n$ даже в случае, когда имеется только одна работа с индивидуальным моментом поступления и директивным сроком.*

Доказательство аналогично доказательству NP-трудности задачи $1|r_j, d_j|\sum_j T_j$ (см. теорему 4 в [170]).

3.1.2 Подход к решению

Рассмотрим индивидуальную задачу $I(\gamma,\ X^i)$ массовой задачи составления расписаний на одном процессоре с целевой функцией $\gamma(\cdot)$ и системой подмножеств работ $X = \{X^i\}_{i=1,\dots,n}$, где $|X^i| \leq 2$.

Построим двудольный граф $G = (X_n, X, U)$ с двумя долями множества вершин X_n, X равных мощностей и множеством рёбер $U = \{\{i, x\} : i \in X_n,\ x \in X^i\}$. Левая доля вершин соответствует позициям, а правая — работам. По построению видно, что существует взаимно однозначное соответствие между совершенными паросочетаниями G и допустимыми перестановками задачи $I(\gamma,\ X^i)$. Напомним, что множество рёбер совершенного паросочетания «покрывает» все вершины графа. Будем обозначать множество совершенных паросочетаний через $\mathcal{W}$, а множество допустимых перестановок для $I(\gamma,\ X^i)$ через Π.

Согласно терминологии из [11], имеем следующие типы рёбер в графе G. *Особые* рёбра принадлежат всем совершенным паросочетаниям в G, а рёбра, смежные с ними, отсутствуют во всех совершенных паросочетаниях G. Последние могут быть найдены с помощью Алгоритма $\mathcal{A}$.

Алгоритм $\mathcal{A}$.

1. Построим вспомогательный граф $G' := G$, множество специальных рёбер $U' := \emptyset$.

2. Повторять, пока условие завершения не выполнится или G' не станет 2-регулярным:

2.1. Если G' содержит вершину z степени 0, то задача неразрешима, останавливаемся.

2.2. Если G' содержит вершину z степени 1, то $U' := U' \cup \{z, y\}$ и удалить $y,\ z$ из G'.

3. $G' := G' \cup U'$.

Общее количество рёбер в G и G' ограничено $2n$, и каждое ребро посещается только один раз, поэтому время работы алгоритма $\mathcal{A}$ составляет $O(n)$. Кроме того, с помощью $\mathcal{A}$ идентифицируется случай, когда задача неразрешима. Таким образом, далее будем исследовать только задачи с непустым множеством допустимых решений и положим $G := G'$.

Рёбра G, не являющиеся особыми, разбиваются на чётные циклы (см. рис. 22). Для нахождения циклов в G можно использовать алгоритм

поиска в глубину [79], трудоемкость которого составляет $O(n)$. Пусть $q(G) = q(I)$ обозначает количество циклов в графе G, построенном для задачи $I(\gamma,\ X^i)$.

Цикл $j,\ j = 1, \ldots, q(G)$, имеет два совершенных паросочетания. А совершенное паросочетание в G состоит из совершенных паросочетаний циклов и особых рёбер (см. рис. 22).

Предложенный алгоритм основан на тех же принципах, что и другие алгоритмы перечисления совершенных паросочетаний в двудольных графах (см., например, [232]), но имеет меньшее время работы благодаря свойству, что вершины левой доли имеют степень не более двух.

Таким образом, задача составления расписаний с запретами работ $I(\gamma,\ X^i)$ может быть решена следующим образом. Перебираем все совершенные паросочетания $W \in \mathcal{W}$, строя соответствующую перестановку $\pi \in \Pi$ для каждого $W \in \mathcal{W}$ и вычисляя $\gamma(\pi)$. Перестановка с оптимальным значением целевой функции соответствует оптимальному решению. Заметим, что $|\Pi| = |\mathcal{W}| = 2^{q(G)} = 2^{q(I)}$. Предложенный алгоритм имеет время работы $O(n+T(\gamma)2^{q(I)})$, где $T(\gamma)$ — время вычисления целевой функции γ, $q(I) \leq \lfloor \frac{n}{2} \rfloor$. Таким образом, получаем

Теорема 49 *Задачи составления расписаний* $1|r_j, d_j|\gamma$, $\gamma \in \{C_{\max}, L_{\max}, \sum_j w_j C_j,\ \sum_j w_j T_j, \sum_j w_j U_j\}$, *с запретами работ могут быть решены за время* $O(n1.42^n)$ *для системы подмножеств работ* $X = \{X^i\}_{i=1,\ldots,n}$, *такой что* $|X^i| \leq 2,\ i = 1, \ldots, n$.

Согласно терминологии из [11], двудольный граф $G = (X_n, X, U)$ называется «хорошим», если количество циклов $q(G)$ ограничено величиной $1.1\ln n$. Обозначим через $\bar{\chi}_n$ множество «хороших» двудольных графов вида $G = (X_n, X, U)$, а через χ_n – множество всех двудольных графов вида $G = (X_n, X, U)$, где степени вершин левой доли не превосходят двух.

Теорема 50 $\frac{|\bar{\chi}_n|}{|\chi_n|} \to 1$ *при* $n \to \infty$ *[11, 98].*

В результате заключаем, что почти все системы подмножеств работ с $|X^i| \leq 2,\ i = 1, \ldots, n$, приводят к индивидуальным задачам $I(\gamma,\ X^i)$,

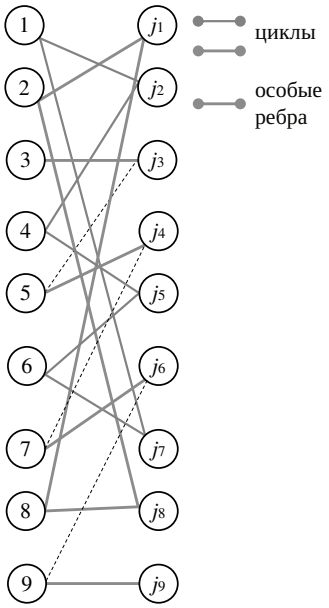


Рисунок 22 — Индивидуальная задача с $n = 9$ работами и системой подмножеств работ X: $X^1 = \{j_2, j_7\}$, $X^2 = \{j_1, j_8\}$, $X^3 = \{j_3\}$, $X^4 = \{j_2, j_5\}$, $X^5 = \{j_3, j_4\}$, $X^6 = \{j_5, j_7\}$, $X^7 = \{j_4, j_6\}$, $X^8 = \{j_1, j_8\}$, $X^9 = \{j_6, j_9\}$. Соответствующий двудольный граф $G = (X_9, X, U)$ имеет представленный вид и содержит два цикла и четыре особых ребра. Задача имеет четыре допустимых решения: $(j_2, j_1, j_3, j_5, j_4, j_7, j_6, j_8, j_9)$, $(j_2, j_8, j_3, j_5, j_4, j_7, j_6, j_1, j_9)$, $(j_7, j_1, j_3, j_2, j_8, j_6, j_9, j_5, j_4)$, $(j_7, j_8, j_3, j_2, j_5, j_6, j_9, j_1, j_4)$.

имеющим не более n допустимых решений и, следовательно, разрешимы за время $O(n^2)$.

Заметим, что вместо константы, равной 1.1 в определении «хорошего» графа, можно было бы выбрать любую другую константу, равную $1+\varepsilon$, где $\varepsilon \in (0, \log_2(e) - 1]$, e – основание натурального логарифма. При таком выборе константы ε индивидуальная задача $I(\gamma,\ X^i)$ также имеет не более чем n допустимых решений.

Представим модификацию алгоритма решения для задачи коммивояжера с множествами предписаний работ мощности не превосходящей 2, время работы модификации составляет $O\left(q(G)2^{q(G)} + n\right)$.

Задача коммивояжера задается полным ориентированным графом $\bar{G}$ с множеством вершин $\bar{V} = \{v_1, \ldots, v_n\}$, множеством дуг $\bar{A} = \{(v_i, v_j) : v_i, v_j \in V,\ i \neq j\}$ и весами (длинами) $c(v_i, v_j) \geq 0$ дуг $(v_i, v_j) \in A$. Требуется найти гамильтонов цикл (контур) с минимальной общей длиной.

Данная задача имеет также интерпретацию как задача составления расписания работ на одной машине с учетом переналадок, где длительность переналадки с работы i на работу j есть $c(v_i, v_j)$. Предписания вершин задаются системой $X = \{X^i\}_{i=1,\dots,n}$, где $|X^i| \leq 2$. Двудольный граф $G = (X_n, X, U)$ строится так же, как описано ранее.

Проведём некоторые предварительные вычисления перед перечислением всех возможных комбинаций совершенных паросочетаний в циклах, чтобы ускорить вычисление целевой функции. *Контактом между циклом j и циклом $j' \neq j$ (или между циклом j и особым ребром)* назовём пару вершин $(i, i+1)$ (полагаем $n+1 := 1$) в левой части графа G, такую, что одна из вершин принадлежит циклу j, а другая принадлежит циклу j' (или особому ребру). *Контактом внутри цикла* будем называть пару вершин в левой части цикла, если их индексы отличаются ровно на единицу, или если эти вершины есть $(n, 1)$.

Рассмотрим цикл j. Если контакт $(i, i+1)$ присутствует внутри этого цикла, то каждое из двух совершенных паросочетаний $w^{0,j}$ и $w^{1,j}$ в этом цикле определяет i-ю дугу контура в графе $\bar{G}$. Аналогично, если цикл j имеет контакт $(i, i+1)$ с особым ребром, каждое из двух совершенных паросочетаний $w^{0,j}$ и $w^{1,j}$ также определяет i-ю дугу контура в графе $\bar{G}$. Для каждого из паросочетаний $w^{k,j}$, $k = 0, 1$, пусть сумма весов дуг, определяемых контактами внутри цикла j и контактами с особыми рёбрами, обозначается через P_j^k.

Если цикл j контактирует с циклом j', $j' \neq j$, то каждая комбинация совершенных паросочетаний этих циклов определяет i-ю дугу контура в графе $\bar{G}$ для любого контакта $(i, i+1)$ между циклами. Если совершенное паросочетание выбрано в каждом из циклов, можно просуммировать веса дуг в $\bar{G}$, определяемых всеми контактами между циклами j и j'. Это даёт четыре значения, которые обозначим через $P^{(0,0)}jj'$, $P^{(0,1)}jj'$, $P^{(1,0)}jj'$ и $P^{(1,1)}jj'$, где верхние индексы определяют выбранные паросочетания в каждом из циклов j и j' соответственно.

Параметры P^0j, P^1j, $P^{(0,0)}jj'$, $P^{(0,1)}jj'$, $P^{(1,0)}jj'$ и $P^{(1,1)}jj'$ могут быть найдены следующим образом. Предположим, что промежуточные значения P^0j, P^1j для $j = 1, \dots, q(G)$ хранятся в одномерных массивах размера $q(G)$, а промежуточные значения $P^{(0,0)}jj'$, $P^{(0,1)}jj'$, $P^{(1,0)}jj'$ и

$P^{(1,1)}jj'$ для $j, j' = 1, \ldots, q(\bar{G})$ хранятся в двумерных массивах размера $q(G) \times q(G)$. Изначально все эти значения полагаются равными нулю, и вычисляются итеративно путём последовательного просмотра пар вершин $(i, i+1)$, $i = 1, \ldots, n-1$ и $(n, 1)$ в левой части графа G. Когда рассматриваем пару вершин $(i, i+1)$ или $(n, 1)$, обновляем не более четырёх параметров (частичных сумм) в зависимости от того, принадлежат ли вершины разным циклам или одному и тому же циклу, или одна из вершин является инцидентной особому ребру. Таким образом, общая вычислительная сложность процедуры предварительной обработки составляет $O(q^2(G) + n)$.

Теперь все возможные комбинации максимальных паросочетаний в циклах могут быть перечислены с использованием кода Грея (см., например, [208]) так, что следующая комбинация отличается от предыдущей изменением совершенного паросочетания только в одном из циклов. Пусть бинарный вектор $\delta = (\delta_1, \ldots, \delta_{q(G)})$ определяет назначения совершенных паросочетаний в циклах. А именно, $\delta_j = 0$, если в цикле j выбрано паросочетание $w^{0,j}$; в противном случае (если в цикле j выбрано паросочетание $w^{1,j}$) имеем $\delta_j = 1$. Таким образом, каждый вектор δ биективно отображается в допустимое решение f_δ задачи коммивояжера с запретами.

В процессе перечисления шаг от текущего вектора $\bar{\delta}$ к следующему вектору δ изменяет совершенное паросочетание в одном из циклов j. Новое значение целевой функции $\rho(f_\delta)$ может быть вычислено через текущее значение $\rho(f_{\bar{\delta}})$ по формуле

$$\rho(f_\delta) = \rho(f_{\bar{\delta}}) - P^{\bar{\delta}j}j + P^{\delta j}j - \sum_{j' \in A(j)} P^{(\bar{\delta}j, \bar{\delta}j')}jj' + \sum_{j' \in A(j)} P^{(\delta j, \delta j')}jj',$$

где $A(j)$ — это множество циклов, контактирующих с циклом j. Очевидно, $|A(j)| \leq q(G)$, поэтому обновление значения целевой функции для следующего решения требует $O(q(\bar{G}))$ единиц времени, и общая вычислительная сложность модифицированного алгоритма решения задачи коммивояжера с запретами составляет $O\left(q(G)2^{q(G)} + n\right)$.

Использованием приведенной модификации гарантирует, что почти все индивидуальные задачи коммивояжера с запретами работ могут быть

решены за время $O(n^{0.77} \ln n + n) = O(n)$.

Отметим, что для произвольной системы подмножеств работ (без ограничений на мощность) допустимое расписание может быть найдено за полиномиальное время решением известной задачи о назначениях [189], а оптимальное решение может быть вычислено за время $O(n^{2.5} + n^2 n_{pm})$ путём перечисления совершенных паросочетаний (см., например, [232]) в соответствующем двудольном графе (n_{pm} — количество совершенных паросочетаний) и вычисления целевых функций для соответствующих расписаний. Здесь время $O(n^{2.5})$ соответствует вычислению одного совершенного паросочетания, а время $O(n^2 n_{pm})$ используется для перехода от текущего совершенного паросочетания к следующему и вычисления требуемых метрик.

3.1.3 Модели целочисленного программирования

Рассмотрим адаптацию классической модели с представлением решений на основе значений в позициях перестановки [48] для задачи на одной машине с запретами работ.

Введем следующие основные переменные:

$$x_{ij} = \begin{cases} 1, & \text{если работа } j \text{ выполняется в позиции } i, \\ 0 & \text{иначе, } i = 1, \ldots, n,\ j \in X^i; \end{cases}$$

$C_i \geq 0$ — время завершения работы в позиции i.

Добавим вспомогательные переменные для упрощения описания модели:

$y_i \geq 0$ — длительность работы в позиции i;

$z_i \geq 0$ — время поступления работы в позиции i;

$v_i \geq 0$ — директивный срок работы в позиции i.

Обозначим через Y^j подмножество позиций, где работа $j \in \mathcal{J}$ может быть выполнена, т.е. $Y^j = \{i \in X_n :\ j \in X^i\}$.

Тогда модель имеет следующие ограничения:

$$\sum_{j \in X^i} x_{ij} = 1,\ i = 1, \ldots, n, \tag{165}$$

$$\sum_{i \in Y^j} x_{ij} = 1,\ j \in \mathcal{J}, \tag{166}$$

$$y_i = \sum_{j \in X^i} x_{ij} p_j,\ i = 1, \ldots, n, \tag{167}$$

$$z_i = \sum_{j \in X^i} x_{ij} r_j,\ i = 1, \ldots, n, \tag{168}$$

$$v_i = \sum_{j \in X^i} x_{ij} d_j,\ i = 1, \ldots, n, \tag{169}$$

$$C_i \geq C_{i-1} + y_i,\ i = 2, \ldots, n, \tag{170}$$

$$C_i \geq z_i + y_i,\ i = 1, \ldots, n, \tag{171}$$

$$C_1 \geq 0, \tag{172}$$

$$x_{ij} \in \{0, 1\},\ i = 1, \ldots, n,\ j \in X^i. \tag{173}$$

Ограничения (165)–(166) гарантируют, что любая позиция содержит ровно одну работу, и каждая работа выполняется в одной позиции. Равенства (167)–(169) вычисляют длительности, времена поступления и директивные сроки работ в позициях (вспомогательные переменные). Ограничение (170) указывает, что время завершения работы в позиции i должно быть не меньше времени завершения работы в предыдущей позиции плюс длительность работы в позиции i. Условие (171) учитывает моменты поступления работ при вычислении моментов завершения. Ограничения (172)–(173) задают область определения переменных. Длительности переналадки могут моделироваться аналогично, например, работе [100].

Критерии записываются следующим образом:
максимальное временное смещение

$$L_{\max} \to \min,$$

$$L_{\max} \geq C_i - v_i,\ i = 1, \ldots, n,$$

суммарное запаздывание

$$\sum T_i = \sum_{i=1}^{n} T_i \to \min,$$

$$T_i \geq 0,\ T_i \geq C_i - v_i,\ i = 1, \ldots, n,$$

суммарное время завершения

$$\sum C_i = \sum_{i=1}^{n} C_i \to \min,$$

количество запаздывающих работ

$$\sum U_i = \sum_{i=1}^{n} U_i \to \min,$$

$$C_i \leq v_i + U_i \cdot M,\ U_i \in \{0, 1\},\ i = 1, \ldots, n.$$

Здесь и далее через M обозначается достаточно большая константа.

Таким образом, представленная модель имеет $n \leq \sum_{i=1}^{n} |X^i| \leq 2n$ булевых переменных и n непрерывных переменных для случая $|X^i| \leq 2,\ i = 1, \ldots, n$. Отметим, что модель не использует дополнительную информацию о свойствах решений для индивидуальной задачи из связи с совершенными паросочетаниями специального двудольного графа и может также использоваться для произвольной системы подмножеств работ $X = \{X^i\}_{i=1,\ldots,n}$. Модели на основе свойства смежности также известны в литературе для задач составления расписаний (см., например, [48]), но такие модели менее подходят для ограничений на запреты работ.

Рассмотрим новую идею для формулировки модели целочисленного программирования для задач составления расписаний на одной машине с системой подмножеств работ X^i для позиций $i = 1, \ldots, n$, таких что $|X^i| \leq 2$. Эта модель использует предварительные вычисления особых рёбер, циклов и совершенных паросочетаний в циклах соответствующего двудольного графа. Из результатов раздела 3.1.2 следует, что достаточно рассматривать только разрешимые индивидуальные задачи и можно

предположить, что все позиции соответствуют особым рёбрам или циклам.

Введём следующие обозначения:
N_l — множество позиций, соответствующих циклу l, $l = 1, \ldots, q(G)$;
l_i — цикл, содержащий позицию i (если позиция i соответствует особому ребру, то положим $l_i = 0$);
p_i^0 (p_i^1) — длительность работы в позиции i, соответствующей первому (второму) максимальному паросочетанию цикла l_i (если $l_i = 0$, то $p_i^0 = p_i^1$);
r_i^0 (r_i^1) — время поступления работы в позицию i, соответствующую первому (второму) максимальному паросочетанию цикла l_i (если $l_i = 0$, то $r_i^0 = r_i^1$);
d_i^0 (d_i^1) — директивный срок работы в позиции i, соответствующей первому (второму) максимальному паросочетанию цикла l_i (если $l_i = 0$, то $d_i^0 = d_i^1$);

Введём следующие булевы переменные:

$$x_l = \begin{cases} 0, & \text{если выбрано первое паросочетание в цикле } l, \\ 1, & \text{если выбрано второе паросочетание в цикле } l; \ l = 1, \ldots, q(G). \end{cases}$$

Пусть переменная C_i и вспомогательные переменные y_i, z_i и v_i имеют тот же смысл, что и в предыдущей модели для позиций $i = 1, \ldots, n$.

Тогда модель частично целочисленного линейного программирования имеет следующий вид:

$$y_i = p_i^0(1 - x_l) + p_i^1 x_l, \ l = 1, \ldots, q(G), \ i \in N_l, \tag{174}$$

$$y_i = p_i^0, \ i = 1, \ldots, n : |X^i| = 1,$$

$$z_i = r_i^0(1 - x_l) + r_i^1 x_l, \ l = 1, \ldots, q(G), \ i \in N_l, \tag{175}$$

$$z_i = r_i^0, \ i = 1, \ldots, n : |X^i| = 1,$$

$$v_i = d_i^0(1 - x_l) + d_i^1 x_l, \ l = 1, \ldots, q(G), \ i \in N_l, \tag{176}$$

$$v_i = d_i^0, \ i = 1, \ldots, n : |X^i| = 1,$$

$$C_i \geq C_{i-1} + y_i,\ i = 2, \ldots, n, \tag{177}$$

$$C_i \geq z_i + y_i,\ i = 1, \ldots, n, \tag{178}$$

$$C_1 \geq 0, \tag{179}$$

$$x_l \in \{0, 1\},\ l = 1, \ldots, q(G). \tag{180}$$

Равенства (174)–(176) идентифицируют вспомогательные переменные: длительности, моменты поступления и директивные сроки работ в позициях. Ограничения (177) и (178) позволяют вычислить моменты завершения в позициях с учётом предыдущих позиций и моментов поступления работ. Ограничения (179)–(180) задают допустимую область переменных. Здесь длительности переналадки могут быть учтены по тому же правилу, как описано ранее для задачи коммивояжера. Правил основано на контактах между циклами, соответствующими последовательным позициям перестановки.

Критерии записываются как в предыдущей модели. Количество вещественных переменных в модели составляет n, а количество булевых переменных – $q(G) \leq \lfloor \frac{n}{2} \rfloor$. Количество ограничений равно $O(n)$. Предложенная модель может использоваться для вычисления нижней оценки целевой функции или в алгоритмах типа ветвей и границ.

3.1.4 Задачи на нескольких машинах с запретами работ

Теперь рассмотрим систему с $m > 1$ машинами. Предположим, что общее количество позиций по всем машинам равно количеству работ, и все работы независимы (см. пример на рис. 23). Тогда задача с несколькими машинами может быть решена с использованием тех же методов, что и задача с одной машиной. Здесь рассматривается общая последовательность позиций $(1, \ldots, n)$ для всех машин, где i — это i'-я позиция на машине l', определяемая как $i' = i - \sum_{l=1}^{l'-1} |N_l| \leq |N_{l'}|$. Все теоремы и модели из разделов 3.1.1, 3.1.2, 3.1.3 справедливы для такого случая с несколькими машинами.

В случае, когда подмножества работ разных машин не пересекаются, задача разбивается на m независимых подзадач.

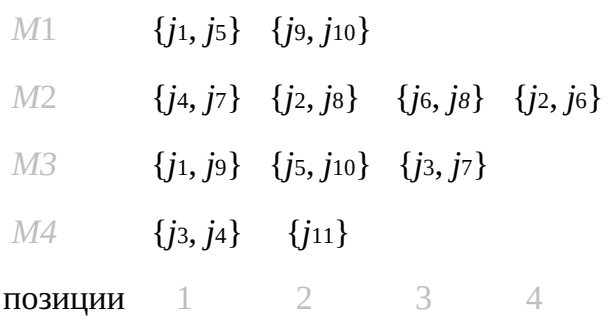


Рисунок 23 — Иллюстративный пример для случая с несколькими машинами. Здесь представлены входные данные для экземпляра индивидуальной задачи с четырьмя машинами и одиннадцатью работами. Индикаторы работ в фигурных скобках представляют подмножества доступных работ в позициях машин. Задача имеет 8 решений (представленных в соответствии с общей последовательностью позиций):
$(j_1, j_{10}, j_4, j_2, j_8, j_6, j_9, j_5, j_7, j_3, j_{11})$, $(j_1, j_{10}, j_7, j_2, j_8, j_6, j_9, j_5, j_3, j_4, j_{11})$,
$(j_1, j_{10}, j_4, j_8, j_6, j_2, j_9, j_5, j_7, j_3, j_{11})$, $(j_1, j_{10}, j_7, j_8, j_6, j_2, j_9, j_5, j_3, j_4, j_{11})$,
$(j_5, j_9, j_4, j_2, j_8, j_6, j_1, j_{10}, j_7, j_3, j_{11})$, $(j_5, j_9, j_7, j_2, j_8, j_6, j_1, j_{10}, j_3, j_4, j_{11})$,
$(j_5, j_9, j_4, j_8, j_6, j_2, j_1, j_{10}, j_7, j_3, j_{11})$, $(j_5, j_9, j_7, j_8, j_6, j_2, j_1, j_{10}, j_3, j_4, j_{11})$.

3.2 Многостадийные системы

В этом разделе рассматриваются многостадийные системы (job-shop, flow-shop и open-shop) с запретами работ. Анализируется вычислительная сложность и предлагаются методы решения.

3.2.1 Задачи job-shop и flow-shop с запретами работ

Задача составления расписаний типа job-shop с запретами работ определяется следующим образом: множество $\mathcal{J} = \{1, \ldots, n\}$ из n работ должно быть выполнено в системе с множеством $\mathcal{M} = \{1, \ldots, m\}$ машин. Работа j состоит из k_j последовательных операций. Операция O_v^j работы j с номером v, $v = 1, \ldots, k_j$, имеет длительность p_{vj} и должна быть выполнена на машине $L_{vj} \in \mathcal{M}$. Прерывания операций не допускаются. В любой момент времени каждая машина может выполнять не более одной работы, и каждая работа может обрабатываться не более чем на одной машине.

Последовательности машин $L_j = (L_{1j}, L_{2j}, \ldots, L_{k_j,j})$, $j \in \mathcal{J}$, могут

быть разными для разных работ. Пусть $O_l := \{O_v^j : \ L_{vj} = l\}, |O_l| = n_l$, обозначает множество операций, которые должны быть выполнены на машине $l \in \mathcal{M}$. Доступные для выполнения работы задаются подмножеством X^{il} для позиции $i = 1, \ldots, n_l$ машины $l \in \mathcal{M}$.

Рассматриваются следующие критерии минимизации: момент окончания всех работ (длина расписания), $C_{\max} = \max\{C_j : j \in \mathcal{J}\}$, суммарное время завершения, $\sum_j C_j$, максимальное временное смещение, $L_{\max} = \max\{C_j - d_j : j \in \mathcal{J}\}$, суммарное запаздывание, $\sum_j T_j = \sum_{j\in\mathcal{J}} \max\{C_j - d_j; 0\}$, количество запаздывающих работ, $\sum_j U_j = \sum_{j\in\mathcal{J}} \chi_{C_j>d_j}$, где d_j — директивный срок для работы j, а C_j — время завершения работы j, $j \in \mathcal{J}$.

Пусть $\pi^l = (\pi_1^l, \ldots, \pi_{n_l}^l)$ обозначает перестановку операций на машине l, т.е. π_i^l — это i-я операция на машине $l \in \mathcal{M}$, $i = 1, \ldots, n_l$. Для заданного семейства перестановок $\{\pi^l\}_{l\in\mathcal{M}}$, раннее расписание и соответствующее значение целевой функции $C_{\max}(\{\pi^l\}_{l\in\mathcal{M}})$ ($L_{\max}(\{\pi^l\}_{l\in\mathcal{M}})$, $\sum_j C_j(\{\pi^l\}_{l\in\mathcal{M}})$, $\sum_j T_j(\{\pi^l\}_{l\in\mathcal{M}})$, $\sum_j U_j(\{\pi^l\}_{l\in\mathcal{M}})$) вычисляются следующим образом.

Построим взвешенный ориентированный ациклический граф $G = (V, A)$ с множеством вершин V и множеством дуг A, где $|V| = \sum\limits_{j\in\mathcal{J}} k_j$, и вершины соответствуют операциям. Множество дуг A определяет отношения предшествования на множестве операций, заданные семейством перестановок $\{\pi^l\}_{l\in\mathcal{M}}$ и последовательностями машин для работ L_j, $j \in \mathcal{J}$. Вес (длительность) вершины-операции O_v^j полагается равным p_{vj}. Тогда раннее расписание для построенного графа может быть вычислено алгоритмом, известным как топологическая сортировка [79], за время $O(|V|+|A|) = O\left(\sum\limits_{j\in\mathcal{J}} k_j + \left(\sum\limits_{j\in\mathcal{J}}(k_j - 1) + \sum\limits_{l\in M}(n_l - 1)\right)\right) = O(nk_{\max} + mn_{\max})$, где $k_{\max} = \max\limits_{j\in\mathcal{J}} k_j$ и $n_{\max} = \max\limits_{l\in M} n_l$.

В дальнейшем будем предполагать, что расписание для семейства перестановок $\{\pi^l\}_{l\in\mathcal{M}}$ соответствует раннему расписанию для этого семейства, а значение целевой функции представлено значением целевой функции для раннего расписания.

Пусть $C_{vj}(\{\pi^l\}_{l\in\mathcal{M}})$ — время завершения операции O_v^j работы j в раннем расписании для семейства $\{\pi^l\}_{l\in\mathcal{M}}$, тогда момент окончания всех

работ

$$C_{\max}(\{\pi^l\}_{l\in\mathcal{M}}) = \max\{C_{k_j,j}(\{\pi^l\}_{l\in\mathcal{M}}) :\ j \in \mathcal{J}\},$$

суммарное время завершения

$$\sum_j C_j(\{\pi^l\}_{l\in\mathcal{M}}) = \sum_{j\in\mathcal{J}} C_{k_j,j}(\{\pi^l\}_{l\in\mathcal{M}}),$$

максимальная задержка

$$L_{\max}(\{\pi^l\}_{l\in\mathcal{M}}) = \{C_{k_j,j}(\{\pi^l\}_{l\in\mathcal{M}}) - d_j : j \in \mathcal{J}\},$$

суммарное запаздывание

$$\sum_j T_j(\{\pi^l\}_{l\in\mathcal{M}}) = \sum_{j\in\mathcal{J}} \max\{C_{k_j,j}(\{\pi^l\}_{l\in\mathcal{M}}) - d_j; 0\},$$

количество запаздывающих работ

$$\sum_j U_j(\{\pi^l\}_{l\in\mathcal{M}}) = \sum_{j\in\mathcal{J}} \chi_{C_{k_j,j}(\{\pi^l\}_{l\in\mathcal{M}})>d_j}.$$

Задача составления расписаний job-shop $J||\gamma$, $\gamma \in \{C_{\max},\ L_{\max},\ \sum_j C_j,\ \sum_j T_j\ \sum_j U_j\}$ с запретами работ формулируется как задача нахождения семейства перестановок $\{\pi^{*l}\}_{l\in\mathcal{M}}$, минимизирующего целевую функцию γ и размещающего работы из заданных подмножеств X^{il} в позициях $i = 1, \ldots, n_l$ машин $l \in \mathcal{M}$.

Задача flow-shop является частным случаем job-shop, когда все работы имеют одинаковые маршруты (последовательности операций), т.е. $L_j = (1, 2, \ldots, m)$ для $j \in \mathcal{J}$. В классическом случае flow-shop $p_{vj} = 0$ означает, что работа j выполняется на машине v, но длительность её обслуживания может быть пренебрежимо мала (обозначается $F||\gamma$ [121]). Будем исследовать классический случай, а также случай, когда $p_{vj} = 0$ означает, что работа j на машине v не выполняется. Такая задача называется задачей flow-shop с пропущенными операциями работ и обозначается $F|Miss - Oper|\gamma$ [121]. Это задача job-shop типа flow-shop. Будем также рассматривать задачу flow-shop с запретами работ. В этом случае раннее расписание и целевые функции вычисляются за время $O(nm)$ с

использованием топологической сортировки [79].

Сначала докажем NP-трудность задачи job-shop с двумя машинами, разными маршрутами работ и запретами. Затем исследуем вычислительную сложность задачи flow-shop с запретами, когда все работы имеют одинаковые маршруты.

Теорема 51 *Задача $J2||C_{\max}$ с запретами работ является NP-трудной даже для системы подмножеств $X = \{X^{i,l}\}$, такой что $|X^{i,l}| \leq 2$, $l \in \mathcal{M},\ i = 1, \ldots, n_l$.*

Доказательство. Доказательство строится на основе полиномиальной сводимости задачи Упорядоченное Разбиение к распознавательной версии задачи $J2||C_{\max}$. Положим $m = 2$ и $n = 2n_0 + 2$. Пусть длительности

$$p_{1,j} = 0,\ p_{2,j} = e_j, j = 1, \ldots, 2n_0;$$

$$p_{1,2n_0+1} = 2E,\ p_{2,2n_0+1} = E;$$

$$p_{2,2n_0+2} = 2E,\ p_{1,2n_0+2} = E.$$

Работы $j = 1, \ldots, 2n_0$ состоят из одной операции, которая должна быть выполнена на второй машине. Работа $2n_0 + 1$ состоит из двух операций: первая выполняется на первой машине, а вторая требует для выполнения вторую машину. Работа $2n_0 + 2$ также состоит из двух операций, но первая выполняется на второй машине, а вторая требует первую машину. Таким образом, эти две работы имеют разные маршруты.

Определим систему подмножеств работ как

$$X^{11} = \{2n_0 + 1\},\ X^{21} = \{2n_0 + 2\}, \tag{181}$$

$$X^{i+n_0+2,2} = X^{i,2} = \{2i - 1,\ 2i\},\ i = 1, \ldots, n_0, \tag{182}$$

$$X^{n_0+1,2} = \{2n_0 + 2\},\ X^{n_0+2,2} = \{2n_0 + 1\}. \tag{183}$$

Теперь покажем, что семейство перестановок $\{\pi'^l\}_{l \in M}$, учитывающее ограничения на запреты работ и имеющее $C_{\max}(\{\pi'^l\}_{l \in M}) \leqslant 4E$, существует тогда и только тогда, когда задача Упорядоченное Разбиение имеет положительный ответ (см. рис. 24).

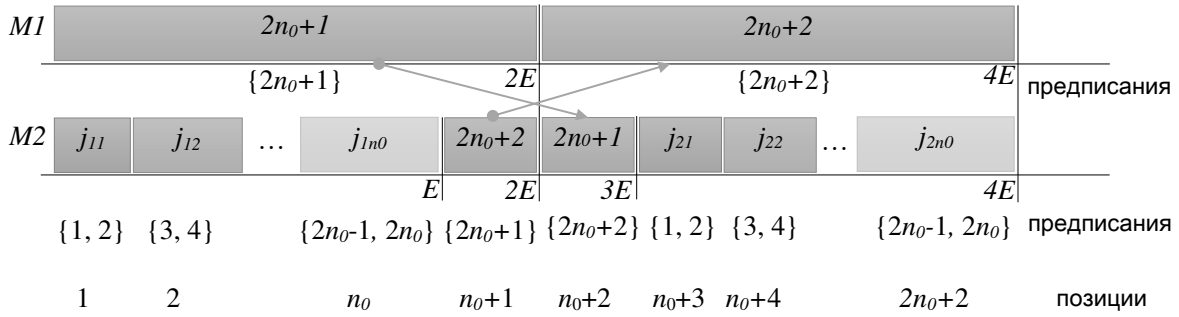


Рисунок 24 — Иллюстративный пример сведения задачи Упорядоченное Разбиение к $J2||C_{\max}$ с запретами работ. Стрелки указывают отношения предшествования между работами. Здесь представлена схема расписания без простоев, где работа $(2n_0+1)$ выполняется в интервале $[0, 3E)$, работа $(2n_0+2)$ выполняется в интервале $[E, 4E)$, а остальные работы разделены на две части $J_1 = \{j_{11}, \ldots, j_{1n_0}\}$ и $J_2 = \{j_{21}, \ldots, j_{2n_0}\}$ в соответствии с системой доступных для выполнения работ в позициях машин. Обе части имеют одинаковую общую длительность E. Такое расписание может быть построено тогда и только тогда, когда соответствующая индивидуальная задача Упорядоченное Разбиение имеет ответ «да». Только расписания такой структуры имеют целевую функцию $C_{\max} \leqslant 4E$. Все остальные расписания имеют большие значения целевой функции.

(I). Предположим, что задача Упорядоченное Разбиение имеет ответ «да», и подмножества A_1, A_2 множества A таковы, что $\sum\limits_{a_i \in A_1} e_i = \sum\limits_{a_i \in A_2} e_i = E$ и $|A_1| = |A_2| = n_0$.

Определим перестановки работ $J_1 = (j'_1, \ldots, j'_{n_0})$ и $J_2 = (j''_1, \ldots, j''_{n_0})$, соответствующие элементам из A_1 и A_2 соответственно. Здесь $j'_i < j'_{i+1}$ и $j''_i < j''_{i+1}$ для всех $i = 1, \ldots, n_0 - 1$. Тогда перестановки $\{\pi'^1 = (2n_0 + 1, 2n_0 + 2),\ \pi'^2 = (J_1, 2n_0 + 2, 2n_0 + 1, J_2),\}$ удовлетворяют условиям (181)-(183) и $C_{\max}(\{\pi'^l\}_{l \in M}) = 4E$. Обе машины работают без простоев в интервале $(0, 4E]$ в соответствующем расписании.

(II). Рассмотрим семейство перестановок $\{\pi'^l\}_{l \in M}$, удовлетворяющее условиям (181)-(183) и имеющее $C_{\max}(\{\pi'^l\}_{l \in M}) \leqslant 4E$. Из допустимости расписания следует, что работа $2n_0 + 1$ начинается в момент 0, работа $2n_0 + 2$ начинается в момент $2E$ на первой машине, и $C_{1,2n_0+1} = 2E$, $C_{2,2n_0+2} = 4E$. Следовательно, работа $2n_0 + 1$ завершается в момент $t' \geq$

$3E$, а работа $2n_0+2$ завершается в момент $t'' \le 2E$ на второй машине.

Пусть J_1 — подмножество работ, представленных в перестановке π'^2 до работы $2n_0+2$, а J_2 — подмножество работ, размещённых в π'^2 после работы $2n_0+1$. Тогда подмножества $A_1 = \{a_i \in A : \ i \in J_1\}$ и $A_2 = \{a_i \in A : \ i \in J_2\}$ удовлетворяют условиям задачи Упорядоченное Разбиение.

Представленная сводимость полиномиальна от n_0. Таким образом, задача $J2||C_{\max}$ с запретами работ является NP-трудной.

□

Отметим, что задача $J2||C_{\max}$ без запретов работ полиномиально разрешима, когда каждая работа состоит из не более чем из двух операций. Она становится NP-трудной в случае трёх машин [13].

При $d_j = 4E, \ j \in \mathcal{J}$, в задаче $J||\gamma, \ \gamma \in \{L_{\max}, \sum_j U_j, \sum_j T_j\}$, теорема 51 гарантирует, что

Теорема 52 *Задача $J2|d_j = d|\gamma, \ \gamma \in \{L_{\max}, \sum_j U_j, \sum_j T_j\}$, с запретами работ является NP-трудной даже для системы подмножеств работ $X = \{X^{il}\}$, где $|X^{il}| \le 2$, $l \in \mathcal{M}, \ i = 1, \ldots, n_l$.*

Теорема 53 *Задача $F3|Miss-Oper|C_{\max}$ с запретами работ является NP-трудной даже для системы подмножеств работ X^{il}, где $|X^{il}| \le 2$, $l \in \mathcal{M}, \ i = 1, \ldots, n_l$, причем каждая работа состоит из двух операций ненулевой длительности.*

Доказательство. Построим серию примеров для $F|Miss-Oper|C_{\max}$ с $m = 3$ и $n = 2n_0+2$. Положим длительности работ

$$p_{1,i} = p_{3,i} = e_i, \ p_{2,i} = 0, i = 1, \ldots, 2n_0;$$

$$p_{1,2n_0+1} = p_{3,2n_0+2} = E, \ p_{2,2n_0+1} = p_{2,2n_0+2} = 2E, \ p_{3,2n_0+1} = p_{1,2n_0+2} = 0.$$

Предположим, что система подмножеств работ имеет следующую структуру

$$X^{i,1} = X^{n_0+1+i,1} = \{2i-1, 2i\}, \ i = 1, \ldots, n_0, \tag{184}$$

$$X^{n_0+1,1} = \{2n_0+1\},$$

$$X^{1,2} = \{2n_0+2\}, \ = X^{2,2} = \{2n_0+1\}, \tag{185}$$

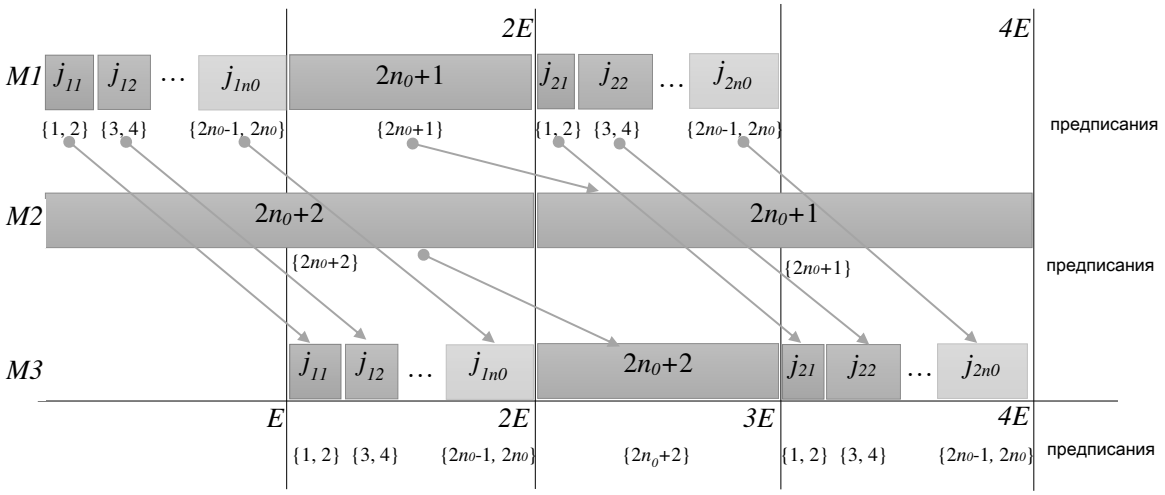


Рисунок 25 — Иллюстративный пример сведения задачи Упорядоченное Разбиение к $F3|Miss{-}Oper|C_{\max}$ с запретами работ. Стрелки указывают отношения предшествования между работами. Здесь представлена схема расписания без простоев, где работа $(2n_0 + 1)$ выполняется в интервале $[E, 4E)$, работа $(2n_0 + 2)$ выполняется в интервале $[0, 3E)$ на соответствующих машинах. Работы из множества $\{1, 2, \ldots, 2n_0\}$ разделены на две части $J_1 = \{j_{11}, \ldots, j_{1n_0}\}$ и $J_2 = \{j_{21}, \ldots, j_{2n_0}\}$ с одинаковой общей длительностью и выполняются на первой и третьей машинах последовательно без простоев. Такое расписание может быть построено тогда и только тогда, когда соответствующая индивидуальная задача Упорядоченное Разбиение имеет ответ «да». Только такие расписания имеют значение целевой функции $C_{\max} \leqslant 4E$.

$$X^{i,3} = X^{n_0+1+i,3} = \{2i - 1, 2i\},\ i = 1, \ldots, n_0, \tag{186}$$

$$X^{n_0+1,3} = \{2n_0 + 2\}.$$

Покажем, что семейство перестановок $\{\pi'^l\}_{l\in M}$, учитывающее запреты работ и имеющее $C_{\max}(\{\pi'^l\}_{l\in M}) \leqslant 4E$ существует тогда и только тогда, когда задача Упорядоченное Разбиение имеет ответ «да» (см. рис. 25).

(I). Пусть задача Упорядоченное Разбиение имеет ответ «да», и подмножества A_1, A_2 множества A таковы, что $\sum\limits_{a_i\in A_1} e_i = \sum\limits_{a_i\in A_2} e_i = E$ и $|A_1| = |A_2| = n_0$.

Обозначим через $N_1 = (j_1, \ldots, j_{n_0})$ и $N_2 = (j'_1, \ldots, j'_{n_0})$ перестановки работ, соответствующие элементам A_1 и A_2 соответственно, где $j_i < j_{i+1}$

и $j'_i < j'_{i+1}$ для всех $i = 1, \ldots, n_0 - 1$. Тогда ограничения на запреты работ и пороговое ограничение $C_{\max}(\{\pi'^l\}_{l \in M}) = 4E$ выполняются для семейства перестановок: $\{\pi'^1 = (N_1, 2n_0 + 1, N_2),\ \pi'^2 = (2n_0 + 2, 2n_0 + 1),\ \pi'^3 = (N_1, 2n_0 + 2, N_2(U))\}$. Все машины работают без простоев: машина 1 используется в интервале $(0, 3E]$, машина 2 работает в интервале $(0, 4E]$, а машина 3 занята в $(E, 4E]$.

(II). Пусть $\{\pi'^l\}_{l \in M}$ — семейство перестановок, для которого учитываются запреты работ и условие $C_{\max}(\{\pi'^l\}_{l \in M}) \leqslant 4E$ выполнено.

Из допустимости следует, что время начала и завершения $t^{st}(2n_0 + 2, 2) = 0$, $t^f(2n_0 + 2, 2) = 2E$, $t^{st}(2n_0 + 1, 2) = 2E$, $t^f(2n_0 + 1, 2) = 4E$ на второй машине. Таким образом, $t^f(2n_0 + 1, 1) \leq 2E$ и $t^{st}(2n_0 + 2, 3) \geq 2E$.

Обозначим через N_1 подмножество работ, размещённых в перестановке π'^1 до работы $2n_0 + 1$, а N_2 — подмножество работ из $\{1, \ldots, 2n_0\}$, которые начинаются до момента времени $2E$ на машине 1, но расположены в π'^1 после работы $2n_0 + 1$. Пусть $N'_1 \subseteq N_1$ и $N'_2 \subseteq N_2$ — подмножество работ, которые начинаются до момента времени $2E$ на машине 3 в расписании, соответствующем перестановке π'^3.

Предположим, что задача Упорядоченное Разбиение имеет отрицательный ответ, тогда $\sum\limits_{i \in N_1} t_{i,1} = \sum\limits_{i \in N_1} t_{i,3} = \sum\limits_{i \in N_1} e_i < E$.

Если $N'_1 = N_1$, то $N'_2 = \emptyset$ ($\{\pi'^l\}_{l \in M}$ удовлетворяет ограничениям на запреты работ) и временной бюджет $2E$ недостаточен для выполнения оставшихся работ в интервале $(2E, 4E]$. В результате приходим к противоречию с условием $C_{\max}(\{\pi'^l\}_{l \in M}) \leqslant 4E$.

Аналогично, если $N'_1 \subset N_1$, то $N'_2 \subseteq N_2$, общая длительность всех работ из $N_1 \cup N_2$ на машине 1 не превышает E, а общая длительность работ из $N'_1 \cup N'_2$ на машине 3 меньше E. Следовательно, после момента времени $2E$ сумма длительностей оставшихся работ на машине 3 больше $2E$, и также приходим к противоречию с $C_{\max}(\{\pi'^l\}_{l \in M}) \leqslant 4E$.

Сведение задачи Упорядоченное Разбиение к распознавательной версии $F3|Miss{-}Oper|C_{\max}$ построено за $O(n_0)$ операций, и, следовательно, рассматриваемая задача является NP-трудной. □

При $d_j = 4E,\ j \in \mathcal{J}$, в задаче $F|Miss - Oper|\gamma,\ \gamma \in \{L_{\max}, \sum_j U_j, \sum_j T_j\}$, теорема 53 гарантирует, что

Теорема 54 *Задача* $F3|d_j = d,\ Miss{-}Oper|\gamma,\ \gamma \in \{L_{\max}, \sum_j U_j, \sum_j T_j\}$, *с запретами работ является NP-трудной даже для системы подмножеств* $X = \{X^{il}\}$, *такой что* $|X^{i,l}| \leq 2,\ l \in \mathcal{M},\ i = 1, \ldots, n_l$.

Теперь покажем, что задача flow-shop с запретами работ и без пропущенных операций является NP-трудной в случае двух машин, когда минимизируется суммарное время завершения.

Теорема 55 *Задача* $F2||\sum_j C_j$ *с запретами работ является NP-трудной даже для системы подмножеств* $X = \{X^{il}\}$, *где* $|X^{il}| \leq 2$, $l \in \mathcal{M},\ i = 1, \ldots, n_l$.

Доказательство. Положим $m = 2$, $n = 2n_0 + 3$, и длительности

$$p_{1,j} = 1,\ p_{2,j} = e_j, j = 1, \ldots, 2n_0;$$

$$p_{1,2n_0+1} = E,\ p_{2,2n_0+1} = E;$$

$$p_{1,2n_0+2} = 2E - n_0,\ p_{2,2n_0+2} = E;$$

$$p_{1,2n_0+3} = 2E - n_0,\ p_{2,2n_0+3} = E.$$

Отметим, что все работы выполняются на обеих машинах.

Определим систему подмножеств работ для позиций как

$$X^{i+n_0+2,1} = X^{i+1,1} = \{2i - 1,\ 2i\},\ i = 1, \ldots, n_0, \tag{187}$$

$$X^{1,1} = \{2n_0 + 1\},\ X^{n_0+2,1} = \{2n_0 + 2\},\ X^{2n_0+3,1} = \{2n_0 + 3\}. \tag{188}$$

$$X^{i,2} = X^{i,1},\ i = 1, \ldots, 2n_0 + 3. \tag{189}$$

Утверждается, что семейство перестановок $\{\pi'^l\}_{l \in M}$, удовлетворяющее ограничениям на запреты работ и имеющее $\sum_j C_j(\{\pi'^l\}_{l \in M}) \leqslant L := \sum_{j=1}^{n_0}(n_0 - j + 1)(e_{2j-1} + e_{2j}) + 2E + 4E + 6E + 2En_0 + 4En_0$ существует тогда и только тогда, когда задача Упорядоченное Разбиение имеет ответ «да» (см. рис. 26).

(I). Пусть задача Упорядоченное Разбиение имеет ответ «да» с подмножествами $A_1,\ A_2$ множества A такими, что $\sum\limits_{a_i \in A_1} e_i = \sum\limits_{a_i \in A_2} e_i = E$ и $|A_1| = |A_2| = n_0$.

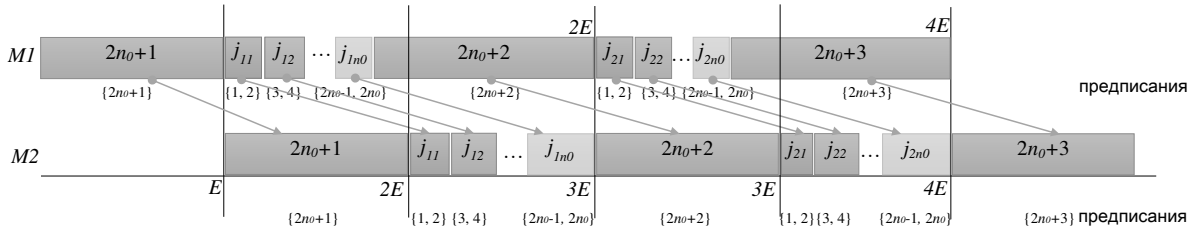


Рисунок 26 — Иллюстративный пример сведения задачи Упорядоченное Разбиение к $F2||\sum_j C_j$ с запретами работ. Здесь представлена схема расписания без простоев, где работа $(2n_0+1)$ выполняется в интервале $[0, 2E)$, работа $(2n_0+2)$ выполняется в интервале $[E+n_0, 4E)$, а работа $(2n_0+3)$ выполняется в интервале $[3E+n_0, 6E)$ на обеих машинах. Работы из множества $\{1, 2, \ldots, 2n_0\}$ разделены на две части с одинаковой общей длительностью $E+n_0$ и выполняются в $[E, E+n_0) \cup [2E, 3E)$ и $[3E, 3E+n_0) \cup [4E, 5E)$, соответственно. Такое расписание может быть построено тогда и только тогда, когда соответствующая индивидуальная задача Упорядоченное Разбиение имеет «да». Только расписания такой структуры имеют значение целевой функции $\sum_j C_j \leqslant L$.

Сформируем перестановки работ $J_1 = (j'_1, \ldots, j'_{n_0})$ и $J_2 = (j''_1, \ldots, j''_{n_0})$, соответствующие элементам из A_1 и A_2 соответственно. Здесь $j'_i < j'_{i+1}$ и $j''_i < j''_{i+1}$ для всех $i = 1, \ldots, n_0 - 1$. Тогда перестановки $\{\pi'^1 = (2n_0 + 1, J_1, 2n_0 + 2, J_2, 2n_0 + 3),\ \pi'^2 = \pi'^1\}$ размещают работы в соответствии с системой ограничений (187)-(189) и условием $\sum_j C_j(\{\pi'^l\}_{l\in M}) = L$.

(II). Рассмотрим семейство перестановок $\{\pi'^l\}_{l\in M}$, которое размещает работы в соответствии с системой ограничений (187)-(189) и приводит к $\sum_j C_j(\{\pi'^l\}_{l\in M}) \leqslant L$.

Согласно системе предписаний работа $2n_0 + 1$ $(2n_0 + 2)$ выполняется в позиции 1 $(n_0 + 2)$, а последняя позиция содержит работу $2n_0 + 3$ на обеих машинах. Обозначим через $J_i,\ i = 1,\ 2$, подмножество работ, выполняемых между работами $2n_0 + i$ и $2n_0 + i + 1$ на второй машине $(|J_1| = |J_2| = n_0)$. Требуется доказать, что общая длительность работ из J_1 равна E. Предположим, что $\sum_{j\in J_1} p_{2,j} < E$, тогда $\sum_{j\in J_2} p_{2,j} > E$ и $\sum_j C_j(\{\pi'^l\}_{l\in M}) \geq L + 1$, так как работа $2n_0 + 3$ завершается в момент $6E+1$ или позже. Если общая длительность работ из J_1 больше E, то $\sum_j C_j(\{\pi'^l\}_{l\in M}) \geq L + 1 + n_0$, поскольку работа $2n_0 + 2$ завершается не раньше момента времени $(4E + 1)$, и, как результат, работы из J_2

начинают выполняться после момента времени $(4E+1)$.

Следовательно, $\sum_{j\in J_1} p_{2,j} = \sum_{j\in J_2} p_{2,j} = E$, и задача Упорядоченное Разбиение имеет ответ «да». $\square$

Докажем NP-трудность варианта задачи flow-shop, когда перестановки работ идентичны на всех машинах (во второе поле обозначения задачи добавляется $'prmu'$).

Теорема 56 *Задача* $F3|prmu|C_{\max}$ *с запретами работ является NP-трудной даже для системы подмножеств* $X = \{X^{il}\}$, *где* $|X^{il}| \leq 2$, $l \in \mathcal{M},\ i = 1, \ldots, n_l$.

Доказательство. Покажем, что к распознавательной версии этой задачи полиномиально сводится задача Упорядоченное Разбиение.

В задаче $F|prmu|C_{\max}$ положим число машин $m = 3$, число работ $n = 2n_0 + 3$. Работы множества N разобьем на две группы: U-работы, обозначаемые U_i и соответствующие элементам $a_i \in A$, $i = 1, \ldots, 2n_0$, и V-работы, обозначаемые V_i, $i = 0, 1, 2$. Пусть

$p_{U_i,1} = p_{U_i,3} = 1$, $p_{U_i,2} = e_i, i = 1, \ldots, 2n_0$;

$p_{V_0,1} = p_{V_2,3} = 1$, $p_{V_0,2} = p_{V_1,2} = p_{V_2,2} = E$;

$p_{V_0,3} = p_{V_1,1} = p_{V_1,3} = p_{V_2,1} = 2E - n_0$.

Определим систему подмножеств работ для позиций:

$X^1 = \{V_0\}$, $X^{n_0+2} = \{V_1\}$, $X^{2n_0+3} = \{V_2\}$,

$X^{i+1} = X^{n_0+i+3} = \{U_{2i-1,\ U_{2i}}\}$, $i = 1, \ldots, n_0$.

Покажем, что в построенной задаче flow-shop с запретами работ перестановка работ π', которая согласуется с системой подмножеств и имеет $C_{\max}(\pi') \leqslant 5E + 2$, существует тогда и только тогда, когда в задаче Упорядоченное Разбиение ответ «да».

(I). Пусть в задаче Упорядоченное Разбиение ответ «да» и $A_1,\ A_2$ – подмножества элементов множества A, такие что $\sum\limits_{a_i\in A_1} e_i = \sum\limits_{a_i\in A_2} e_i = E$ и $|A_1| = |A_2| = n_0$.

Обозначим через $N_1(U) = (U_{j_1}, \ldots, U_{j_{n_0}})$ и $N_2(U) = (U_{k_1}, \ldots, U_{k_{n_0}})$ перестановки U-работ, соответствующих элементам из A_1 и A_2 соответственно, где $j_i < j_{i+1}$ и $k_i < k_{i+1}$ для всех $i = 1, \ldots, n_0 - 1$. Тогда для перестановки $\pi' = (V_0, N_1(U), V_1, N_2(U), V_2)$ выполняются условия на запреты работ и $C_{\max}(\pi') = 5E + 2$ (см. рис. 27). При этом все три машины

функционируют без простоев: машина 1 – в интервале $(0, 4E{+}1]$, машина 2 – в интервале $(1, 5E + 1]$ и машина 3 – в интервале $(E + 1, 5E + 2]$.

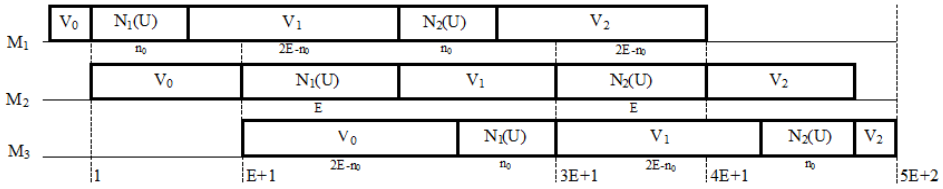


Рисунок 27 — Структура расписания для перестановки $(V_0, N_1(U), V_1, N_2(U), V_2)$ в задаче $F3|prmu|C_{\max}$ с запретами работ

(II). Пусть существует перестановка π', которая согласуется с системой предписаний работ и имеет $C_{\max}(\pi') \leqslant 5E + 2$. Будем полагать, что в задаче Упорядоченное Разбиение $E > n_0$, так как в противном случае данная задача тривиальна. Обозначим через $N'_j(U)$ множество работ, стоящих в перестановке π' между работами V_{j-1} и V_j, $j = 1, 2$. При этом $|N'_1(U)| = |N'_2(U)| = n_0$, суммарная длительность работ из $N'_j(U)$ на первой и третьей машинах равна n_0, а на второй машине – $\sum\limits_{U_i \in N'_j(U)} e_i$, где $j = 1, 2$.

В расписании, соответствующем π', все три машины функционируют без простоев. Действительно, суммарная длительность работ на первой машине равна $4E{+}1$, а так как общая длительность работы V_2 на второй и третьей машинах есть $E{+}1$, то машина 1 функционирует без простоев в интервале $(0, 4E + 1]$. Аналогичным образом можно показать, что машина 2 функционирует без простоев в интервале $(1, 5E + 1]$, а машина 3 – в интервале $(E + 1, 5E + 2]$.

Докажем, что суммарная длительность работ из $N'_1(U)$ на второй машине $\sum\limits_{U_i \in N'_1(U)} e_i = E$. Идея доказательства заключается в следующем. Если $\sum\limits_{U_i \in N'_1(U)} e_i > E$, то для работы V_1 моменты начала $t^{st}(V_1, 2) > 2E{+}1$, $t^{st}(V_1, 3) > 3E{+}1$ и, в результате, $C_{\max}(\pi') > 5E{+}2$. Если $\sum\limits_{U_i \in N'_1(U)} e_i < E$, то для работы V_1 все же момент начала $t^{st}(V_1, 2) \geqslant 2E + 1$ (поскольку суммарная длительность работ V_0, $N'_1(U)$ и V_1 на первой машине равна $2E + 1$), а так как $\sum\limits_{U_i \in N'_2(U)} e_i > E$, то $C_{\max}(\pi') > 5E + 2$.

Приведем формальное доказательство. Рассмотрим работы V_0 и V_1. Так как все три машины функционируют без простоев в соответствующих интервалах, то имеют место соотношения ($t^{st}(j,i)$, $t^f(j,i)$ обозначают моменты начала и окончания работы j на машине i):

$$t^{st}(V_1,1) = t^f(V_0,1) + n_0, \tag{190}$$

$$t^{st}(V_1,2) = t^f(V_0,2) + \sum_{U_i \in N_1'(U)} e_i \geqslant t^f(V_1,1) = t^f(V_0,1) + 2E, \tag{191}$$

$$t^{st}(V_1,3) = t^f(V_0,3) + n_0 \geqslant t^f(V_1,2) = t^f(V_0,2) + E + \sum_{U_i \in N_1'(U)} e_i, \tag{192}$$

$$t^f(V_0,2) = t^f(V_0,1) + E, \tag{193}$$

$$t^f(V_0,3) = t^f(V_0,1) + 3E - n_0. \tag{194}$$

Следовательно,

$$t^f(V_1,2) \overset{(191)}{=} t^f(V_0,2) + \sum_{U_i \in N_1'(U)} e_i + E \overset{(193)}{=} t^f(V_0,1) + 2E + \sum_{U_i \in N_1'(U)} e_i, \tag{195}$$

$$\begin{aligned} t^{st}(V_1,3) &\overset{(192);(194)}{=} t^f(V_0,1) + 3E \overset{(192)}{\geqslant} t^f(V_1,2) \\ &\overset{(195)}{=} t^f(V_0,1) + 2E + \sum_{U_i \in N_1'(U)} e_i, \end{aligned} \tag{196}$$

$$t^{st}(V_1,2) \overset{(191);(193)}{=} t^f(V_0,1) + E + \sum_{U_i \in N_1'(U)} e_i \overset{(191)}{\geqslant} t^f(V_1,1) = t^f(V_0,1) + 2E, \tag{197}$$

а значит из (196) имеем $\sum\limits_{U_i \in N_1'(U)} e_i \leqslant E$, а из (197) – $\sum\limits_{U_i \in N_1'(U)} e_i \geqslant E$.

Таким образом, $\sum\limits_{U_i \in N_1'(U)} e_i = E$. Поскольку $\sum\limits_{U_i \in N_1'(U) \cup N_2'(U)} e_i = 2E$, то $\sum\limits_{U_i \in N_2'(U)} e_i = E$.

В результате, множества $A_1 = \{a_i \in A : U_i \in N_1'(U)\}$ и $A_2 = \{a_i \in A : U_i \in N_2'(U)\}$ соответствуют условиям задачи Упорядоченное Разбиение и данная индивидуальная задача имеет ответ «да».

Построено за $O(n_0)$ операций сведение задачи Упорядоченное Разбиение к рассматриваемой задаче распознавания, и следовательно, зада-

ча $F3|prmu|C_{\max}$ с запретами работ является NP-трудной. $\square$

Теорема 57 *Задача $F3||C_{\max}$ с запретами работ является NP-трудной даже для системы подмножеств $X = \{X^{il}\}$, где $|X^{il}| \leq 2$, $l \in \mathcal{M},\ i = 1, \ldots, n_l$.*

Доказательство. Пусть в задаче $F3||C_{\max}$ с запретами работ система подмножеств работ такова, что $X^{i1} = X^{i2} = X^{i3}$. Если в оптимальном решении данной задачи перестановки $\pi^{'l}$, $l = 1, 2, 3$, не совпадают, то, используя подход из [77], можно показать, что набор перестановок $\{\pi^{''l}\}_{l \in M}$, где $\pi^{''1} = \pi^{''2} = \pi^{''3} = \pi^{'2}$, также является оптимальным для рассматриваемой задачи. Тогда из теоремы 56 непосредственно следует требуемое утверждение. $\square$

Теорема 58 *Задача $F2|prmu|L_{\max}$ с запретами работ является NP-трудной даже для системы подмножеств $X = \{X^{il}\}$, где $|X^{il}| \leq 2$, $l \in \mathcal{M},\ i = 1, \ldots, n_l$.*

Доказательство. Покажем, что к распознавательной версии этой задачи полиномиально сводится задача Упорядоченное Разбиение.

В задаче $F|prmu|L_{\max}$ положим число машин $m = 2$, число работ $n = 2n_0 + 2$. Работы множества N разобьем на две группы: U-работы, обозначаемые U_i и соответствующие элементам $a_i \in A$, $i = 1, \ldots, 2n_0$, и V-работы, обозначаемые V_i, $i = 0, 1$. Пусть
$p_{U_i,1} = e_i$, $p_{U_i,2} = 2e_i$, $d_{U_i} = 8E$, $i = 1, \ldots, 2n_0$;
$p_{V_0,1} = p_{V_1,1} = 2E$, $p_{V_0,2} = p_{V_1,2} = E$, $d_{V_0} = 3E$, $d_{V_1} = 6E$.

Определим систему подмножеств работ для позиций:
$X^1 = \{V_0\}$, $X^{i+1} = X^{n_0+i+3} = \{U_{2i-1,\ U_{2i}}\}$, $i = 1, \ldots, n_0$, $X^{n_0+2} = \{V_1\}$.

Покажем, что в построенной задаче с запретами работ перестановка работ π', которая согласуется с системой предписаний работ и имеет $L_{\max}(\pi') \leqslant 0$, существует тогда и только тогда, когда в задаче Упорядоченное Разбиение ответ «да». Пусть $t^{st}(j, i),\ t^{f}(j, i)$ обозначают моменты начала и окончания работы j на машине i.

(I). Пусть в задаче Упорядоченное Разбиение ответ «да» и $A_1,\ A_2$ – подмножества элементов множества A, такие что $\sum\limits_{a_i \in A_1} e_i = \sum\limits_{a_i \in A_2} e_i = E$ и $|A_1| = |A_2| = n_0$.

Обозначим через $N_1(U) = (U_{j_1}, \ldots, U_{j_{n_0}})$ и $N_2(U) = (U_{k_1}, \ldots, U_{k_{n_0}})$ перестановки U-работ, соответствующих элементам из A_1 и A_2 соответственно, где $j_i < j_{i+1}$ и $k_i < k_{i+1}$ для всех $i = 1, \ldots, n_0 - 1$. Тогда для перестановки $\pi' = (V_0, N_1(U), V_1, N_2(U))$ выполняется условие на запреты работ и $L_{\max}(\pi') = 0$, так как в расписании для π' соблюдены директивные сроки всех работ и $t^f(V_0, 2) = d_{V_0}$ (см. рис. 28). При этом обе машины функционируют без простоев: машина 1 – в интервале $(0, 6E]$, а машина 2 – в интервале $(2E, 8E]$.

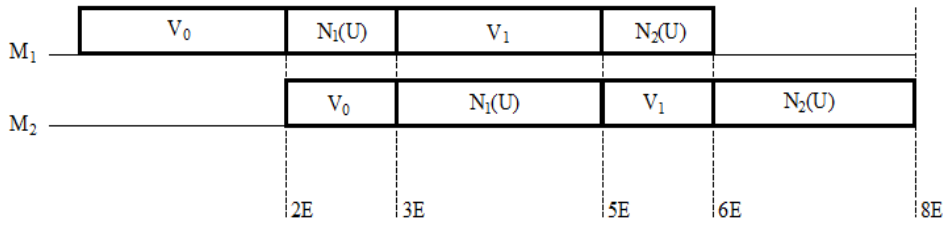


Рисунок 28 — Структура расписания для перестановки $(V_0, N_1(U), V_1, N_2(U))$ в задаче $F2|prmu|L_{\max}$ с запретами работ.

(II). Пусть существует перестановка π', которая согласуется с системой предписаний работ и имеет $L_{\max}(\pi') \leqslant 0$. Обозначим через $N'_j(U)$ множество U-работ, стоящих в перестановке π' после работы V_{j-1}, $j = 1, 2$.

Поскольку $p_{V_0,1} + p_{V_0,2} = 3E = d_{V_0}$, то работа V_0 обслуживается машиной 1 в интервале $(0, 2E]$, а машиной 2 – в интервале $(2E, 3E]$. Так как суммарная длительность обслуживания всех работ на машине 2 равна $6E$, то данная машина функционирует без простоев в интервале $(2E, 8E]$.

Рассмотрим временной интервал $(3E, 6E]$. В этом интервале машина 2 должна выполнить работу V_1 в течение E единиц времени (так как $d_{V_1} = 6E$) и U-работы в течение $2E$ единиц времени.

Очевидно, что $t^f(V_1, 1) \leqslant 5E$, иначе $t^f(V_1, 2) > 6E$. Предположим, что $t^f(V_1, 1) = 5E - C$, где $0 < C < E$. Тогда машина 1 выполняет в $(2E, 3E - C]$ работы из $N'_1(U)$ в течение не более $E - C$ единиц времени, и эти же работы обслуживаются на машине 2 в течение не более $2E - 2C$ единиц времени в $(3E, 5E - C]$ (см. рис. 29). Однако на машине 2 момент начала $t^{st}(V_1, 2) \geqslant 5E - C$, а значит, после обслуживания V_1 на машине 2 останется не более $2E + C$ единиц времени до момента $8E$,

которых не хватит на выполнение работ из $N_2'(U)$, суммарная длительность которых на машине 2 составит не менее $2E + 2C$. Следовательно, $t^f(V_1, 1) = t^{st}(V_1, 2) = 5E$, $\sum\limits_{U_i \in N_1'(U)} e_i = E$ и $\sum\limits_{U_i \in N_2'(U)} e_i = E$.

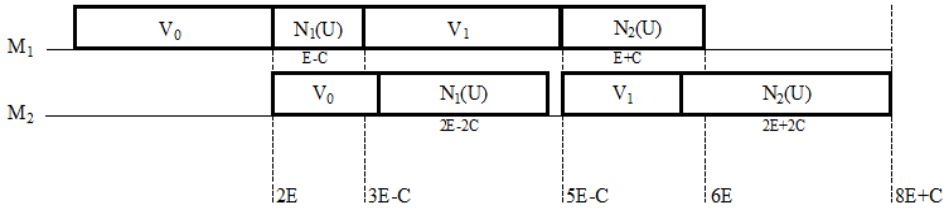


Рисунок 29 — Иллюстрация к доказательству от противного для перестановки π' в задаче $F2|prmu|L_{\max}$.

Таким образом, множества $A_1 = \{a_i \in A : \ U_i \in N_1'(U)\}$ и $A_2 = \{a_i \in A : U_i \in N_2'(U)\}$ соответствуют условиям задачи Упорядоченное Разбиение и данная индивидуальная задача имеет ответ «да».

Построено за $O(n_0)$ операций сведение задачи Упорядоченное Разбиение к рассматриваемой задаче распознавания, и следовательно, задача $F2|prmu|L_{\max}$ с запретами работ является NP-трудной. □

Теорема 59 *Задача $F2||L_{\max}$ с запретами работ является NP-трудной даже для системы подмножеств $X = \{X^{il}\}$, где $|X^{il}| \leq 2$, $l \in \mathcal{M}$, $i = 1, \ldots, n_l$.*

Доказательство. Пусть в задаче $F2||L_{\max}$ с запретами работ система предписаний работ такова, что $X^{1i} = X^{2i}$. Если в оптимальном решении данной задачи перестановки π'^1 и π'^2 не совпадают, то, используя подход из [77], можно показать, что набор перестановок $\{\pi''^l\}_{l \in M}$, где $\pi''^1 = \pi''^2 = \pi'^2$, также является оптимальным для рассматриваемой задачи. Тогда из теоремы 58 непосредственно следует требуемое утверждение. □

В задаче flow-shop без задержек ($F|no-wait|\gamma$ [121]) предполагается, что каждая работа выполняется непрерывно одной машиной за другой. Предположим, что в задаче $F|no - wait|\gamma$ допускаются $t_{il} = 0$ и это означает, что работа i на машине l не выполняется. При указанной интерпретации записи $t_{il} = 0$ в оптимальном решении задачи $F|no-wait|\gamma$ машины могут выполнять работы ненулевой длительности в различном порядке.

Будем рассматривать задачу $F|no-wait,\ prmu|\gamma,\ \gamma \in \{C_{\max},\ L_{\max}\}$, которая является NP-трудной в сильном смысле уже при $m = 2$ [111]. Для перестановки $\pi = (\pi_1, \ldots, \pi_n)$, определяющей порядок выполнения работ, раннее расписание задачи $F|no-wait,\ prmu|\gamma,\ \gamma \in \{C_{\max},\ L_{\max}\}$, вычисляется за время $O(mn)$. Обозначим через $M_i = (l^i_1, \ldots, l^i_{m_i})$ последовательность машин, на которых выполняется работа $i \in N$, а через $\pi^k_{j'}$ – работу, которая предшествует работе π_j на машине $l^{\pi_j}_k$ согласно последовательности π (если работа π_j выполняется на машине $l^{\pi_j}_k$ первой, то полагаем $\pi^k_{j'} := 0,\ t^f(\pi^k_{j'}, l^{\pi_j}_k) := 0$), $j = 1, \ldots, n,\ k = 1, \ldots, m_j$. Тогда моменты начала и окончания работ на машинах $t^{st}(j,i)$ и $t^f(j,i)$ вычисляются по правилу

$$t^{st}(\pi_1, l^{\pi_1}_1) = 0,\ t^{st}(\pi_1, l^{\pi_1}_j) = t^f(\pi_1, l^{\pi_1}_{j-1}),\ j = 2, \ldots, m_1;$$

$$t^{st}(\pi_i, l^{\pi_i}_1) = \max_{k=1,\ldots,m_i} \left\{ t^f(\pi^k_{i'}, l^{\pi_i}_k) - \sum_{j=1}^{k-1} p_{\pi_i, l^{\pi_i}_j} \right\},\ i = 2, \ldots, n;$$

$$t^{st}(\pi_i, l^{\pi_i}_j) = t^f(\pi_i, l^{\pi_i}_{j-1}),\ i = 2, \ldots, n,\ j = 2, \ldots, m_i;$$

$$C_{\max}(\pi) = \max\{t^f(i, l^i_{m_i}) :\ i \in N\};$$

$$L_{\max}(\pi) = \max\{t^f(i, l^i_{m_i}) - d_i :\ i \in N\}.$$

Теорема 60 *Задача $F2|no-wait,\ prmu|C_{\max}$ с запретами работ является NP-трудной, если запись $t_{il} = 0$ означает, что работа i на машине l не обслуживается, даже для системы подмножеств $X = \{X^{il}\}$, где $|X^{il}| \le 2$, $l \in \mathcal{M},\ i = 1, \ldots, n_l$.*

Доказательство. Покажем, что к распознавательной версии рассматриваемой задачи полиномиально сводится задача Упорядоченное Разбиение.

В задаче $F|no-wait,\ prmu|C_{\max}$ положим число машин $m = 2$, число работ $n = 2n_0 + 3$. Работы множества N разобьем на две группы: U-работы, обозначаемые U_i и соответствующие элементам $a_i \in A$, $i = 1, \ldots, 2n_0$, и V-работы, обозначаемые V_i, $i = 0, 1, 2$. Пусть

$p_{U_i,1} = 0$, $p_{U_i,2} = e_i, i = 1, \ldots, 2n_0$;

$p_{V_0,1} = p_{V_1,1} = p_{V_2,1} = 2E$;

$p_{V_0,2} = p_{V_1,2} = p_{V_2,2} = E$.

Определим систему подмножеств работ для позиций:
$X^1 = \{V_0\}$, $X^{n_0+2} = \{V_1\}$, $X^{2n_0+3} = \{V_2\}$,
$X^{i+1} = X^{n_0+i+3} = \{U_{2i-1,\ U_{2i}}\}$, $i = 1, \ldots, n_0$.

Поскольку U-работы не выполняются на машине 1, то система предписаний работ порождает следующую подсистему предписаний для указанной машины:
$\tilde{X}^{11} = \{V_0\},\ \tilde{X}^{12} = \{V_1\},\ \tilde{X}^{13} = \{V_2\}$.

Покажем, что в построенной задаче перестановка работ π', которая согласуется с системой предписаний и имеет $C_{\max}(\pi') \leqslant 7E$, существует тогда и только тогда, когда в задаче Упорядоченное Разбиение ответ «да».

(I). Пусть в задаче Упорядоченное Разбиение ответ «да» и $A_1,\ A_2$ – подмножества элементов множества A, такие что $\sum\limits_{a_i \in A_1} e_i = \sum\limits_{a_i \in A_2} e_i = E$ и $|A_1| = |A_2| = n_0$.

Обозначим через $N_1(U) = (U_{j_1}, \ldots, U_{j_{n_0}})$ и $N_2(U) = (U_{k_1}, \ldots, U_{k_{n_0}})$ перестановки U-работ, соответствующих элементам из A_1 и A_2 соответственно, где $j_i < j_{i+1}$ и $k_i < k_{i+1}$ для всех $i = 1, \ldots, n_0 - 1$. Тогда для перестановки $\pi' = (V_0, N_1(U), V_1, N_2(U), V_2)$ выполняются условия на запреты работ и $C_{\max}(\pi') = 7E$ (см. рис. 30). При этом машины функционируют без простоев: машина 1 – в интервале $(0, 6E]$ (обслуживает работы в последовательности $\tilde{\pi}' = (V_0, V_1, V_2)$), а машина 2 – в интервале $(2E, 7E]$ (обслуживает работы в последовательности π').

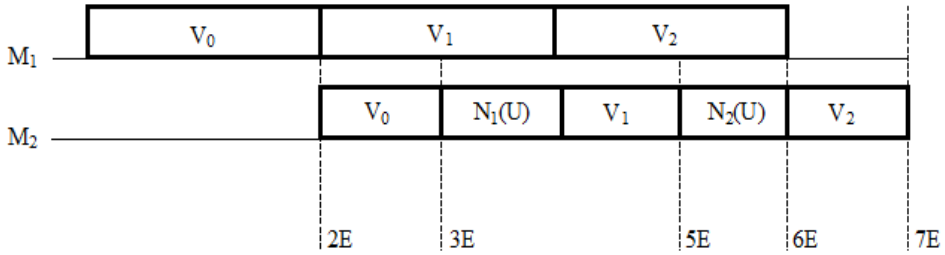


Рисунок 30 — Структура расписания для перестановки $(V_0, N_1(U), V_1, N_2(U), V_2)$ в задаче $F2|no-wait,\ prmu|C_{\max}$.

(II). Пусть существует перестановка π', которая согласуется с системой предписаний и имеет $C_{\max}(\pi') \leqslant 7E$. Обозначим через $N'_j(U)$ множество работ, стоящих в перестановке π' между работами V_{j-1} и V_j, $j = 1, 2$.

Поскольку суммарная длительность всех работ на первой машине

равна $6E$, на второй машине равна $5E$, $p_{V_0,1} = 2E$, $p_{V_2,2} = E$, то обе машины функционируют без простоев: машина 1 – в интервале $(0, 6E]$, а машина 2 – в интервале $(2E, 7E]$.

Так как задержки при выполнении работ запрещены, то работа V_0 выполняется на машине 1 в интервале $(0, 2E]$, а на машине 2 – в интервале $(2E, 3E]$; работа V_1 выполняется на машине 1 в интервале $(2E, 4E]$, а на машине 2 – в интервале $(4E, 5E]$; работа V_2 выполняется на машине 1 в интервале $(4E, 6E]$, а на машине 2 – в интервале $(6E, 7E]$. Тогда U-работы обслуживаются на машине 2 в интервалах $(3E, 4E]$ и $(5E, 6E]$, следовательно $\sum\limits_{U_i \in N_1'(U)} e_i = E$ и $\sum\limits_{U_i \in N_2'(U)} e_i = E$.

В результате, множества $A_1 = \{a_i \in A : \ U_i \in N_1'(U)\}$ и $A_2 = \{a_i \in A : U_i \in N_2'(U)\}$ соответствуют условиям задачи Упорядоченное Разбиение и данная индивидуальная задача имеет ответ «да».

Построено за $O(n_0)$ операций сведение задачи Упорядоченное Разбиение к рассматриваемой задаче распознавания, и следовательно, задача $F|no-wait,\ prmu|C_{\max}$ с запретами работ является NP-трудной. □

Если в задаче $F|no-wait,\ prmu|L_{\max}$ положить $d_i = 0,\ i \in N$, то из теоремы 60 будет непосредственно следовать

Теорема 61 *Задача $F2|no-wait,\ prmu|L_{\max}$ с запретами работ является NP-трудной, если запись $t_{il} = 0$ означает, что работа i на машине l не обслуживается, даже для системы подмножеств $X = \{X^{il}\}$, где $|X^{il}| \leq 2$, $l \in \mathcal{M},\ i = 1, \ldots, n_l$.*

3.2.2 Задача open-shop с запретами работ

В задаче open-shop порядок, в котором работы выполняются на машине, и порядок, в котором работа обрабатывается машинами, могут быть выбраны произвольно. В постановке с запретами работ имеют место дополнительные ограничения, указывающие, что позиция i машины l может содержать только работы из заданного подмножества $X^{i,l} \subseteq \mathcal{J}$.

Предположим, что $X^{i,l} = \{i\}$, т.е. $|X^{i,l}| = 1$ для всех $l \in \mathcal{M}$, и поменяем смысл работ и машин местами. Тогда получим классическую задачу flow-shop , которая является NP-трудной даже в случае трёх ра-

бот [223]. Таким образом, задача open-shop с запретами работ является NP-трудной даже в случае трёх машин. Более того, используя те же аргументы, что и в теоремах 51, 52 для job-shop на двух машинах с системой подмножеств работ мощностью не более двух можно доказать, что

Теорема 62 *Задача* $O2|Miss-Oper|\gamma,\ \gamma \in \{C_{\max}, L_{\max}, \sum_j U_j, \sum_j T_j\}$, *с запретами работ является NP-трудной даже для системы подмножеств* $X = \{X^{il}\}$, *где* $|X^{il}| \leq 2$, $l \in \mathcal{M},\ i = 1, \ldots, n_l$.

Здесь нет отношений предшествования между операциями в отличие от job-shop, но запреты, используемые в доказательствах теорем 51, 52 и пороговые значения задают требуемые условия.

Теорема 63 *Задача* $O2|Miss - Oper|\sum_j C_j$ *с запретами работ является NP-трудной даже для системы подмножеств* $X = \{X^{il}\}$, *где* $|X^{il}| \leq 2$, $l \in \mathcal{M},\ i = 1, \ldots, n_l$.

Доказательство. Положим $m = 2$, $n = 2n_0 + 3$, и длительности

$$p_{1,j} = 0,\ p_{2,j} = e_j, j = 1, \ldots, 2n_0;$$

$$p_{1,2n_0+1} = 2E,\ p_{2,2n_0+1} = E;$$

$$p_{2,2n_0+2} = 2E,\ p_{1,2n_0+2} = E;$$

$$p_{1,2n_0+3} = 0,\ p_{2,2n_0+3} = E.$$

Работы $j = 1, \ldots, 2n_0$ и $2n_0 + 3$ состоят из одной операции, выполняемой на второй машине. Работы $2n_0 + 1$ и $2n_0 + 2$ состоят из двух операций, одна из которых требует первую машину, а другая работа использует вторую машину.

Определим систему подмножеств работ для позиций как

$$X^{1,1} = \{2n_0 + 1\},\ X^{2,1} = \{2n_0 + 2\}, \tag{198}$$

$$X^{i+n_0+2,2} = X^{i,2} = \{2i - 1,\ 2i\},\ i = 1, \ldots, n_0, \tag{199}$$

$$X^{n_0+1,2} = \{2n_0 + 2\},\ X^{n_0+2,2} = \{2n_0 + 1\},\ X^{2n_0+3,2} = \{2n_0 + 3\}. \tag{200}$$

Утверждается, что семейство перестановок $\{\pi'^l\}_{l\in M}$, удовлетворяющее ограничениям на запреты работ и имеющее $\sum_j C_j(\{\pi'^l\}_{l\in M}) \leqslant L := \sum_{j=1}^{n_0}(n_0-j+1)(e_{2j-1}+e_{2j}) + 4E + 3E + 3En_0 + 5E$ существует тогда и только тогда, когда задача Упорядоченное Разбиение имеет ответ «да» (см. рис. 31).

(I). Пусть задача Упорядоченное Разбиение имеет ответ «да» с подмножествами $A_1,\ A_2$ множества A такими, что $\sum\limits_{a_i\in A_1} e_i = \sum\limits_{a_i\in A_2} e_i = E$ и $|A_1| = |A_2| = n_0$.

Определим перестановки работ $J_1 = (j'_1, \ldots, j'_{n_0})$ и $J_2 = (j''_1, \ldots, j''_{n_0})$, соответствующие элементам из A_1 и A_2 соответственно. Здесь $j'_i < j'_{i+1}$ и $j''_i < j''_{i+1}$ для всех $i = 1, \ldots, n_0 - 1$. Тогда перестановки $\{\pi'^1 = (2n_0 + 1, 2n_0+2),\ \pi'^2 = (J_1, 2n_0+2, 2n_0+1, J_2, 2n_0+3)\}$ удовлетворяют системе ограничений (198)-(200) и условию $\sum_j C_j(\{\pi'^l\}_{l\in M}) = L$.

(II). Рассмотрим семейство перестановок $\{\pi'^l\}_{l\in M}$, которое удовлетворяет системе ограничений (198)-(200) и имеет $\sum_j C_j(\{\pi'^l\}_{l\in M}) \leqslant L$.

Согласно ограничениям на запреты работа $2n_0 + 2$ $(2n_0 + 1)$ выполняется в позиции $n_0 + 1$ $(n_0 + 2)$, а последняя позиция содержит работу $2n_0 + 3$ на второй машине. Обозначим через J_1 подмножество работ, выполняемых до работы $2n_0 + 2$, а через J_2 – подмножество работ, выполняемых после работы $2n_0 + 1$ и до работы $2n_0 + 3$ на второй машине ($|J_1| = |J_2| = n_0$). Требуется доказать, что общая длительность работ из J_1 равна E. Предположим, что $\sum_{j\in J_1} p_j < E$, тогда $\sum_{j\in J_2} p_j > E$ и $\sum_j C_j(\{\pi'^l\}_{l\in M}) \geq L + 1$, так как работа $2n_0 + 3$ завершается в момент $5E + 1$ или позже. Если суммарная длительность работ из J_1 больше E, то $\sum_j C_j(\{\pi'^l\}_{l\in M}) \geq L + 2 + n_0$, поскольку работы из J_2 начинают выполняться после момента времени $(3E + 1)$, работа $2n_0 + 1$ завершается не раньше момента времени $(3E + 1)$, а работа $2n_0 + 2$ завершается в момент $4E + 1$ или позже.

Следовательно, $\sum_{j\in J_1} p_j = \sum_{j\in J_2} p_j = E$, и задача Упорядоченное Разбиение имеет ответ «да». □

3.2.3 Подход к решению

Перебирая на машинах независимо друг от друга все возможные комбинаций допустимых перестановок работ, согласованные с запретами на

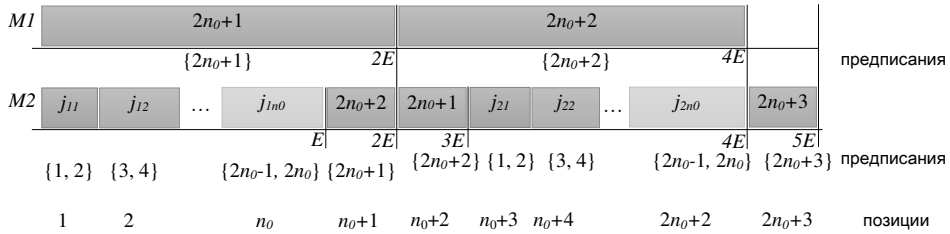


Рисунок 31 — Иллюстративный пример для сводимости задачи Упорядоченное Разбиение к $O2||\sum_j C_j$ с запретами работ. Здесь представлена схема расписания без простоев, где работа $(2n_0+1)$ выполняется в интервале $[0, 2E)$ на первой машине и в $[2E, 3E)$ на второй машине, работа $(2n_0+2)$ выполняется в интервале $[E, 2E)$ на второй машине и в $[2E, 4E)$ на первой машине, а работа $(2n_0+3)$ выполняется в интервале $[4E, 5E)$ на второй машине. Остальные работы разделены на две части $J_1 = \{j_{11}, \ldots, j_{1n_0}\}$ и $J_2 = \{j_{21}, \ldots, j_{2n_0}\}$ с одинаковой общей длительностью E и выполняются в интервалах $[0, E)$ и $[3E, 4E)$ на второй машине. Такое расписание может быть построено тогда и только тогда, когда соответствующая индивидуальная задача Упорядоченное Разбиение имеет ответ «да». Только расписания такой структуры имеют значение целевой функции $\sum_j C_j \leqslant \sum_{j=1}^{n_0}(n_0 - j + 1)(e_{2j-1} + e_{2j}) + (3n_0 + 12)E$.

выполнение работ, и используя подход из раздела 3.1.2, получаем следующий точный алгоритм решения задачи job-shop с запретами работ.

Для каждой машины l (независимо от остальных) строится двудольный граф, вычисляются его особые рёбра и циклы за время $O(n_l)$. Затем перечисляются все возможные комбинации совершенных паросочетаний в графах, соответствующих машинам. Общее количество комбинаций составляет $O(1.42^{n_{\max}m})$. Для каждой комбинации строится раннее допустимое расписание и вычисляется целевая функция за время $O(n_{\max}m + k_{\max}n)$. Решение с наилучшим значением целевой функции соответствует оптимальному расписанию. Таким образом, имеет место

Теорема 64 *Задача $J||\gamma$, $\gamma \in \{C_{\max}, L_{\max}, \sum_j T_j, \sum_j C_j, \sum_j U_j\}$, с запретами работ может быть решена за время $O\left(1.42^{n_{\max}m}(nk_{\max} + mn_{\max})\right)$ для системы подмножеств работ $X^{i,l}$ такой, что $|X^{i,l}| \leq 2$, $l \in \mathcal{M}$, $i = 1, \ldots, n_l$.*

Из теоремы 50 заключаем, что почти все системы подмножеств работ $X^{i,l}$ такие, что $|X^{i,l}| \leq 2$, приводят к полиномиально разрешимым инди-

видуальным задачам $Jm||\gamma$, $Fm||\gamma$, $Fm|Miss-Oper|\gamma$, $\gamma \in \{C_{\max}, L_{\max}, \sum_j T_j, \sum_j C_j, \sum_j U_j\}$ с запретами работ.

Для произвольных систем подмножеств работ на машинах (без ограничений на мощность) допустимое расписание может быть найдено за полиномиальное время с помощью решения задачи о назначениях [189] для всех m машин независимо. Оптимальное решение в этом случае может быть вычислено за время $O\left(n_{\max}^{2.5}m + \left(n_{pm}^{\max}\right)^m \left(mn_{\max}^2 + nk_{\max}\right)\right)$ путём перебора совершенных паросочетаний (см. раздел 3.1.2) в соответствующих двудольных графах для машин ($n_{pm}^{\max}$ — максимальное количество совершенных паросочетаний в графах для машин) и вычисления целевых функций для соответствующих расписаний.

В случае задачи open-shop с критерием минимизации длины расписания применить подход для решения с перечислением перестановок операций на машинах и полиномиальным временем вычисления целевой функции можно для случая $m = 2$. Здесь получаем классические полиномиально разрешимые индивидуальные задачи job-shop для вычисления оптимального значения целевой функции при заданных последовательностях на двух машинах [13] (последовательности представляют две работы с заданными маршрутами операций для job-shop). Целевая функция вычисляется за время $O(n\log(n))$ [13] для заданной перестановки. Таким образом, имеем

Теорема 65 *Задача $O2||C_{\max}$ с запретами работ может быть решена за время $O\left(1.42^{2n}n\log(n)\right)$ для системы подмножеств работ $X^{i,l}$ такой, что $|X^{i,l}| \leq 2$, $l \in \mathcal{M}$, $i = 1, \ldots, n_l$.*

Напомним, что $O2||C_{\max}$ без запретов на выполнение работ полиномиально разрешима [13].

Целевая функция не может быть вычислена за полиномиальное время для фиксированных последовательностей работ на трёх и более машинах, если P≠NP (здесь имеем NP-трудную задачу job-shop [13]).

3.2.4 Модели целочисленного программирования

Приведем адаптацию классической модели частично целочисленного линейного программирования, основанной на назначении работ в пози-

ции машин [162], для задач job-shop и flow-shop с запретами работ.

Пусть O — множество всех операций всех работ, а O_l — подмножество операций, которые должны быть выполнены на машине $l \in \mathcal{M}$, $n_l := |O_l|$. Обозначим через p_o длительность операции o, а через $Y^{o,l}$ — подмножество позиций, где операция o может быть размещена на машине l, причем $o = O_v^j,\ i = L_{vj},\ j \in \mathcal{J},\ v = 1, \ldots, k_j$. Пусть $n_{\max} := \max_{l \in \mathcal{M}} n_l$ и $k_{\max} := \max_{j \in \mathcal{J}} k_j$.

Введём следующие булевы переменные:

$$x_{oi} = \begin{cases} 1, & \text{если операция } o \text{ выполняется в позиции } i \text{ машины } l, \\ 0 & \text{иначе; } i = 1, \ldots, n_l,\ o \in X^{i,l},\ l \in \mathcal{M}, \end{cases}$$

и вещественные переменные $C_{li} \geq 0$ (время завершения операции работы в позиции $i = 1, \ldots, n_l$ машины $l \in \mathcal{M}$).

Тогда множество допустимых решений определяется следующим образом:

$$\sum_{o \in X^{i,l}} x_{oi} = 1,\ l \in \mathcal{M},\ i = 1, \ldots, n_l, \tag{201}$$

$$\sum_{i \in Y^{o,l}} x_{oi} = 1,\ l \in \mathcal{M},\ o \in O_l, \tag{202}$$

$$C_{li} + \sum_{o \in X^{i+1,l}} p_o x_{o,i+1} \leq C_{l,i+1},\ l \in \mathcal{M},\ i = 1, \ldots, n_l - 1, \tag{203}$$

$$C_{l_1,i_1} + p_{o_2} \leq C_{l_2,i_2} + \tag{204}$$

$$M(1 - x_{o_1,i_1}) + M(1 - x_{o_2,i_2}),$$

$$j \in \mathcal{J}, v = 1, \ldots, k_j - 1,\ o_1 = O_v^j,\ o_2 = O_{v+1}^j,$$

$$l_1 = L_{v,j},\ l_2 = L_{v+1,j},\ i_1 \in Y^{o_1,l_1},\ i_2 \in Y^{o_2,l_2},$$

$$C_{li} \geq 0,\ x_{oi} \in \{0, 1\},\ o \in X^{i,l},\ l \in \mathcal{M},\ i = 1, \ldots, n_l. \tag{205}$$

Ограничение (201) гарантирует, что в каждую позицию назначается ровно одна допустимая для выполнения операция. Условия (202) задают то, что все операции работ размещены в позициях. Неравенство (203) задает нижнюю границу на время завершения операции в следующей позиции машины. Условия (204) обеспечивают выполнение частичного

порядка между операциями одной работы. Ограничение (205) задаёт допустимую область переменных. Модель содержит $O(mn_{\max})$ булевых переменных и $O(mn_{\max})$ вещественных переменных для системы подмножеств работ в позициях с $|X^{i,l}| \le 2,\ l \in \mathcal{M},\ i = 1, \ldots, n_l$.

Критерии могут быть записаны следующим образом:
время окончания всех работ

$$C_{\max} \to \min,$$

$$C_{l,n_l} \le C_{\max},\ l \in \mathcal{M},$$

максимальное временное смещение

$$L_{\max} \to \min,$$

$$L_{\max} \ge C_{li} - \sum_{o \in X^{i,l},\ j \in \mathcal{J}:\ o = O_v^j} x_{oi} d_j,\ l \in \mathcal{M},\ i = 1, \ldots, n_l,$$

суммарное запаздывание

$$\sum T_j = \sum_{j=1}^{n} T_j \to \min,$$

$$T_j \ge 0,\ T_j \ge C_{li} - d_j - M(1 - x_{oi}),\ j = 1, \ldots, n,\ o = O_{k_j}^j,\ k \in Y^{o, L_{k_j, j}},$$

суммарное время завершения

$$\sum C_j = \sum_{j=1}^{n} C_j \to \min,$$

$$C_j \ge C_{li} - M(1 - x_{oi}),\ j = 1, \ldots, n,\ o = O_{k_j}^j,\ i \in Y^{o, L_{k_j, j}},$$

количество запаздывающих работ

$$\sum U_j = \sum_{j=1}^{n} U_j \to \min,$$

$$C_j \le d_j + U_j \cdot M,\ U_j \in \{0, 1\},\ j = 1, \ldots, n.$$

Перейдем к описанию модели, основанной на совершенных паросоче-

таниях во вспомогательном двудольном графе, для случая $|X^{il}| \leq 2,\ l \in \mathcal{M},\ i = 1, \ldots, n_l$, аналогичную модели для случая одного процессора из раздела 3.1.3. Пусть $s = 1, \ldots, q(l)$ нумерует циклы двудольных графов, соответствующих системам подмножеств работ на машинах $l \in \mathcal{M}$.

Рассмотрим следующие булевы переменные:

$$x_{sl} = \begin{cases} 0, & \text{если выбрано первое паросочетание в цикле } s \text{ машины } l, \\ 1, & \text{если выбрано второе паросочетание в цикле } s \text{ машины } l; \end{cases}$$

и вещественные переменные $C_{li} \geq 0$ (время завершения операции работы в позиции i машины l), $C(O_v^j) \geq 0$ (время завершения операции O_v^j).

Будем использовать следующие обозначения:

s_{li} – цикл машины l, содержащий позицию i;

s_{vj} – цикл машины L_{vj}, содержащий операцию O_v^j;

$p_{li}^0(p_{li}^1)$ — длительность операции в позиции i машины l, соответствующая первому (второму) совершенному паросочетанию цикла s_{li};

k_{vj}^0 (k_{vj}^1) — позиция машины L_{vj}, соответствующая операции O_v^j в соответствии с первым (вторым) совершенным паросочетанием цикла l_{vj}.

В данном подходе ограничения записываются в виде:

$$C_{li} + p_{li}^0(1 - x_{s_{li},l}) + p_{li}^1 x_{s_{li},l} \leq C_{l,i+1},\ l \in \mathcal{M},\ i = 1, \ldots, n_l - 1, \quad (206)$$

$$C(O_v^j) \geq C_{L_{vj},k_{vj}^0} - M(x_{s_{vj},L_{vj}}),\ j \in \mathcal{J},\ v = 1, \ldots, k_j - 1, \quad (207)$$

$$C(O_v^j) \geq C_{L_{vj},k_{vj}^1} - M(1 - x_{s_{vj},L_{vj}}),\ j \in \mathcal{J},\ v = 1, \ldots, k_j - 1, \quad (208)$$

$$C(O_v^j) + p_{v+1,j} \leq C(O_{v+1}^j),\ j \in \mathcal{J},\ v = 1, \ldots, k_j - 1, \quad (209)$$

$$C(O_v^j) \geq 0,\ j \in \mathcal{J},\ v = 1, \ldots, k_j - 1, \quad (210)$$

$$C_{li} \geq 0,\ l \in \mathcal{M},\ i = 1, \ldots, n_l, \quad (211)$$

$$x_{sl} \in \{0, 1\},\ l \in \mathcal{M},\ s = 1, \ldots, q(l). \quad (212)$$

Ограничение (206) устанавливает связь между моментами завершения операций в последовательных позициях. Условия (202) определяют время завершения операций через моменты завершения в позициях. Неравенство (203) гарантирует выполнение отношений предшествования

Задача	Мощность подмножеств
Одномашинные системы	
$1\|\|\sum_j w_j U_j$	$\|X^i\| = 2$
$1\|r_j\|L_{\max}$	$\|X^i\| \leq 2$
$1\|r_j\|\sum_j U_j$	$\|X^i\| \leq 2$
$1\|r_j\|\sum_j T_j$	$\|X^i\| \leq 2$
$1\|C_j \leq d_j\|\sum_j w_j C_j$	$\|X^i\| \leq 2$
$1\|r_j\|\sum_j C_j$	$\|X^i\| \leq 2$
$1\|l_{ij}\|C_{\max}$	$\|X^i\| \leq 2$
$1\|\|\sum_j w_j T_j$	$\|X^i\| \leq 2$
Многомашинные системы	
$2\|\|\gamma \in \{C_{\max},\ L_{\max},\ \sum_j U_j,\ \sum_j T_j\}$	$\|X^i\| = 2$
$J2\|\|C_{\max}$	$\|X^i\| \leq 2$
$J2\|\|\gamma \in \{L_{\max},\ \sum_j U_j,\ \sum_j T_j\}$	$\|X^i\| \leq 2$
$F3\|Miss-Oper\|C_{\max}$	$\|X^i\| \leq 2$
$F3\|Miss-Oper\|\gamma \in \{L_{\max},\ \sum_j U_j,\ \sum_j T_j\}$	$\|X^i\| \leq 2$
$F3\|\|C_{\max}$	$\|X^i\| \leq 2$
$F2\|\|\gamma \in \{L_{\max},\ \sum_j U_j,\ \sum_j T_j\}$	$\|X^i\| \leq 2$
$F2\|\|\sum_j C_j$	$\|X^i\| \leq 2$
$O3\|\|\gamma \in \{C_{\max},\ L_{\max},\ \sum_j U_j,\ \sum_j T_j\}$	$\|X^i\| = 1$
$O2\|Miss-Oper\|\gamma \in \{C_{\max},\ L_{\max}\}$	$\|X^i\| \leq 2$
$O2\|Miss-Oper\|\gamma \in \{\sum_j U_j,\ \sum_j T_j,\ \sum_j C_j\}$	$\|X^i\| \leq 2$

Таблица 2 — NP-трудность задач с запретами работ.

между операциями. Ограничения (210)–(212) задают область определения переменных. Критерии могут быть сформулированы как в предыдущем случае. Модель (206)–(212) содержит $O(mq_{\max})$ булевых переменных и $O(mn_{\max} + nk_{\max})$ вещественных переменных, где $q_{\max} = \max_{l \in \mathcal{M}} q(l)$. Заметим, что количество булевых переменных будет значительно меньше по сравнению с предыдущей моделью для малого числа циклов в двудольных графах машин.

Для произвольных систем с запретами на выполнение работ в позициях машин модели частично целочисленного программирования могут быть записаны аналогичным образом.

Таблица 2 содержит сводные результаты по NP-трудным задачам составления расписаний с запретами работ в позициях машин.

Глава 4

Оптимальная рекомбинация в эволюционных алгоритмах для задач на перестановках

В четвертой главе проводится теоретическое и экспериментальное исследование задачи оптимальной рекомбинации, возникающей в эволюционных алгоритмах для задач составления расписаний, и показывается целесообразность ее использования наряду с механизмами перезапуска и адаптации.

1. Предлагаются эволюционные алгоритмы с оптимизированными операторами, которые показывают конкурентоспособные результаты на тестовых примерах известных библиотек.

2. Экспериментально исследуется перспективность использования оптимальной рекомбинации в составе эволюционных алгоритмов и показывается ее статистически значимое превосходство над рандомизированными аналогами.

3. Разрабатываются стратегии управления популяцией при использовании оптимизированных операторов и строятся схемы алгоритма локального поиска с применением оптимальной рекомбинации.

Результаты четвертой главы опубликованы в работах [4, 99, 101, 103, 247, 248, 250, 252].

4.1 Эволюционный алгоритм с оптимальной рекомбинацией

Гибридный эволюционный алгоритм (генетический локальный поиск, меметический алгоритм) представляет собой алгоритмическую гибридизацию популяционного метода поиска, такого как генетический алгоритм (ГА), например, и одной или нескольких техник локального поиска

и/или проблемно-ориентированных конструктивных эвристик (см., например [8, 195]). Меметические алгоритмы являются наиболее известными представителями методов из области меметических вычислений [194], которые возникли благодаря комбинации методов из нескольких областей компьютерных наук (эволюционные вычисления, многокритериальная оптимизация, машинное обучение, аппроксимационные алгоритмы и др.), а также переносу методов из других наук, таких как биология, социология и физика.

Локальные улучшения могут быть включены на разных этапах эволюционного алгоритма (ЭА). Возможно применение улучшений на этапе инициализации, постобработки или уточнения, а также процедур, переплетенных с популяционными эволюционными операторами. Также меметические алгоритмы могут быть построены в форме кооперируемых мультипопуляций.

Согласование алгоритмических компонентов в современных меметических алгоритмах часто осуществляется посредством адаптивных правил на основе оценки разнообразия, пригодности или расстояния между особями, адаптивных гиперэвристик, самоадаптивных и коэволюционных схем, а также методов обучения с подкреплением [194, 196].

В диссертации адаптивный гибридный эволюционный алгоритм с оптимальной рекомбинацией апробируется на классических NP-трудных задачах составления расписаний и маршрутизации (минимизация суммарного временного запаздывания на одной машине, задача коммивояжера, задача составления расписаний выполнения заказов клиентов), где имеется ряд различных известных метаэвристик для сравнения, и на задачах первых двух глав с учетом расхода энергии в виде ограничения или критерия.

Предлагаемый гибридный эволюционный алгоритм с оптимальной рекомбинацией основан на стационарной схеме воспроизведения (см. алгоритм 7). Каждая особь соответствует пробному решению задачи (фенотип), представленному в алгоритме как перестановка (генотип). Элементами генотипа выступают гены. В роли функции пригодности, как правило, выступает целевая функция.

Число особей в популяции обозначается как N и остается постоянным

в процессе работы алгоритма.

Алгоритм 7 Гибридный эволюционный алгоритм со стационарной схемой воспроизведения.

1: Построить начальную популяцию.

2: Пока не сработал критерий остановки, выполнять шаги:

2.1 Выбрать два родительских решения π^1 и π^2.

2.2 Применить оператор мутации к родительским решениям.

2.3 Создать потомка π', применив оператор кроссинговера к π^1 и π^2.

2.4 Потомок π' заменяет худшую особь в популяции.

3: Выполнить локальное улучшение особей в финальной популяции.

Для баланса эволюционного поиска и локального поиска используется несколько правил перезапуска. Суть классического правила перезапуска заключается в следующем: гибридный ЭА сначала выполняет $N/2$ итераций без ограничений, после чего алгоритм перезапускается, как только текущее число итераций становится вдвое больше номера итерации, на которой было получено крайнее улучшение рекорда. В некоторых случаях лучше себя показывает схема перезапуска, когда алгоритм перезапускается через фиксированное число итераций без улучшения рекорда, и схема перезапука, основанная на переписи Шнабеля.

Рассмотрим классические операторы кроссинговера, мутации и селекции для задач на перестановках.

Одноточечный кроссинговер (1PX) [143]. Случайно выбирается одна позиция перестановки, и элементы из левой части наследуются от одного из родителей. Остальные элементы размещаются в порядке, соответствующем второму родителю (рис. 32).

Циклический кроссинговер (CX) [143]. Цикл определяется соответствующими позициями подмножества элементов, общих для обоих родителей. Элементы в цикле одного родителя копируются на те же позиции потомка. Остальные элементы берутся от другого родителя (рис. 33).

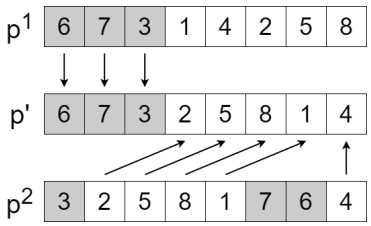


Рисунок 32 — Одноточечный кросинговер.

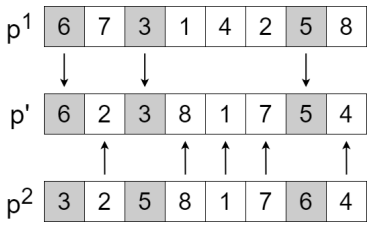


Рисунок 33 — Циклический кроссинговер.

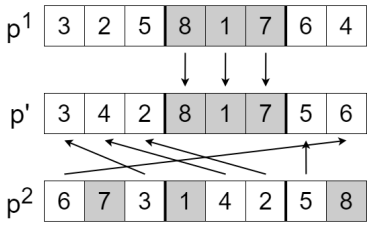


Рисунок 34 — Порядковый кроссинговер.

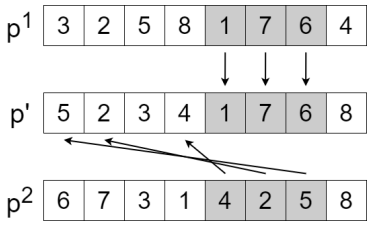


Рисунок 35 — Кроссинговер с частичным отображением.

Порядковый кроссинговер (OX) [143]. Случайно выбираются две позиции перестановки, и элементы между этими позициями копируются от одного родителя на те же позиции потомка. Остальные элементы размещаются в том же порядке, что и у другого родителя, начиная со второй выбранной позиции (рис. 34).

Оператор кроссинговера с частичным отображением (PMX) [143]. Случайно выбираются две позиции перестановки. Элементы между позициями копируются от одного родителя на те же позиции потомка. Остальные элементы берутся от другого родителя с заменой уже скопированных элементов по правилу парных обменов и сохранением элементов на местах, где это возможно (рис. 35).

Типичными операторами мутации являются мутации сдвига и обмена. Мутация сдвига перемещает один случайно выбранные элемент в случайно выбранную позицию (рис. 36). Мутация обмена меняет местами два случайно выбранных элемента (рис. 37).

В качестве оператора селекции используется ранговая селекция и турнирная селекция [113]. При ранговой селекции все особи популяции ранжируются путем присвоения номеров от 1 до n в соответствии с невозрастанием функции пригодности. Ранг перестановки i обозначается как r_i, вероятность ее выбора в операторе равна $(r_i / \sum r_k)$. Турнирная селекция случайно извлекает s перестановок и возвращает наилучшую (по

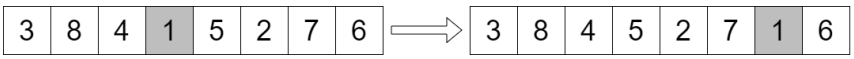


Рисунок 36 — Мутация сдвига.

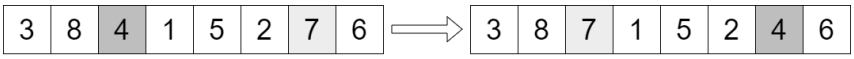


Рисунок 37 — Мутация обмена.

значению функции пригодности).

Производительность эволюционных алгоритмов существенно зависит от оператора кроссинговера, в котором компоненты родительских решений комбинируются для построения потомков, как, например, в указанных выше операторах. *Задача оптимальной рекомбинации* (ЗОР) – это подзадача, заключающаяся в нахождении наилучшего возможного потомка в результате скрещивания двух допустимых родительских решений при условии, что рекомбинация удовлетворяет свойству передачи генов согласно терминологии [204].

Задача оптимальной рекомбинации (ЗОР) для задачи на перестановках формулируется следующим образом. Даны два родительских решения π^1 и π^2. Требуется найти перестановку π' такую, что:

(I) $\pi'_i = \pi^1_i$ или $\pi'_i = \pi^2_i$ для всех $i = 1, \ldots, n$;

(II) π' имеет оптимальное значение целевой функции среди всех перестановок, удовлетворяющих условию (I).

Оптимальную рекомбинацию можно рассматривать улучшающий шаг в большой окрестности, определяемой двумя родительскими решениями. Альтернативно, её можно рассматривать как окрестность вокруг первого родительского решения, где множество соседей определяется комбинацией компонентов от обоих родителей. Во многих примерах можно видеть, что окрестность, определённая таким образом, имеет экспоненциальный размер.

Соответствующие операторы рекомбинации, в которых решается ЗОР, называются оптимизированными операторами кроссинговера.

Первые ГА, использующие ЗОР в контексте задачи о максимальном независимом множестве (или её эквивалента, задачи о максимальной клике), были предложены в [19, 29]. Здесь используется тот факт, что

подзадачи, определяемые ЗОР, могут быть эффективно решены. В [244] предложены ГА, решающие ЗОР для нескольких задач на перестановках. Экспериментальные результаты показали, что операторы скрещивания, решающие ЗОР точно [29]) или приближённо, могут быть эффективно применены в гибридных ЭА, и в меметических алгоритмах в частности [29, 52, 70, 71, 80, 85, 98, 103, 212, 244].

Отметим, что формулировка ЗОР зависит от представления решений, и полезность решения ЗОР в рамках ЭА существенно зависит от этого представления. Например, допустимые решения задачи коммивояжёра могут быть представлены с использованием кодировки на основе позиций или кодировки на основе свойства смежности. Недостатком первого подхода к решению задачи коммивояжера является то, что ЗОР с представлением на основе позиций не направлена на сохранение дуг из родительских решений. Однако наследование перспективных последовательностей дуг влияет на результат, поскольку длины дуг явно присутствуют в целевой функции.

Некоторые известные комбинаторные оптимизационные задачи порождают полиномиально разрешимые ЗОР. В частности, если используется стандартное бинарное представление решений, то с помощью полиномиальных преобразований ЗОР для ряда задач могут быть сведены к ЗОР для задачи о максимальной взвешенной клике [98], которая разрешима за время $O(n^3)$, как показано в [29].

Однако NP-трудные случаи ЗОР встречаются очень часто. Известны два основных подхода к доказательству NP-трудности ЗОР. Первый подход заключается в использовании известного сведения из некоторой задачи распознавания P к оптимизационной задаче P' и построении аналогичного сведения из некоторого NP-трудного, но хорошо структурированного специального случая P_0 задачи P к ЗОР, порождаемой задачей P'. Этот метод был применён для доказательства NP-трудности ЗОР в случае задачи о рюкзаке с бинарным представлением решений [96], симметричной и общей задач коммивояжера с кодировкой на основе свойства смежности [97, 98], одномашинной задачи составления расписаний с переналадками и задачи flow-shop с представлением решений на основе позиций [9, 98]. В этих доказательствах роль ограниченной задачи P_0

играют ограниченные версии задачи о гамильтоновом цикле в решетчатых графах, задачи о вершинном покрытии и задачи коммивояжера с ограничениями на степени вершин, соответственно.

Во втором подходе строится полиномиальное сведение NP-трудной ЗОР к рассматриваемой ЗОР [98]. Например, в [99], NP-трудная ЗОР для ассиметричной задачи коммивояжёра полиномиально сводится к ЗОР для задачи составления расписаний из работы [99].

Для задач на перестановках существуют различные алгоритмы решения NP-трудных ЗОР и их вариаций в случае кодировки на основе позиций и на основе свойства смежности (например, динамическое программирование [244], перебор совершенных паросочетаний в специальном двудольном графе [9], разбиение вершин графа на независимые компоненты [229], аналог рекурсивного алгоритма Эпштейна для кубических графов [103]). Экспериментальные исследования показывают, что оптимальная рекомбинация может успешно использоваться в ЭА (см., например, [29, 244]).

Для решения рассматриваемых ЗОР будем в большинстве случаев использовать алгоритм, представленный в разделе 3.1.2, где предписания соответствуют элементам в позициях перестановок родительских решений. Построим двудольный граф $G = (X_n, X, U)$ с двумя долями множества вершин X_n (позиции перестановки), X (элементы перестановки) равных мощностей и множеством рёбер

$$U = \{\{i, x\} : i \in X_n,\ x \in \{\pi_i^1,\ \pi_i^2\}\}.$$

Алгоритм перебирает совершенные паросочетания графа G (см. раздел 3.1.2), формируя их из максимальных паросочетаний в циклах и особых ребер (определяются случаями $\pi_i^1 = \pi_i^2$). Каждому совершенному паросочетанию ставится в соответствие допустимое решение ЗОР и вычисляется значение целевой функции. Число допустимых решений ЗОР составляет $2^{q(G)}$, где $q(G) \leq \lfloor \frac{n}{2} \rfloor$ – число циклов в графе G.

Покажем, что для «почти всех» пар родительских решений выполняется неравенство $q(G) \leqslant 1.1 \cdot \ln(n)$, т. е. «почти все» индивидуальные задачи оптимальной рекомбинации имеют не более чем n допустимых решений. Граф $G = (X_n, X, U)$ будем называть «хорошим», если

$q(G) \leqslant 1.1 \cdot \ln(n)$, и «плохим» – в противном случае. Пару родительских решений π^1 и π^2 в задаче оптимальной рекомбинации будем называть «хорошей», если соответствующий ей граф $G = (X_n, X, U)$ является «хорошим», и «плохой» – в противном случае.

Обозначим через $\bar{\mathcal{G}}_n$ ($\bar{\mathcal{R}}_n$) множество «хороших» графов (множество «хороших» пар родительских решений), $\tilde{\mathcal{G}}_n$ ($\tilde{\mathcal{R}}_n$) – множество «плохих» графов (множество «плохих» пар родительских решений). Положим $\mathcal{G}_n = \bar{\mathcal{G}}_n \cup \tilde{\mathcal{G}}_n$, $\mathcal{R}_n = \bar{\mathcal{R}}_n \cup \tilde{\mathcal{R}}_n$.

Также введем следующие вспомогательные обозначения:
S_l – множество подстановок множества $\{1, \ldots, l\}$, не содержащих циклов единичной длины;
$\bar{S}_l$ – множество подстановок из S_l с числом циклов, не превосходящим $1.1 \cdot \ln(l)$.

Положим $\tilde{S}_l = S_l \backslash \bar{S}_l$. Из результатов [11] следует

Лемма 27 $|\tilde{S}_l|/|\bar{S}_l| \longrightarrow 0$ *при* $l \to \infty$.

Используя данную лемму, докажем следующую теорему.

Теорема 66 $|\bar{\mathcal{R}}_n|/|\mathcal{R}_n| \longrightarrow 1$ *при* $n \to \infty$.

Доказательство. 1. Аналогично [11] оценим величины $|\bar{\mathcal{G}}_n|$ и $|\tilde{\mathcal{G}}_n|$. Для этого поставим в соответствие произвольной подстановке $\sigma \in S_l$, $l \leqslant n$, множество двудольных графов $\mathcal{G}_n(\sigma) \subset \mathcal{G}_n$ следующим образом. Выделим произвольные $n - l$ ребер в качестве особых. Непомеченые вершины $\{i_1, i_2, \ldots, i_l\} \subset X_n$ левой доли, где $i_j < i_{j+1}$, $j = 1, \ldots, l-1$, разобьем на $\xi(\sigma)$ циклов графа, где $\xi(\sigma)$ – число циклов в подстановке σ, таким образом, чтобы вершины с номерами $\{i_{t_1}, i_{t_2}, \ldots, i_{t_r}\}$ принадлежали одному циклу графа в том и только том случае, когда элементы $(t_1, t_2, \ldots, t_r)$ составляют цикл в подстановке σ. В этом случае потребуем, чтобы для любой пары вершин i_{t_j}, $i_{t_{j+1}}$ существовала смежная с ними вершина из правой правой доли X графа, $j = 1, \ldots, r$, $i_{t_{r+1}} = i_{t_1}$.

Существует $n!$ способов выбора соответствия между вершинами левой и правой долей, поэтому количество различных графов из класса $\mathcal{G}_n(\sigma)$, $\sigma \in S_l$, $l \leqslant n$, равно $|\mathcal{G}_n(\sigma)| = C_n^l \frac{n!}{2^{\xi_1(\sigma)}}$, где $\xi_1(\sigma)$ – число циклов длины два в подстановке σ. Деление на $2^{\xi_1(\sigma)}$ обусловлено тем, что

для любого цикла графа, соответствующего циклу длины два в подстановке σ, существует два эквивалентных способа нумерации вершин его правой доли.

Пусть $\sigma = c_1 c_2 \dots c_{\xi(\sigma)}$ – подстановка из множества S_l, представленная в виде циклов c_i, $i = 1, \dots, \xi(\sigma)$, и c_j – произвольный цикл подстановки σ, содержащий не менее трех элементов, $1 \leqslant j \leqslant \xi(\sigma)$. Преобразуем подстановку σ в σ^1 путем замены цикла c_j обратным:

$$\sigma^1 = c_1 c_2 \dots c_{j-1} c_j^{-1} c_{j+1} \dots c_{\xi(\sigma)}. \tag{213}$$

Ясно, что подстановка σ^1 порождает то же множество графов из класса $\mathcal{G}_n$, что и подстановка σ. Таким образом, для любых двух подстановок σ^1 и σ^2 из множества S_l, $l \leqslant n$, порождаемые ими множества графов из класса $\mathcal{G}_n$ совпадают, если одну из них можно получить из другой путем применения нескольких преобразований вида (213), и не пересекаются – в противном случае. Кроме того, $\mathcal{G}_n(\sigma^1) \cap \mathcal{G}_n(\sigma^2) = \varnothing$, если $\sigma^1 \in S_{l_1}$, $\sigma^2 \in S_{l_2}$, $l_1 \neq l_2$.

Если $\sigma \in \bar{S}_l$, $l \leqslant n$, то $\mathcal{G}_n(\sigma) \subseteq \bar{\mathcal{G}}_n$, а если $\sigma \in \tilde{S}_l$, то при $l < n$ не исключается возможность того, что $\mathcal{G}_n(\sigma) \subseteq \bar{\mathcal{G}}_n$. С учетом вышесказанного имеем

$$|\bar{\mathcal{G}}_n| \geqslant \sum_{l=2}^{n} \sum_{\sigma \in \bar{S}_l} C_n^l \frac{n!}{2^{\xi_1(\sigma)} 2^{\xi(\sigma)-\xi_1(\sigma)}} = \sum_{l=2}^{n} \sum_{\sigma \in \bar{S}_l} C_n^l \frac{n!}{2^{\xi(\sigma)}}, \tag{214}$$

$$|\tilde{\mathcal{G}}_n| \leqslant \sum_{l=\lfloor 1.1 \cdot \ln(n) \rfloor}^{n} \sum_{\sigma \in \tilde{S}_l} C_n^l \frac{n!}{2^{\xi_1(\sigma)} 2^{\xi(\sigma)-\xi_1(\sigma)}} = \sum_{l=\lfloor 1.1 \cdot \ln(n) \rfloor}^{n} \sum_{\sigma \in \tilde{S}_l} C_n^l \frac{n!}{2^{\xi(\sigma)}}. \tag{215}$$

2. Оценим величины $|\bar{\mathcal{R}}_n|$, $|\tilde{\mathcal{R}}_n|$ и докажем справедливость утверждения теоремы. Напомним, что каждый граф $\bar{G} \in \mathcal{G}_n(\sigma)$, $\sigma \in S_l$, $l \leqslant n$, имеет $\xi(\sigma)$ циклов. Обозначим через $w^j = \{(i_1, x^{i_1}), (i_2, x^{i_2}), \dots, (i_{m_j}, x^{i_{m_j}})\}$, $\bar{w}^j = \{(i_1, \bar{x}^{i_1}), (i_2, \bar{x}^{i_2}), \dots, (i_{m_j}, \bar{x}^{i_{m_j}})\}$ максимальные паросочетания, на которые разбиваются ребра цикла j графа $\bar{G}$, $j = 1, \dots, \xi(\sigma)$. Тогда в любой задаче оптимальной рекомбинации, порождающей граф $\bar{G}$, либо $\pi^1_{i_m} = x^{i_m}$, $\pi^2_{i_m} = \bar{x}^{i_m}$, $m = 1, \dots, m_j$, либо $\pi^1_{i_m} = \bar{x}^{i_m}$, $\pi^2_{i_m} = x^{i_m}$, $m = 1, \dots, m_j$, для всех $j = 1, \dots, \xi(\sigma)$. Следовательно, каждый двудольный граф из класса $\mathcal{G}_n(\sigma)$ соответствует $2^{\xi(\sigma)}$ парам родительских

решений в задаче оптимальной рекомбинации (пары родителей $\pi^1 = a$, $\pi^2 = b$ и $\pi^1 = b$, $\pi^2 = a$ считаются различными). Учитывая (214) и (215), получаем:

$$|\bar{\mathcal{R}}_n| \geqslant \sum_{l=2}^{n} \sum_{\sigma \in \bar{S}_l} C_n^l \frac{n!}{2^{\xi(\sigma)}} 2^{\xi(\sigma)} \geqslant \sum_{l=\lfloor 1.1 \cdot \ln(n) \rfloor}^{n} |\bar{S}_l| C_n^l n!, \tag{216}$$

$$|\tilde{\mathcal{R}}_n| \leqslant \sum_{l=\lfloor 1.1 \cdot \ln(n) \rfloor}^{n} \sum_{\sigma \in \tilde{S}_l} C_n^l \frac{n!}{2^{\xi(\sigma)}} 2^{\xi(\sigma)} = \sum_{l=\lfloor 1.1 \cdot \ln(n) \rfloor}^{n} |\tilde{S}_l| C_n^l n!. \tag{217}$$

Теперь, полагая $\psi(n) = \max\limits_{l=\lfloor 1.1 \cdot \ln(n) \rfloor, \ldots, n} |\tilde{S}_l| / |\bar{S}_l|$, в силу соотношений (216), (217) и леммы 27 имеем

$$|\tilde{\mathcal{R}}_n| / |\bar{\mathcal{R}}_n| \leqslant \psi(n) \to 0 \text{ при } n \to \infty. \tag{218}$$

Справедливость утверждения теоремы следует из (218). $\square$

Предположим, что рассматривается задача оптимизации на перестановках, для которой значение целевой функции для заданной перестановки вычисляется за полиномиальное время. В алгоритме 7 в качестве оператора мутации используется мутация сдвига или обмена, а в качестве оператора селекции – ранговая или турнирная селекция. Тогда справедлива

Теорема 67 *Алгоритм 7 сходится почти наверное к перестановке, соответствующей оптимальному решению.*

Доказательство. Оператор воспроизведения состоит в последовательном применении мутации и оптимальной рекомбинации к родительским особям $\pi^1,\ \pi^2$.

Покажем, что для любой перестановки π'' существует положительная вероятность перехода из π'' в перестановку, соответствующую оптимальному решению, за конечное число итераций. Такой переход можно осуществить только благодаря мутации (сдвиг или обмен).

Поскольку родительские особи $\pi^1,\ \pi^2$ выбираются независимо друг от друга, причем каждая особь из текущей популяции может быть выбрана в качестве родительской, то вероятность того, что произвольная

особь из текущей популяции будет выбрана одновременно в качестве π^1 и π^2, положительна. Вероятность того, что после независимого применения мутации к π^1 и $\pi^2 = \pi^1$ будет получена одна и та же особь $\tilde{\pi}$, также положительна. Тогда результатом оптимальной рекомбинации будет генотип $\pi' = \tilde{\pi}$.

Оператор выживания позволяет сохранить полученную особь в популяции.

Далее сходимость алгоритма 7 к оптимуму почти наверное доказывается аналогично теореме 2.1 в [209]. □

4.2 Задача минимизации суммарного взвешенного временного запаздывания на одной машине

Рассмотрим следующую задачу составления расписаний множества работ $\mathcal{J} = \{1, \ldots, n\}$ на одной машине. Каждая работа $j \in \mathcal{J}$ характеризуется временем поступления r_j, длительностью выполнения p_j, весом w_j и директивным сроком d_j, к которому работу следует завершить. Машина может выполнять не более одной работы одновременно, а прерывания выполнения работ запрещены.

Пусть C_j обозначает время завершения работы $j \in \mathcal{J}$, тогда запаздывание T_j работы j вычисляется как $\max\{0; C_j - d_j\}$. Требуется построить расписание, минимизирующее суммарное взвешенное запаздывание $\sum_{j\in\mathcal{J}} w_j T_j$. На основе классической системы обозначений в теории расписаний задачу обозначается через $1|r_j, d_j, w_j|\sum_j w_j T_j$ и называется *задачей суммарного взвешенного запаздывания на одной машине (total weighted tardiness problem, TWTP)*. Эта задача возникает в различных практических приложениях, в частности в химической промышленности [235]. Кроме того, задачи составления расписаний на одной машине также имеют важную роль для исследования задач на нескольких машинах, где результаты, полученные для задач с одной машиной, часто могут быть применены к более сложным системам, таким как параллельные или специализированные машины, flow-shop, job-shop и open-shop.

Пусть $\pi = (\pi_1, \ldots, \pi_n)$ обозначает перестановку работ, где π_i – это работа в i-й позиции на машине, $i = 1, \ldots, n$. Тогда моменты завершения

вычисляются как $C(\pi_i) = C(\pi_{i-1}) + p_{\pi_i}$, а запаздывания как $T(\pi_i) := \max\{0; C(\pi_i) - d_{\pi_i}\}$ для работ в позициях $i = 1, \ldots, n$ (полагаем $C(\pi_0) := 0$). Суммарное взвешенное запаздывание для перестановки π обозначим через $T(\pi) = \sum_{i=1}^{n} w_{\pi_i} T(\pi_i)$.

Рассматриваемая задача является NP-трудной, поэтому точные методы применимы на практике только к примерам малой размерности. Следовательно, для этой задачи целесообразна разработка метаэвристик, в частности эволюционных алгоритмов.

Кратко опишем современные метаэвристики, известные для задачи $1|r_j, d_j, w_j| \sum_j w_j T_j$: методы локального поиска, алгоритмы муравьиной колонии, эволюционные алгоритмы и другие популяционные методы.

Хорошие экспериментальные результаты для $1|r_j, d_j, w_j| \sum_j w_j T_j$ продемонстрированы методами локального поиска, основанными на улучшении одного решения или популяции [45,72,76,108,112,239,245]. Это связано с «хорошими» свойствами целевой функции, которые значительно сокращают область поиска решений в окрестностях (см., например, [112]). В [17, 25, 43] разработаны конкурентоспособные алгоритмы муравьиной колонии для TWTP, которые также используют локальные улучшения.

Гибридные эволюционные алгоритмы занимают одно из ведущих мест среди метаэвристик, известных для рассматриваемой задачи. Один из первых генетических алгоритмов для TWTP был представлен в [16]. Алгоритм использует различные жадные эвристики в операторах назначения работ на позиции. Авторы применяют равномерный и одноточечный операторы кроссинговера, а также методы пертурбации решений. Различные операторы кроссинговера сравнивались в [143]. Наилучшие результаты в каноническом генетическом алгоритме (базовая схема) для TWTP продемонстрировали операторы кроссинговера, основанные на наследовании значений в позициях перестановки. L. Chaabane [56] предложил для TWTP гибридный генетический алгоритм, применяющий имитацию отжига на этапе локального улучшения, одноточечный кроссинговер и мутацию сдвига. Гибридный метод [83], сочетающий эволюционный алгоритм и процедуру динамического программирования, демонстрирует на данный момент лучшие экспериментальные результаты как по значению целевой функции, так и по времени выполнения.

В методах дифференциальной эволюции [192,226,227] для рассматриваемой задачи решение кодируется вещественным вектором длины n, а соответствующая перестановка строится с помощью правила наименьшего значения для очередности позиций, где работы сортируются в порядке возрастания значений компонент вектора. На итерациях используются методы локального поиска на основе операторов сдвига и обмена и их комбинации для улучшения результатов.

Алгоритм пчелиного роя также является популяционным методом, основанным на поведении пчел при поиске нектара и их активностью в улье. В алгоритме пчелиного роя из [217] для рассматриваемой задачи решения представляются вещественными векторами как и в указанных ранее алгоритмах дифференциальной эволюции. В механизм поиска вовлечены операторы из непрерывной оптимизации и методы локального улучшения.

Целью текущего исследования является сравнение оптимизированного кроссинговера, основанного на наследовании значений в позициях перестановок, с другими известными методами локальной оптимизации.

4.2.1 Операторы

Используемый ЭА основан на схеме алгоритма 7. Начальная популяция включает решения, сгенерированные случайным образом или с помощью конструктивных эвристик. Основная конструктивная эвристика следующая: на каждом шаге выбирается случайная работа из оставшихся. Этой работе назначается позиция, которая дает наилучшее значение целевой функции для текущего частичного решения. Также популяция содержит перестановки, соответствующие неубывающему порядку директивных сроков d_j, невозрастающему порядку w_j/p_j и невозрастающему порядку $(w_j/p_j) \cdot exp\{-\max\{0,(d_j-p_j)\}/(n*p_{aver})\}$, где $p_{aver} = \sum_{j=1}^{n} p_j/n$.

Особи финальной популяции улучшаются процедурой локальной оптимизации, основанной на операциях *обмена* и *сдвига*. Для сокращения времени выполнения локального поиска используются стратегии, предложенные в [112] специально для случая задачи $1|r_j,d_j,w_j|\sum_j w_jT_j$. В частности, заведомо неперспективные шаги исключаются, и рассматри-

вается только часть окрестности путем перемещения работ с позиций, находящихся на расстоянии не более $20\%n$ друг от друга.

Используется два оператора мутации, которые выполняют случайный шаг в окрестности путем обмена или сдвига [112]. Операторы используются для мутации с равной вероятностью. В операторе кроссинговера решается задача оптимальной рекомбинации (I)-(II) с вероятностью p_{cross} (этот оператор кроссинговера называется *OCX*, Optimized Cycle Crossover), и порядковый кроссинговер *OX* (Ordered Crossover) [143] с вероятностью $1 - p_{cross}$. Вводится ограничение на время решения ЗОР в операторе кроссинговера и в качестве результата берется текущий рекорд. Перезапуск алгоритма вызывается по классическому правилу.

Можно доказать NP-трудность ЗОР для двух случаев аналогично доказательству теорем 46 и 47: когда веса работ различны и все работы поступают в момент времени $r_j = 0$ и когда моменты поступления работ различны.

Теорема 68 *Задача оптимальной рекомбинации (I)-(II) для* $1|r_j = 0, d_j, w_j|\sum_j w_j T_j$ *является NP-трудной даже в случае, когда все работы, кроме двух, имеют одинаковые директивные сроки.*

Теорема 69 *Задача оптимальной рекомбинации (I)-(II) для задачи составления расписаний* $1|r_j, d_j|\sum_j T_j$ *является NP-трудной даже в случае, когда только одна работа имеет индивидуальный момент поступления и директивный срок.*

4.2.2 Вычислительный эксперимент

В этом разделе предложенный эволюционный алгоритм HEA-OCX сравнивается с эвристиками, которые ранее продемонстрировали свою конкурентоспособность при решении TWTP.

Генетический алгоритм (GA) из [16] — это популяционный алгоритм с локальными улучшениями, где начальная популяция генерируется случайным образом; на итерациях фиксируется некоторое случайно выбранное подмножество работ в выбранном решении, и локальные улучшения применяются к остальным работам с адаптацией приоритетов (тестировался на компьютере с Pentium II 400MHz, 96Mb).

Популяционный алгоритм локального поиска с чередующимися окрестностями (PVNS) [239] использует схему поиска с чередующимися окрестностями на популяции с процедурой встряски, основанной на окрестностях сдвига и обмена, и такими методами локального поиска, как поиск с переменной глубиной, Path-relinking и поиск с запретами; начальная популяция строится с помощью известной эвристики NEH со случайными последовательностями работ и детерминированными правилами их размещения (тестировался на компьютере с Pentium IV 3.0GHz, 512Mb).

Локальный поиск с чередующимися окрестностями (m-VNS) из [73] соответствует базовому принципу локального поиска с чередующимися окрестностями, где начальное решение генерируется с использованием правила рулетки; в локальном поиске используются классические окрестности сдвига и обмена и их составные версии с сериями независимых шагов; проблемно-ориентированные операции встряски и сопоставления применяются для выбора окрестностей на итерациях (тестировался на компьютере с Intel Core 2.3GHz, 2Gb).

Гибридный эволюционный алгоритм (HEA-DS) из [83] представляет собой генетический алгоритм со стационарной схемой воспроизведения, где начальная популяция генерируется случайным образом, кроссинговер, основанный на наследовании значений в позициях перестановок, и мутация обмена составляют операторы воспроизведения. Динамическое программирование (Dynasearch), основанное на операциях обмена, применяется на этапе инициализации и итерациях ЭА для локального улучшения решений (тестировался на компьютере с Xeon E5440 2.83GHz, 16Gb). Отметим, что Dynasearch – это алгоритм поиска в окрестности, где экспоненциально большие окрестности исследуются с использованием динамического программирования.

Используются тестовые примеры TWTP из библиотеки OR-Library (http://people.brunel.ac.uk/ mastjjb/jeb/orlib/wtinfo.html). Количество работ в сериях составляет 40, 50 и 100. Каждая серия содержит 125 примеров. Алгоритм HEA-OCX запускался на каждом примере 50 раз. Алгоритм останавливается, когда находит оптимальное решение или достигает максимальное число итераций (такое условие выбрано для сравнения результатов с известными результатами из [16, 73, 83, 239]). Эксперимент

алгоритм	время	n_{hit}	n_{opt}	Δ_{aver}
GA	29.11	–	100%	0
PVNS	6.19	100%	100%	0
m-VNS	0.908	100%	100%	–
HEA-OCX	0.05	100%	100%	0
HEA-DS	0.003	100%	100%	0

Таблица 3 — Результаты для серии с $n = 40$.

алгоритм	время	n_{hit}	n_{opt}	Δ_{aver}
GA	41.02	–	99.2%	0.000
PVNS	12.15	100%	100%	0
m-VNS	1.62	100%	100%	–
HEA-OCX	0.14	100%	100%	0
HEA-DS	0.006	100%	100%	0

Таблица 4 — Результаты для серии с $n = 50$.

проводился на компьютере с процессором Intel Core i3-10100F 3.60 GHz, 16 Gb.

Результаты эксперимента представлены в таблицах 3, 4 и 5. Здесь n_{hit} — средний процент запусков, где найдено оптимальное решение для тестового примера, n_{opt} — процент задач, для которых найден оптимум, Δ_{aver} — среднее относительное отклонение от оптимума. Видно, что алгоритм HEA-OCX демонстрирует конкурентоспособные результаты, находя рекорд во всех случаях за сравнимое время. Отметим, что из предыдущих исследований известно, что метод Dynasearch, основанный на окрестности обмена, демонстрирует лидирующие результаты на рассматриваемых сериях примеров с точки зрения качества решений и

алгоритм	время	n_{hit}	n_{opt}	Δ_{aver}
GA	118.97	–	66.4%	0.02
PVNS	183.47	100%	100%	0
m-VNS	5.845	99.84%	100%	–
HEA-OCX	1.5	100%	100%	0
HEA-DS	0.032	100%	100%	0

Таблица 5 — Результаты для серии с $n = 100$.

оператор	n_{bt}	n_{opt}	Δ_{aver}
$n = 40$			
OCX	–	100%	0
CX	43/125	82.51%	0.0029
OX	20/125	93.73%	0.00078
OX+LD	17/125	94.56%	0.00076
$n = 50$			
OCX	–	100%	0
CX	67/125	68.27%	0.0057
OX	41/125	84%	0.0017
OX+LD	38/125	92.88%	0.0013
$n = 100$			
OCX	–	100%	0
CX	92/125	34.59%	0.0097
OX	90/125	40.56%	0.0055
OX+LD	80/125	62.32%	0.0031

Таблица 6 — Экспериментальное сравнение четырех операторов кроссинговера.

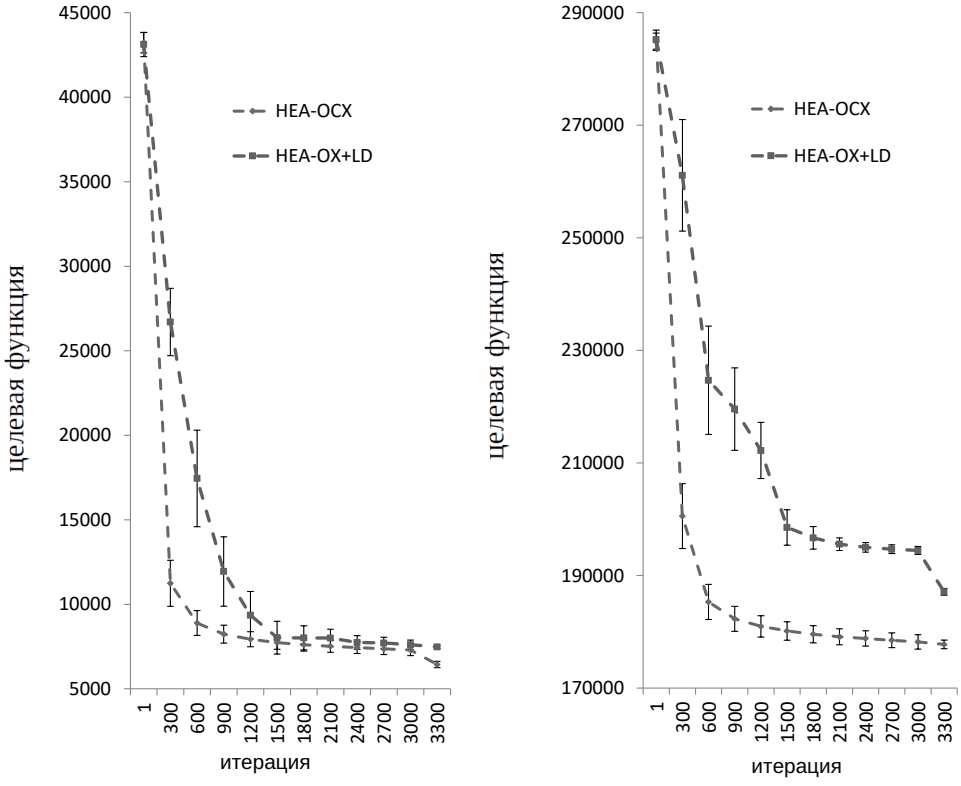


Рисунок 38 — Динамика целевой функции в ходе работы ЭА.

временных затрат на поиск решения. Алгоритм HEA-OCX показывает несколько худшие результаты, чем HEA-DS, с точки зрения времени достижения рекорда. Но демонстрируемое время также можно считать хорошим с практической точки зрения.

Также рассмотрена модификация предложенного алгоритма HEA-OCX, где вместо кроссинговера OCX используются следующие операторы рекомбинации [143]:
– циклический кроссинговер (CX);
– порядковый кроссинговер (OX);
– порядковый кроссинговер с последующим локальным спуском, основанным на окрестности сдвига (OX+LD).

Результаты сравнения различных операторов кроссинговера в ЭА при одинаковом ограничении на время вычислений представлены в таблице 6. Версии ЭА сравниваются на основе частоты нахождения оптимума за 50 запусков. Столбец n_{bt} показывает количество примеров, на которых разница между частотой нахождения оптимума HEA-OCX и его конкурента статистически значима при уровне значимости $\alpha \leq 0.05$ (используется тест из [59], см. описание в разделе 4.3.2). На основе полученных значений заключаем, что увеличение количества работ n приводит к увеличению n_{bt} на тестируемых сериях.

Для дополнительного анализа перспективности использования оптимизированного кроссинговера проведен следующий эксперимент: при фиксированном бюджете, равном 4000 итераций, ЭА с кроссинговером OCX и с кроссинговером OX с последующим локальным спуском запускался 50 раз на каждой задаче. Лучшие найденные значения целевой функции (усредненные по 50 запускам) представлены для двух примеров на рис. 38 («усы» показывают стандартное отклонение). Как можно увидеть, выбор оператора кроссинговера играет важную роль для конечного результата, однако также необходимо учитывать затраты на рекомбинацию и, возможно, использовать ограниченные или жадные версии для задач большой размерности (см. примеры в [103]).

4.3 Задача коммивояжера

Задача коммивояжера (ЗК) является одной из классических задач комбинаторной оптимизации (см., например, [107, 111, 160, 197]. Задан полный ориентированный граф G с множеством вершин $V = \{v_1, \ldots, v_n\}$, множеством дуг $A = \{(v_i, v_j) : \ v_i, v_j \in V, \ i \neq j\}$ и весами (длинами) $c(v_i, v_j) \geq 0$ дуг $(v_i, v_j) \in A$. Требуется найти гамильтонов цикл (контур) с минимальной общей длиной. Если $c(v_i, v_j) \neq c(v_j, v_i)$ хотя бы для одной пары вершин $v_i, \ v_j$, то задача называется асимметричной задачей коммивояжера (АЗК).

АЗК возникает в различных практических ситуациях, таких как задачи составления расписаний с асимметричными длительностями переналадки оборудования, управление производством продукции, распределение товаров и услуг, задачи маршрутизации транспортных средств и т.д. Из-за NP-трудности в сильном смысле АЗК [111] эвристики и метаэвристики, в частности меметические алгоритмы, занимают важное место среди методов, разработанных для этой задачи.

ЗК была одной из первых задач, к которым применялся меметический подход [107, 197]. С тех пор роль и эффект от локального поиска в меметических алгоритмах для ЗК интенсивно изучались. Общей чертой алгоритмов, предложенных в [63, 190, 228], является использование проблемно-ориентированных для задачи операторов кроссинговера. В ГА [243] для АЗК известный рандомизированный оператор кроссинговера выполняется несколько раз для одной и той же пары родительских особей, и два лучших решения считаются потомками. В меметических алгоритмах из [63, 228, 243] локальный поиск применяется для улучшения результата кроссинговера на каждой итерации ГА. В [190] локальный поиск используется только на этапе инициализации. Новые стратегии управления популяцией и специфические процедуры поддержания разнообразия популяции предложены в [190, 243]. Экспериментальные результаты [228] и [244] показывают, что операторы кроссинговера, которые приближенно решают ЗОР, могут эффективно применяться в ГА для задач на перестановках.

В данном разделе предлагается и исследуется меметический алгоритм с оптимальной рекомбинацией для решения АЗК, где представле-

ние решений основано на свойстве смежности. Алгоритм использует эвристику локального поиска с окрестностью 3-opt и проблемно-ориентированную эвристику [255] для генерации начальной популяции. Кроме того, алгоритм применяет новый оператор мутации, выполняющий случайный шаг в окрестности 3-opt или 4-opt.

Баланс между интенсивностью популяционного поиска и методами локального поиска настраивается с помощью правила перезапуска, основанного на переписи Шнабеля [102]. Перепись Шнабеля, первоначально разработанная для оценки численности популяции животных, используется здесь не для подсчета особей, а для оценки количества решений, которые могут быть посещены в текущем запуске меметического алгоритма.

4.3.1 Операторы

Используемый ЭА основан на схеме алгоритма 7. Для двух родительских особей задача оптимальной рекомбинации состоит в нахождении наилучшего возможного потомка, который может возникнуть в результате кроссинговера, где значение каждого гена копируется из одного из значений этого гена в родительских решениях.

Будем использовать свойство смежности для кодирования маршрутов АЗК. Решения кодируются векторами, где для каждой вершины указывается непосредственный предшественник. *Задача оптимальной рекомбинации* в случае представления решений на основе свойства смежности состоит в нахождении маршрута коммивояжера минимальной длины среди циклов, составленных только из дуг, присутствующих у родителей, и содержащих все общие для обоих родителей дуги. ЗОР для АЗК с представлением решений на основе свойства смежности является NP-трудной в сильном смысле [98], ЗОР также сводится к ЗК на графе с ограниченными степенями вершин и для ее решения предложен алгоритм трудоемкости $O(n2^{\frac{n}{2}})$. Алгоритм перечисляет все допустимые решения ЗОР за $O(2^{\frac{n}{2}})$ операций и требует линейное время для вычисления длины каждого маршрута. В диссертации предлагается модификация алгоритма с вычислением целевой функции за константное время, что приводит к общей вычислительной сложности $O(2^{\frac{n}{2}})$.

Будем использовать так называемую *АЗК с предписанными дугами*, которая формулируется следующим образом. Дан ориентированный граф с неотрицательными длинами (весами) дуг и заданным подмножеством *предписанных* дуг. Требуется найти гамильтонов цикл минимальной длины, содержащий все предписанные дуги. Степень вершины понимается как общее количество входящих и исходящих дуг для этой вершины. Кубический орграф – это граф, где каждая вершина имеет степень не более 3.

Пусть A_1 и A_2 – множества дуг двух родительских маршрутов. ЗОР можно рассматривать как АЗК с предписанными дугами на орграфе $G' = (V, A_1 \cup A_2)$ с множеством предписанных дуг $F' := A_1 \cap A_2$. Каждая вершина в G' имеет степень не более четырех. Заметим, что АЗК с предписанными дугами на орграфе G' эквивалентна задаче АЗК с предписаннными дугами на следующем кубическом орграфе $G'' = (V'', A'')$ с множеством предписанных дуг F'': каждая вершина степени четыре $v \in V$ заменяется парой вершин $\bar{v}, \tilde{v}$, соединенных искусственной дугой $(\bar{v}, \tilde{v})$ длины $c(\bar{v}, \tilde{v}) = 0$. Все дуги, входящие в v, теперь входят в $\bar{v}$, а все дуги, исходящие из v, теперь исходят из $\tilde{v}$. Множество F'' состоит из множества предписанных дуг F' и искусственных дуг (см. пример на рис. 39).

Лемма 28 *АЗК с предписанными дугами на кубическом орграфе* $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ *с множеством предписанных дуг* $\mathcal{F}$ *разрешима за время* $O(2^{(|\mathcal{A}|-|\mathcal{F}|)/4})$.

Доказательство леммы 28 будет представлено ниже. На орграфе G'' алгоритм из леммы 28 имеет временную сложность $O(2^{d/4})$, где $d \leq 2n$ – количество дуг, присутствующих только у одного из родителей. Это приводит к следующей теореме.

Теорема 70 *Для двух родительских решений ЗОР на графе* $G = (V, A)$ *с представлением решений на основе свойства смежности может быть решена за время* $O(2^{n/2})$.

Оптимизированный оператор кроссинговера, решающий ЗОР с представлением решений на основе свойства смежности, называется *Optimized*

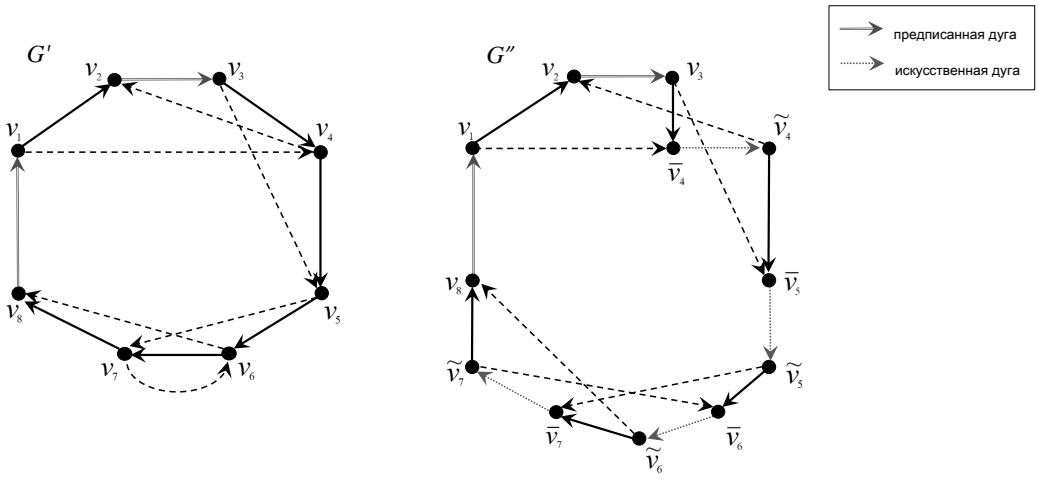


Рисунок 39 — Пример графов G' и G'' для перестановок $p_1 = (v_8, v_1, v_2, v_3, v_4, v_5, v_6, v_7)$, $p_2 = (v_8, v_4, v_2, v_1, v_3, v_7, v_5, v_6)$. Здесь (v_2, v_3), (v_8, v_1) – предписанные дуги, а $(\bar{v}_4, \tilde{v}_4)$, $(\bar{v}_5, \tilde{v}_5)$, $(\bar{v}_7, \tilde{v}_7)$ – искусственные дуги. Допустимые решения ЗОР: p_1, p_2 и $q_1 = (v_8, v_1, v_2, v_3, v_4, v_7, v_5, v_6)$, $q_2 = (v_8, v_4, v_2, v_1, v_3, v_5, v_6, v_7)$. В позиции i указана вершина, непосредственно предшествующая вершине v_i, $i = 1, \ldots, 8$.

Directed Edge Crossover (ODEC). Этот оператор можно рассматривать как детерминированный «прямой потомок» Directed Edge Crossover (DEC) [241]. В отличие от DEC, Optimized Directed Edge Crossover гарантирует выполнение свойства передачи генов [204].

Допустимые решения АЗК также могут быть представлены так, что гены кодируют вершины *в позициях* маршрута. ЗОР с таким представлением решений является NP-трудной в сильном смысле, но почти все индивидуальные задачи ЗОР разрешимы за время $O(n\log(n))$ (см. раздел 3.1.2). Недостатком такого подхода для АЗК является то, что ЗОР с представлением решений на основе кодировки значений в позициях не направлена на сохранение дуг родительских решений. Однако длины дуг явно присутствуют в целевой функции, и их сохранение способствует лучшей производительности алгоритма. Экспериментальное исследование из [99] показало, что на примерах из библиотеки TSPLIB ЭА с ЗОР с представлением решений на основе свойства смежности показывает лучшие результаты, чем ЭА с ЗОР с представлением решений на основе наследования значений в позициях перестановки.

Заметим, что в большинстве известных гибридных ГА для ЗК (см.,

например, [63, 138, 228, 243]) применяется локальный поиск на итерациях ГА. Оптимальная рекомбинация также может рассматриваться как наилучший шаг в окрестности, определяемой двумя родительскими решениями. Поэтому в предлагаемом ЭА локальный поиск используется только на этапе инициализации и для финальной популяции.

Алгоритм Д. Эппштейна. Опишем рекурсивный алгоритм Д. Эппштейна, предложенный в [93] для перечисления всех гамильтоновых циклов в графах степени три.

Пусть $G = (V, E)$ – простой граф с максимальной степенью вершин, равной трем, F – множество предписанных ребер и $U := E \backslash F$. Алгоритм Д. Эппштейна перечисляет все гамильтоновы циклы G, содержащие все предписанные ребра, как показано в описании алгоритма. На каждом шаге сначала уменьшается размер входного графа без ветвления (шаг 1). Затем выбирается ребро для ветвления и рассматривается две подзадачи (шаг 2), одну, в которой ребро обязательно входит в решение, и другую, в которой оно исключено. Эти две подзадачи решаются рекурсивно. Шаги алгоритма либо возвращают решение, либо переходят от входного графа к одному или двум кубическим графам меньшего размера.

Алгоритм. Перечисление всех гамильтоновых циклов с заданными предписанными ребрами

Вход: Простой кубический граф G с множеством предписанных ребер F.
Выход: Гамильтоновы циклы графа G, содержащие все предписанные ребра.

1. Повторять следующие шаги, пока один из них не вернет решение или ни один из них не применится:

   (a) Если G содержит вершину степени ноль или один, или если F содержит три ребра, сходящиеся в одной вершине, вернуться назад.

   (b) Если G содержит только три вершины и F состоит из гамильтонова цикла, вывести цикл и вернуться назад.

(c) Если G содержит вершину степени два, добавить инцидентные ей ребра в F.

(d) Если G содержит треугольник xyz, и ребро, не входящее в треугольник, инцидентное x, принадлежит F, добавить ребро yz в F.

(e) Если F содержит ровно два ребра $\{x, y\}$ и $\{y, z\}$, сходящиеся в некоторой вершине y, удалить из G эту вершину и любое другое инцидентное ей ребро, и заменить два ребра одним ребром $\{x, z\}$. Если это приведет к двум параллельным ребрам в G, удалить второе ребро из G.

2. Если F не пусто, пусть $\{x, y\}$ – любое ребро в F и $\{y, z\}$ – смежное ребро в $G \backslash F$. Иначе, если F пусто, пусть $\{y, z\}$ - любое ребро в графе G. Рекурсивно вызвать алгоритм для двух графов $G,\ F \cup \{y, z\}$ и $G \backslash \{y, z\},\ F$.

Теорема 71 является комбинацией результатов, доказанных в [93]. Каждый шаг алгоритма может быть реализован за константное время.

Теорема 71 *[93] Количество гамильтоновых циклов в* $G = (V, F \cup U)$ *составляет* $O(2^{|U|/4})$ *в худшем случае, и эти циклы могут быть перечислены за время* $O(2^{|U|/4})$ *и линейную память.*

Доказательство леммы 28. Рассмотрим АЗК с предписанными дугами на кубическом орграфе $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, где $\mathcal{V}$ - множество вершин, а $\mathcal{A}$ – множество дуг. Каждая вершина $v \in \mathcal{V}$ имеет степень не более трех. Каждая дуга $(v_i, v_j) \in \mathcal{A}$ ассоциирована с длиной (весом) $c(v_i, v_j) \geq 0$. Пусть $\mathcal{F}$ – множество предписанных дуг в графе $\mathcal{G}$. Предлагается точный алгоритм для нахождения гамильтонова цикла минимальной длины в $\mathcal{G}$.

Изначально граф $\mathcal{G}$ модифицируется с помощью следующей процедуры предварительной обработки. Для каждой дуги $(u, v) \in \mathcal{G}$ проверяем, есть ли дуга (v, u) в графе $\mathcal{G}$. Если такой дуги (v, u) нет в $\mathcal{A}$, то добавляем ее в граф и полагаем $c(v, u) := +\infty$. Если дуга (v, u) есть в $\mathcal{A}$, то идентифицируем следующие четыре случая:

1) если $(u, v) \in \mathcal{F}$ и $(v, u) \in \mathcal{F}$, то сообщаем, что АЗК с предписанными дугами на орграфе $\mathcal{G}$ неразрешима;

2) если $(u, v) \in \mathcal{F}$, но $(v, u) \notin \mathcal{F}$, то $c(v, u) := +\infty$;

3) если $(v, u) \in \mathcal{F}$, но $(u, v) \notin \mathcal{F}$, то $c(u, v) := +\infty$;

4) если $(u, v) \notin \mathcal{F}$ и $(v, u) \notin \mathcal{F}$, то ничего не делаем.

Если дуга (u, v) предписана в $\mathcal{G}$ после предварительной обработки, то длина (v, u) равна бесконечности. Решение модифицированной задачи может быть получено путем перечисления всех допустимых решений ЗК с предписанными ребрами на вспомогательном графе $\bar{G} = (\mathcal{V}, \bar{E})$, где пара вершин u, v соединена ребром, если эти вершины были соединены парой дуг в орграфе $\mathcal{G}$. Ребро $\{u, v\} \in \bar{E}$ считается предписанным, если $(u, v) \in \mathcal{F}$ или $(v, u) \in \mathcal{F}$. Множество предписанных ребер в $\bar{G}$ обозначается $\bar{F}$.

Учитывая множество предписанных дуг $\mathcal{F}$, гамильтонов цикл минимальной длины в $\mathcal{G}$ может быть найден с помощью представленного ниже алгоритма, который является адаптацией алгоритма, предложенного Д. Эппштейном в [93].

Алгоритм. Нахождение гамильтонова цикла минимальной длины с заданными предписанными дугами

Вход: Кубический орграф $\mathcal{G}$ и соответствующий неориентированный граф $\bar{G}$ с множеством предписанных ребер $\bar{F}$.
Выход: Гамильтонов цикл минимальной длины графа $\mathcal{G}$, содержащий все предписанные дуги, если такой цикл существует.

1. Пусть c_{bst} обозначает наилучшее найденное значение целевой функции. Изначально положим $c_{bst} := +\infty$.

2. Повторять следующие шаги, пока один из них не вернет решение или ни один из них не применится:

   (a) Если $\bar{G}$ содержит вершину степени ноль или один, или если $\bar{F}$ содержит три ребра, сходящиеся в одной вершине, вернуться

назад.

(b) Если граф $\bar{G}$ содержит только три вершины и $\bar{F}$ состоит из гамильтонова цикла, вычислить длины обходов в этом цикле в обоих направлениях, используя длины дуг из графа $\mathcal{G}$. При необходимости обновить значение c_{bst} и обход, соответствующую этому значению, и вернуться назад. Удаленные вершины исходного графа $\mathcal{G}$ будут добавлены к наилучшему найденному циклу на этапах возврата.

(c) Если $\bar{G}$ содержит вершину степени два, добавить инцидентные ей ребра в $\bar{F}$.

(d) Если $\bar{G}$ содержит треугольник xyz, и ребро, не входящее в треугольник, инцидентное x, принадлежит $\bar{F}$, добавить ребро $\{y, z\}$ в $\bar{F}$.

(e) Если $\bar{F}$ содержит ровно два ребра $\{x, y\}$, $\{y, z\}$, сходящиеся в некоторой вершине y, добавить ребро $\{x, z\}$ в $\bar{G}$ и дуги (x, z), (z, x) в $\mathcal{G}$. Если это приведет к двум параллельным ребрам в $\bar{G}$ (или параллельным дугам в $\mathcal{G}$), удалить другое ребро из $\bar{G}$ (или другую дугу из $\mathcal{G}$). Длины дуг (x, z) и (z, x) вычисляются следующим образом: $c(x, z) := c(x, y) + c(y, z)$, $c(z, x) := c(z, y) + c(y, x)$. Удалить вершину y из $\bar{G}$ и $\mathcal{G}$.

3. Если $\bar{F}$ не пусто, пусть $\{x, y\}$ – любое ребро в $\bar{F}$ и $\{y, z\}$ – смежное ребро в $\bar{G}\backslash\bar{F}$. Иначе, если $\bar{F}$ пусто, пусть $\{y, z\}$ – любое ребро в $\bar{G}$. Рекурсивно вызвать алгоритм для следующих двух наборов входных данных: $\mathcal{G}$, $\bar{G}$, $\bar{F} \cup \{y, z\}$ и $\mathcal{G}\backslash\{(y, z) \cup (z, y)\}$, $\bar{G}\backslash\{y, z\}$, $\bar{F}$.

4. Если $c_{\text{bst}} < +\infty$, то вернуть гамильтонов цикл, соответствующий c_{bst}. Иначе решения не существует.

Представленный алгоритм является рекурсивным, и при каждом вызове он либо возвращает решение, либо преобразует входной граф к двум кубическим графам меньшей размерности, выбирая дуги для ветвления. В отличие от алгоритма из [93], данный алгоритм работает с ориентированным графом. Он перечисляет все гамильтоновы циклы этого графа и находит цикл минимальной длины. Из-за структуры исходного графа $\mathcal{G}$

на шаге 2b идентифицируется не более одного цикла конечной длины на трех вершинах. Цикл соответствует маршруту в исходном графе $\mathcal{G}$, содержащему все предписанные дуги.

Когда в алгоритме происходит возврат, все удаленные на текущем этапе вершины, ребра и дуги восстанавливаются, и эти вершины добавляются к наилучшему найденному циклу, если это необходимо. Это приводит к тому же количеству дополнительных шагов, что и выполнены на текущем этапе. Таким образом, каждый шаг алгоритма имеет временную сложность $O(1)$, а общее количество шагов составляет $O(2^{(|\bar{E}|-|\bar{F}|)/4})$, как и для алгоритма из [93] (см. теорему 71). Временная сложность предварительной обработки составляет всего $O(|\mathcal{V}|)$, поэтому лемма 28 верна.

Начальная популяция. Две особи начальной популяции строятся с помощью жадной конструктивной эвристики, предложенной В. Зангом в работе [255]. Эвристика сначала решает задачу о назначениях, а затем применяет операцию склеивания подциклов для формирования допустимого маршрута двумя способами. В первом способе выбирается подцикл максимальной длины, а оставшиеся подциклы встраиваются в него. Во втором способе подциклы встраиваются один за другим согласно последовательности, предложенной в [255], начиная с самого короткого цикла. Оставшиеся $(N-2)$ особей начальной популяции строятся с помощью эвристики «случайного присоединения к контуру» [244]. Особи начальной и последней популяций улучшаются посредством применения эвристики локального поиска, основанного на 3-opt окрестности.

В качестве оператора селекции используется s-турнирная селекция, которая случайным образом извлекает s особей из текущей популяции и выбирает лучшую среди них.

Мутация применяется к каждому из родительских решений с вероятностью p_{mut}, которая является настраиваемым параметром алгоритма. Реализовано два оператора мутации, которые выполняют случайный спуск в 3-opt или 4-opt окрестности (см. рис. 40). В операторе скрещивания решается задача оптимальной рекомбинации с помощью представленного ранее алгоритма.

Эвристика локального поиска. Окрестность k-opt в задаче коммивояжера состоит из всех гамильтоновых контуров, получающихся из данного маршрута заменой k дуг. Используемая в ЭА эвристика локального поиска основана на просмотре подмножества контуров в окрестности 3-opt.

Предпринимается попытка улучшить заданный маршрут путем замены трех дуг. В примере, представленном слева на рис. 40, дуги (v_{i_1}, v_{i_2}), (v_{i_4}, v_{i_3}) и (v_{i_6}, v_{i_5}) удаляются, а дуги (v_{i_1}, v_{i_3}), (v_{i_4}, v_{i_5}) и (v_{i_6}, v_{i_2}) добавляются. В качестве кандидатов для (v_{i_1}, v_{i_2}) рассматриваются все дуги текущего маршрута в порядке убывания их весов. Варианты для вершины v_{i_3} достаточно искать только среди тех вершин, которые ближе к вершине v_{i_1} чем v_{i_2}. Поэтому для каждой вершины v предварительно формируется список из остальных вершин в порядке возрастания весов дуг от вершины v. Кандидаты на вершину v_{i_3} просматриваются согласно списку для v_{i_1} пока не будет достигнута вершина u, для которой $c_{v_{i_1},u} \geq c_{v_{i_1},v_{i_2}}$. Более того, только $\lceil 0.2n \rceil$ ближайших вершин хранится в отсортированном списке каждой вершины, что позволяет уменьшить время работы и объем используемой памяти в алгоритме (см., например, [138]). Отметим, что при добавлении дуги (v_{i_1}, v_{i_3}) образуется замкнутый контур C. Среди всех вершин этого контура в качестве v_{i_5} выбирается вершина, приводящая после замены трех дуг к наиболее короткому маршруту (вершины v_{i_4} и v_{i_6} идентифицируются однозначно). Эвристика локального поиска останавливается, если никакая замена трех дуг не приводит к улучшению текущего маршрута. В противном случае описанная процедура повторяется для нового контура.

Для уменьшения времени работы представленной эвристики локального поиска используются такие известные стратегии как «don't look bits» (не просматривай вершины, приводившие к ухудшению) и «first improving move» (переходи в первый улучшающий маршрут) из работы [138].

Операторы мутации. Реализовано два оператора мутации, которые выполняют случайный шаг в окрестности 3-opt или 4-opt. Каждый раз один из операторов используется для мутации с равной вероятностью.

Первый оператор мутации выполняет шаг в окрестности 3-opt. Сначала случайным образом выбирается дуга (v_{i_1}, v_{i_2}) среди всех дуг данного маршрута. Затем дуга (v_{i_1}, v_{i_3}) идентифицируется с использованием следующей идеи из [140]. Для каждой возможной дуги (v_{i_1}, u) вычисляем $F(u) = c_{v,u} + |C(u)| \cdot c_{aver}$, где v – непосредственный предшественник u в данном маршруте, $|C(u)|$ – длина контура, созданного u, и c_{aver} – средний вес дуг в графе G. Затем дуга (v_{i_1}, v_{i_3}) выбирается случайным образом с равномерным распределением среди 50% дуг (v_{i_1}, u) с наибольшим значением $F(u)$. Причиной учета значения $|C(u)|$ при выборе (v_{i_1}, v_{i_3}) является то, что чем больше значение $C(u)$, тем больше вариантов для (v_{i_4}, v_{i_5}) станут доступны впоследствии. Далее выбирается дуга (v_{i_6}, v_{i_5}) среди дуг $C(v_{i_3})$ так, чтобы получить большее улучшение целевой функции.

Второй оператор мутации основан на окрестности 4-opt [140] (см. рис. 40). Здесь две дуги (v_{i_1}, v_{i_2}) и (v_{i_7}, v_{i_8}) выбираются случайным образом и удаляются, а две другие дуги для удаления выбираются так, чтобы получить наиболее существенное улучшение целевой функции.

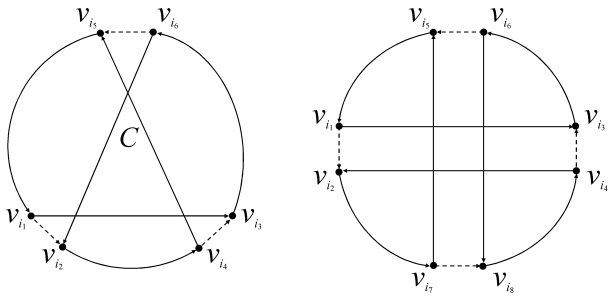


Рисунок 40 — Окрестности 3-opt и 4-opt.

Правило перезапуска. Баланс между локальным и эволюционным методами поиска настраивается адаптивно. Одним из простых способов адаптации является следующее классическое правило перезапуска: ЭА выполняет $N/2$ итераций без перезапуска. Далее ЭА перезапускается как только будет достигнута итерация, номер которой в два раза больше номера итерации, где получено последнее улучшение рекордного значения целевой функции.

Также используется правило перезапуска, основанное на методе переписи Шнабеля. Метод переписи Шнабеля был разработан в биометрии для статистической оценки размера популяции животных. Согласно данному методу, из популяции извлекаются выборки размера n_0 и подсчитывается количество *различных* животных, наблюдаемых в выборках. При этом предполагается, что вероятность отлова для всех животных одинакова. Отобранные животные помечаются, если это не было сделано ранее, и возвращаются обратно в популяцию. Далее статистическая оценка для общего числа ν особей в популяции вычисляется на основе общего числа отловленных и помеченных животных. В текущей работе метод переписи Шнабеля применяется для оценки количества значений, которые дискретная случайная величина может принимать с ненулевой вероятностью аналогично работе [205].

Пусть величина r задает длину исторического периода, рассматриваемого для статистического анализа в правиле перезапуска ЭА. Предполагается, что в течении последних r итераций все новые потомки в ЭА имеют одинаковое распределение вероятностей, и могут рассматриваться как отобранные животные в методе переписи Шнабеля. В свою очередь указанный метод используется для оценки числа ν различных решений, которые можно посетить с положительной вероятностью, в предположении, что текущее распределение потомков остается неизменным.

Также предполагается, что выборка, наблюдаемая за последние r итераций ЭА, состоит из r независимых решений-потомков. Определим случайную величину K как число различных решений в этой выборке. Делается еще одно упрощающее предположение о том, что все решения, которые могут быть сгенерированы с текущим распределением, имеют равные вероятности. Тогда для любого фиксированного ν, случайная величина K имеет следующее распределение:

$$\Pr\{K = k\} = \frac{\nu!}{(\nu - k)!} \frac{S(r,k)}{\nu^r},$$

где $S(r,k) = \frac{1}{k!} \sum_{s=0}^{k} (-1)^k \binom{k}{s} (k-s)^r$ – это число Стирлинга второго рода. Данное распределение также известно как распределение Арфведсона. Тогда оценка максимального правдоподобия $\hat{\nu}^{\mathrm{ML}}$ для неизвестной

величины ν равна

$$\hat{\nu}(r,k) = \operatorname{argmax}\left\{\frac{\nu!}{(\nu-k)!\nu^r}\right\}, \tag{219}$$

где k – это общее количество решений сгенерированных на последних r итерациях. Значение величины $\hat{\nu}^{\mathrm{ML}} = \hat{\nu}(r,k)$ может быть вычислено из (219) с помощью стандартных методов одномерной оптимизации. Технические подробности и дополнительные ссылки на литературу можно найти в [205].

Алгоритм перезапускается как только оценка $\hat{\nu}^{\mathrm{ML}}$ становится равной k. Обоснование такого правила заключается в том, что выполнение равенства $\hat{\nu}^{\mathrm{ML}} = k$ свидетельствует, скорее всего, об отсутствии непосещенных решений в той части допустимой области, где популяция алгоритма находилась последние r итерации. В такой ситуации целесообразно перезапустить алгоритм, а не ждать пока распределение популяции существенно изменится под влиянием эволюционных процессов.

Значение параметра r настраивается в процессе работы алгоритма. Всякий раз, когда значение рекорда уменьшается, величина r полагается равной размеру популяции N. Если же за последние $2r$ итераций значение рекорда не изменяется, то величина r увеличивается в два раза. Отметим, что улучшение рекорда говорит о достижении популяцией новой неисследованной области поиска, поэтому длину исторического периода следует уменьшить (в предлагаемом варианте до размера популяции N). С целью уменьшения времени работы алгоритма, критерий остановки проверяется только при обновлении значения r.

4.3.2 Вычислительный эксперимент

Структура раздела. В этом разделе представлены результаты экспериментального исследования ЭА с оптимальной рекомбинацией для задачи коммивояжера. Рассматриваются результаты предварительных экспериментов, направленных на оценку вычислительной стоимости кроссинговера ODEC и эффекта от локального поиска, применяемого на этапе инициализации. Описывается статистический тест, используемый для сравнения алгоритмов в дальнейшем. Представлено сравнение предло-

женного алгоритма со следующими алгоритмами:

- меметический алгоритм SAX/RAI [63],
- гибридный алгоритм [243],
- ГА с кроссинговером GAPX [228],
- ГА с кроссинговером EAX [190],
- алгоритмом с двусвязными выталкивающими цепями [206],
- алгоритмом локального поиска с чередующимися окрестностями [64],
- алгоритм ветвей и границ [231].

Также представлено экспериментальное исследование трех модификаций предложенного алгоритма:

- изменение стратегии управления популяцией на элитарную рекомбинацию,
- замена детерминированного кроссинговера ODEC на его рандомизированную версию DEC, и
- тестирование правила перезапуска на основе переписи Шнабеля.

Тестовые примеры, оборудование и язык программирования. Используются примеры АЗК из библиотеки TSPLIB [207]. Библиотека включает примеры из различных приложений и состоит из трех серий ftv, ft и rbg, а также отдельных задач br17, p43, ry48p, kro124p. Размеры примеров варьируются от 17 до 443. Предлагаемый меметический алгоритм ($\mathcal{MA}$) был запрограммирован на языке Java (NetBeans IDE 8.2). Вычислительный эксперимент проводился на компьютере с процессором Intel Core 2 Duo CPU E7200 2.53 GHz, 2 Gb RAM, если не указано иное. Во всех экспериментах установлен размер популяции $N = 100$, размер турнира $s = 10$ и вероятность мутации $p_{mut} = 0.1$.

В дальнейшем различные версии с конкретными операторами будут обозначаться как $\mathcal{MA}$ с подстрочным индексом, например $\mathcal{MA}_{SC}$ или $\mathcal{MA}_{ODEC}$.

Предварительный вычислительный эксперимент. Проведен предварительный эксперимент для оценки того, как вычислительная стоимость оптимальной рекомбинации зависит от текущего номера итерации t. $\mathcal{MA}$ был запущен на примерах серии ftv 1000 раз, и каждый запуск продолжался 2000 итераций. Здесь используется только оператор кроссинговера на каждой итерации алгоритма, а начальная популяция строится без использования методов локального поиска. Динамика времени, необходимого для решения ЗОР, показана на рис. 41. Графики для остальных задач серии ftv были аналогичны и здесь опущены. Видно из рис. 41, что время решения ЗОР уменьшается с увеличением номера итерации, что связано с уменьшением разнообразия популяции. Оптимизированный кроссинговер очень затратен по времени в случае серии rbg. Алгоритм выполнил менее 20 итераций за 15 минут на примерах этой серии. Скорее всего, причина больших затрат времени в кроссинговере на серии rbg – это высокий уровень разнообразия популяции, специфичный для этой серии из-за малого разброса значений длин дуг.

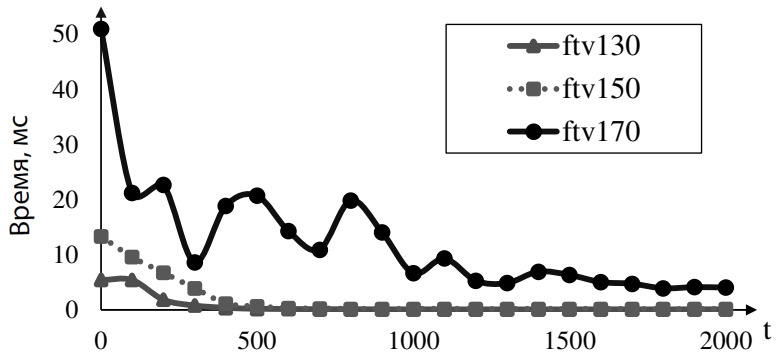


Рисунок 41 — Среднее время работы оператора кроссинговера на примерах ftv130, ftv150 и ftv170.

Чтобы избежать больших вычислительных затрат на рекомбинацию в начале запуска $\mathcal{MA}$, во всех последующих экспериментах установлено ограничение по времени равным $\frac{T}{1000}$ для каждого вызова кроссинговера ODEC, где T – общий бюджет времени, выделенный $\mathcal{MA}$. Когда оператор рекомбинации завершается из-за ограничения по времени, лучшее найденное решение для этой индивидуальной задачи ЗОР возвращается как результат рекомбинации.

Чтобы проанализировать важность локального поиска на начальном

и завершающем этапах $\mathcal{MA}$, был проведен следующий вычислительный эксперимент: для фиксированного бюджета времени, равного 2000 итераций, алгоритм с начальным локальным поиском ($\mathcal{MA}_1$) и без него ($\mathcal{MA}_0$) был запущен 30 раз на каждом примере. Лучшие значения целевой функции сохранялись для начальной популяции, в конце каждой итерации и после локального обновления последней популяции. Эти значения усреднялись по 30 запускам и представлены для примеров ftv150 и kro124p в виде box-plot на рис. 42 (прямоугольники показывают стандартное отклонение, усы – максимальные и минимальные значения, треугольники – результат улучшения локального поиска после популяционного поиска). Чтобы учесть время, затраченное на начальный локальный поиск, график $\mathcal{MA}_1$ сдвинут на итерациях $1, 2, \ldots$ вправо. Этот сдвиг равен среднему числу итераций $\mathcal{MA}_0$, которые были бы выполнены во время фазы начального локального поиска в $\mathcal{MA}_1$.

Как видно, начальный локальный поиск играет важную роль для конечного результата, но поскольку локальный поиск – это трудоемкая процедура, поэтому баланс между локальным поиском и популяционным поиском настраивается адаптивно. Также отметим, что если жадная эвристика при построении особей начальной популяции генерирует только один подцикл, это указывает на то, что пример АЗК был решен оптимально, и алгоритм останавливается на этапе инициализации.

В дальнейшем версия $\mathcal{MA}$, основанная на кроссинговере ODEC и классическом правиле перезапуска, обозначается как $\mathcal{MA}_{ODEC}$.

Инструменты статистического анализа. Статистический анализ экспериментальных данных проводился с использованием теста значимости из [59], гл. 8, §2. Предположим, что два алгоритма сравниваются по вероятности «успеха», где «успех» соответствует нахождению оптимального решения. Пусть P_1 и P_2 обозначают вероятности успеха для рассматриваемых алгоритмов. Нулевая гипотеза формулируется как $P_1 = P_2$.

Процедура тестирования следующая. При нулевой гипотезе оценка общего уровня успеха: $\hat{P} = \frac{\hat{P}_1 N_1 + \hat{P}_2 N_2}{N_1 + N_2}$, где $\hat{P}_1$ обозначает частоту успеха в N_1 запусках для первого алгоритма, а $\hat{P}_2$ - частоту успеха в N_2 запусках для второго алгоритма. Затем разница $\hat{P}_1 - \hat{P}_2$ выражается в единицах

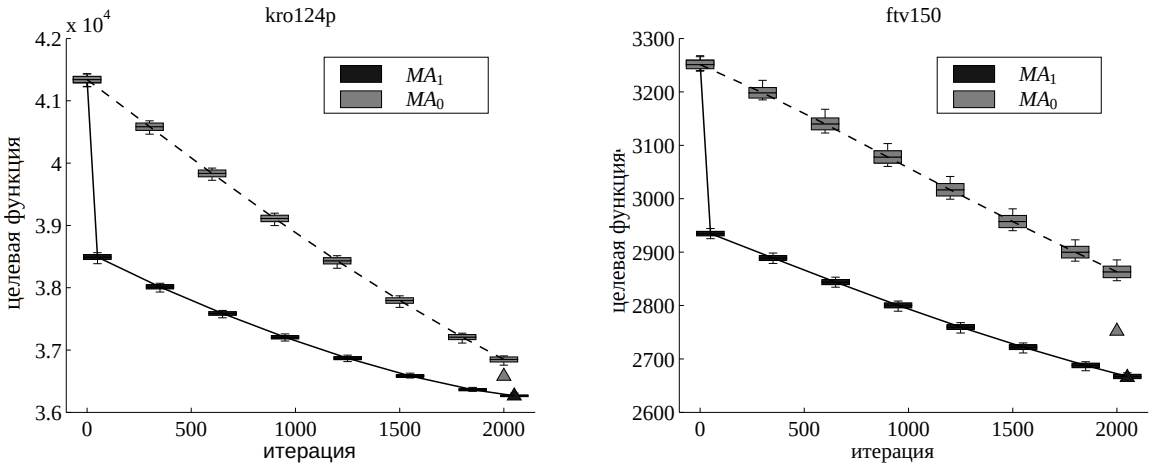


Рисунок 42 — Динамика целевой функции в ходе работы $\mathcal{MA}$. Прямоугольники показывают стандартное отклонение, усы – максимальные и минимальные значения, треугольники – среднее значение целевой функции, достигнутое после локального поиска.

стандартного отклонения путем вычисления статистики $A = \frac{|\hat{P}_1 - \hat{P}_2|}{\hat{SD}}$, где $\hat{SD} = \sqrt{\frac{\hat{P}(1-\hat{P})}{N_1} + \frac{\hat{P}(1-\hat{P})}{N_2}}$ – оценка стандартного отклонения.

Предполагается, что статистика A имеет нормальное распределение. Для проверки нулевой гипотезы против альтернативной при заданном уровне значимости α сравниваем A с квантилем стандартного нормального распределения $z_{\alpha/2}$. Если A больше $z_{\alpha/2}$, нулевая гипотеза отвергается. В противном случае нулевая гипотеза принимается. При $\alpha = 0.05$ имеем $z_{0.025} = 1.96$.

Сравнение с известными алгоритмами. В этом подразделе предложенный меметический алгоритм $\mathcal{MA}_{ODEC}$ сравнивается с эвристиками, которые ранее продемонстрировали свою конкурентоспособность для АЗК [63, 64, 190, 206, 228, 243].

Меметический алгоритм SAX/RAI ($MA_{SAX/RAI}$) из [63] является одним из наиболее известных алгоритмов в литературе по эвристикам для АЗК. Популяция $MA_{SAX/RAI}$ состоит из 13 агентов, организованных в виде троичного дерева из трех уровней. Каждый агент обрабатывает два допустимых решения. На этапе инициализации одно решение строится с использованием жадной эвристики, как и в алгоритме $\mathcal{MA}_{ODEC}$, а все остальные решения генерируются с использованием жадной эври-

стики на основе стратегии «случайной вставки». $MA_{SAX/RAI}$ применяет Strategic Arc Crossover (SAX), пытаясь построить потомка, используя только дуги от родителей, но без гарантии, что потомок унаследует все общие дуги. Процедура локального поиска под названием Recursive Arc Insertion (RAI) улучшает решения после рекомбинации.

Эксперимент организуется так, чтобы $MA_{SAX/RAI}$ и $\mathcal{MA}_{ODEC}$ получают примерно равные CPU ресурсы на вычисления. Для этого $\mathcal{MA}_{ODEC}$ получает ограничение по времени (обозначенное как T) в 3 раза меньше, чем время CPU, использованное $MA_{SAX/RAI}$ на ноутбуке с Pentium 1.7 GHz под управлением GNU/linux в [63]. Этот масштабирующий коэффициент выбран на основе сравнения компьютеров с помощью таблицы производительности [86].

Для статистического сравнения на каждом примере $\mathcal{MA}_{ODEC}$ запускался 1000 раз. В каждом запуске $\mathcal{MA}_{ODEC}$ получал то же ограничение по времени, что указанно выше. В [63] $MA_{SAX/RAI}$ запускался 20 раз на каждом примере. Таблица 7 показывает полученные результаты, где F_{opt} представляет частоту нахождения оптимального решения, Δ_{fin} – среднее относительное отклонение для полученного решения (%), Δ_{init} – среднее относительное отклонение для начального решения (%). В таблицах 7 и 8 значения F_{opt} и Δ_{fin}, выделенные жирным, соответствуют лучшим значениям в строках. Как видно из таблицы 7, $\mathcal{MA}_{ODEC}$ достиг 100% успеха на 20 из 27 примеров. На каждом примере $\mathcal{MA}_{ODEC}$ находил оптимум не менее чем в 99% запусков.

В таблице 7 символ «*» указывает на статистическую значимость различий между значениями F_{opt} $\mathcal{MA}_{ODEC}$ и $MA_{SAX/RAI}$ при уровне значимости $\alpha \leq 0.05$. На 14 из 27 примеров $\mathcal{MA}_{ODEC}$ находит оптимум чаще, чем $MA_{SAX/RAI}$ (во всех 14 случаях разница между частотами F_{opt} статистически значима). Оба алгоритма демонстрируют 100% частоту получения оптимума на 12 задачах. Среднее относительное отклонение $\Delta_{\text{init}} = 0$ означает, что оптимальное решение уже найдено в начальной популяции. Заметим, что эвристика В. Занга весьма эффективна на серии rbg, и оптимальные решения для всех rbg-примеров были найдены в $\mathcal{MA}_{ODEC}$ и $MA_{SAX/RAI}$ на этапе инициализации. Частоты успеха на серии rbg для предложенного алгоритма без эвристики В. Занга на этапе

инициализации были следующими: rbg323 – 0.91, rbg358 – 0.93, rbg403 и rbg443 – 1.

$MA_{SAX/RAI}$ немного превосходит $\mathcal{MA}_{ODEC}$ на примере ft53, но разница не статистически значима. Среднее относительное отклонение полученного решения для $\mathcal{MA}_{ODEC}$ примерно в 60 раз меньше, чем среднее относительное отклонение для $MA_{SAX/RAI}$.

Гибридный подход (HA) на основе кодировки значений в позициях перестановки был предложен для АЗК в [243]. Здесь начальная популяция строится случайным образом. Кроссинговер выполняется много раз для каждой пары родителей, и два лучших решения берутся как потомки. HA использует механизмы миграции решений, локального и глобального улучшения, которые работают в соответствии с представлением решений на основе кодировки значений в позициях. Алгоритм из [243] был протестирован на 16 примерах АЗК из библиотеки TSPLIB и показал нулевое относительное отклонение на всех этих задачах, кроме ry48p, kro124p и ftv170, где Δ_{fin} была равна 0.001, 0.001 и 0.02 соответственно. Время работы HA составило 56.5 сек. в среднем за 100 запусков на Pentium 1.6 GHz. $\mathcal{MA}_{ODEC}$ был протестирован в аналогичном эксперименте, где реализовано 100 запусков для каждого примера с ограничением по времени в 6.6 раз меньше, чем CPU ресурс, использованный HA [243] (коэффициент 6.6 выбран с использованием таблицы производительности [86]). $\mathcal{MA}_{ODEC}$ превзошел HA в этом эксперименте со средним относительным отклонением $\Delta_{\text{fin}} = 0$ на всех 16 примерах.

ГА с кроссинговером GAPX [228], который основан на построении жадного решения ЗОР. ГА из [228] использует стратегию управления популяцией, известную как популяционная схема, и элитизм используется для сохранения текущего лучшего решения. Локальный поиск на основе окрестности 3-opt применяется на этапе инициализации и когда кроссинговер не улучшил лучшее решение. ГА [228] показывает конкурентоспособные результаты с точки зрения частоты нахождения оптимума при достаточно большом времени счета. На всех 19 задачах библиотеки TSPLIB, рассмотренных в [228], ГА с GAPX (GA_{GAPX}) показал 100%

успеха, а среднее время счета составило 93.52 сек. на компьютере с процессором Core 2 Quad 2.83 GHz. $\mathcal{MA}_{ODEC}$ протестирован с ограничением на время счета в 1.12 раз больше, чем у GA_{GAPX} [228] (коэффициент 1.12 выбран на основе соотношения тактовых частот процессоров Core 2). $\mathcal{MA}_{ODEC}$ также показал 100% частоту получения оптимума на всех 19 задачах.

ГА с кроссинговером EAX (GA_{EAX}) был разработан для АЗК в [190]. Этот оператор кроссинговера использует концепцию AB-циклов и генерирует потомков, которые, в отличие от ODEC, могут содержать некоторые дуги, отсутствующие у родителей. ГА [190] работает согласно популяционной схеме воспроизведения. Начальная популяция в [190] генерируется с помощью локального поиска на основе окрестности 3-opt, как и в предложенном меметическом алгоритме. ГА [190] был реализован на языке C++ и выполнялся на Intel Xeon 2.93 GHz. Для двух размеров популяции 100 и 300 ГА [190] запускался 100 раз на каждом примере из TSPLIB.

Для сравнения был проведен эксперимент, в котором меметический алгоритм тестировался на Intel Xeon E5420 2.5 GHz, 16 Gb RAM. $\mathcal{MA}_{ODEC}$ с размером популяции $N = 100$ был запущен 100 раз на каждом примере. В каждом запуске $\mathcal{MA}_{ODEC}$ ограничение по времени было в 1.465 раз больше, чем время счета GA_{EAX} с размером популяции 100 в [190]. Масштабирующий коэффициент выбран на основе соотношения тактовых частот процессоров и того факта, что компилятор Java оценивается как минимум на 25% медленнее, чем компилятор C++ (см., например, [63]).

Для $\mathcal{MA}_{ODEC}$ и GA_{EAX} с размером популяции 100 частота нахождения оптимума представлена в таблице 8 и на рис. 43 (таблица и рисунок также содержат результаты $\mathcal{MA}_{SC}$). На рис. 43 доверительные интервалы для вероятности успеха построены с использованием метода из [187], Глава 8.

В таблице 8 символ «$*$» обозначает статистическую значимость различий между значениями F_{opt} для $\mathcal{MA}_{ODEC}$ и GA_{EAX} при уровне значимости $\alpha \leq 0.05$. Таблица также содержит среднее относительное отклонение для полученных решений.

Задача	Версии $\mathcal{MA}$							Меметический аглоритм [63]			
	$\mathcal{MA}_{ODEC}$			$\mathcal{MA}_{ER}$			T, сек.	$MA_{SAX/RAI}$			
	Δ_{init}	F_{opt}	Δ_{fin}	Δ_{init}	F_{opt}	Δ_{fin}		Δ_{init}	F_{opt}	Δ_{fin}	время, сек.
br17	0.00	**1**	**0.00**	0.00	**1**	**0.00**	0.047	0	**1**	**0.00**	0.14
ftv33	0.00	**1**	**0.00**	0.00	**1**	**0.00**	0.097	12.83	**1**	**0.00**	0.29
ftv35	0.0003	**1**	**0.00**	0.12	0.975	0.004	0.11	0.14	**1**	**0.00**	0.33
ftv38	0.0001	**1***	**0.00**	0.056	0.91	0.017	0.103	0.13	0.25	0.10	0.31
p43	0.00	**1***	**0.00**	0.008	0.982	0.0005	0.16	0.05	0.55	0.01	0.48
ftv44	0.235	**1***	**0.00**	0.438	0.845	0.031	0.137	7.01	0.35	0.44	0.41
ftv47	0.099	**1**	**0.00**	0.192	0.991	0.0014	0.157	2.70	**1**	**0.00**	0.47
ry48p	0.612	**0.997***	**0.0016**	0.519	0.51	0.033	0.187	5.42	0.85	0.03	0.56
ft53	0.39	0.998	0.0008	0.413	0.593	0.086	0.187	18.20	**1**	**0.00**	0.56
ftv55	0.004	**1**	**0.00**	0.0013	**1**	**0.00**	0.167	3.61	**1**	**0.00**	0.50
ftv64	0.028	**1**	**0.00**	0.367	0.764	0.013	0.22	3.81	**1**	**0.00**	0.66
ft70	0.524	**0.997***	**0.0005**	0.239	0.568	0.013	0.32	1.88	0.4	0.03	0.96
ftv70	0.993	**1***	**0.00**	0.924	0.885	0.012	0.277	3.33	0.95	0.01	0.83
ftv90	0.062	**1**	**0.00**	0.351	0.516	0.009	0.317	3.67	**1**	**0.00**	0.95
ftv100	0.425	**1**	**0.00**	0.588	0.787	0.028	0.4	3.24	**1**	**0.00**	1.2
kro124p	0.191	**0.991***	**0.0005**	0.3	0.261	0.039	0.457	6.46	0.90	0.01	1.37
ftv110	0.285	**1***	**0.00**	0.344	0.822	0.027	0.57	4.7	0.90	0.02	1.71
ftv120	0.182	**0.999***	**0.0011**	1.232	0.325	0.705	0.73	8.31	0.35	0.14	2.19
ftv130	0.243	**0.996***	**0.0006**	1.57	0.335	0.081	0.727	3.12	0.90	0.01	2.18
ftv140	0.132	**0.994***	**0.0006**	0.257	0.463	0.077	0.887	2.23	0.70	0.08	2.66
ftv150	0.819	**1***	**0.00**	1.463	0.524	0.076	0.897	2.3	0.90	0.01	2.69
ftv160	0.026	**1***	**0.00**	0.074	0.488	0.0099	1.093	1.71	0.80	0.02	3.28
ftv170	0.104	**1***	**0.00**	0.265	0.225	0.162	1.307	1.38	0.75	0.05	3.92
rbg323	0.00	**1**	**0.00**	0.00	**1**	**0.00**	0.03	0.00	**1**	**0.00**	0.07
rbg358	0.00	**1**	**0.00**	0.00	**1**	**0.00**	0.03	0.00	**1**	**0.00**	0.08
rbg403	0.00	**1**	**0.00**	0.00	**1**	**0.00**	0.032	0.00	**1**	**0.00**	0.08
rbg443	0.00	**1**	**0.00**	0.00	**1**	**0.00**	0.033	0.00	**1**	**0.00**	0.09
Average	0.198	0.999	0.0002	0.36	0.732	0.053	0.371	3.564	0.835	0.036	1.073

Таблица 7 — Сравнение $\mathcal{MA}_{ODEC}$ и $\mathcal{MA}_{ER}$ с $MA_{SAX/RAI}$.

Задача	Версии $\mathcal{MA}$					GA_{EAX}		
	$\mathcal{MA}_{ODEC}$		$\mathcal{MA}_{SC}$					
	F_{opt}	Δ_{fin}	F_{opt}	Δ_{fin}	Xeon 2.5 ГГц время, сек.	F_{opt}	Δ_{fin}	Xeon 2.93 ГГц время, сек.
br17	**1**	**0**	**1**	**0**	0.015	**1**	**0**	0.01
ftv33	**1**	**0**	**1**	**0**	0.015	**1**	**0**	0.01
ftv35	0.99	0.001	**1**	**0**	0.029	**1**	**0**	0.02
ftv38	**1**	**0**	**1**	**0**	0.029	**1**	**0**	0.02
p43	**1***	**0**	**1****	**0**	0.029	0.94	0.001	0.02
ftv44	0.99*	0.006	**1****	**0**	0.029	0.66	0.211	0.02
ftv47	**1**	**0**	0.99	0.0034	0.029	**1**	**0**	0.02
ry48p	0.98	0.009	0.97	0.0069	0.029	**1**	**0**	0.02
ft53	0.96	0.002	**1****	**0**	0.0439	0.95	0.006	0.03
ftv55	**1**	**0**	**1**	**0**	0.0439	**1**	**0**	0.03
ftv64	**1**	**0**	**1**	**0**	0.059	**1**	**0**	0.04
ft70	0.85	0.001	**0.88**	**0.0009**	0.059	0.85	0.001	0.04
ftv70	0.99	0.005	**1**	**0**	0.059	**1**	**0**	0.04
ftv90	0.62	0.056	0.89	0.0087	0.073	**0.96***	**0.006**	0.05
kro124p	**1**	**0**	0.91	0.0027	0.117	0.97	0.001	0.08
ftv100	0.86	0.035	**0.98**	**0.0022**	0.088	0.97*	0.003	0.06
ftv110	0.78	0.01	**0.93**	0.0092	0.117	**0.93***	**0.007**	0.08
ftv120	0.81	0.023	**0.90**	**0.0046**	0.147	0.87	0.016	0.1
ftv130	0.88	**0.001**	**0.92**	0.0055	0.161	0.89	0.01	0.11
ftv140	**0.95**	0.012	0.94	**0.0074**	0.176	0.89	0.012	0.12
ftv150	0.8	0.044	**0.97**	**0.0046**	0.219	0.91*	0.007	0.15
ftv160	**1**	**0**	**1**	**0**	0.249	0.99	0.001	0.17
ftv170	0.94	0.005	**0.99**	**0.0011**	0.278	0.98	0.002	0.19
rbg323	**1**	**0**	**1**	**0**	1.406	**1**	**0**	0.96
rbg358	**1**	**0**	**1**	**0**	1.934	**1**	**0**	1.32
rbg403	**1**	**0**	**1**	**0**	2.359	**1**	**0**	1.61
rbg443	**1**	**0**	**1**	**0**	2.491	**1**	**0**	1.7
Среднее	0.941	0.008	0.972	0.002	0.38	0.953	0.0105	0.26

Таблица 8 — Сравнение характеристик решений двух версий $\mathcal{MA}$ с GA_{EAX}.

В 10 из 27 случаев GA_{EAX} находит оптимум чаще, чем $\mathcal{MA}_{ODEC}$, причем в 4 из этих случаев разница между частотами статистически значима. $\mathcal{MA}_{ODEC}$ превосходит GA_{EAX} на 6 примерах, причем в 2 случаях разница статистически значима. Оба алгоритма демонстрируют одинаковую частоту на 11 задачах. Среднее относительное отклонение для $\mathcal{MA}_{ODEC}$ немного меньше, чем для GA_{EAX} (примерно в 1.3 раза).

Алгоритм с двусвязными выталкивающими цепями (DR) был предложен Рего и др. [206] для асимметричной задачи коммивояжера. Алгоритм [206] запускался на процессоре Intel Core Duo 2.66 ГГц с 3 ГБ ОЗУ. Сравниваются результаты $\mathcal{MA}_{ODEC}$ из таблицы 7 с экспериментальными результатами DR из [206]. На основе соотношения тактовых частот процессоров оценена производительность используемого компьютера как примерно 0.95 от производительности компьютера, использованного в [206]. Среднее относительное отклонение решения для $\mathcal{MA}_{ODEC}$ составило 0.0002, тогда как для DR — 0.369 (на каждом примере $\mathcal{MA}_{ODEC}$ демонстрирует меньшее отклонение Δ_{fin}, чем DR). Средние вычислительные ресурсы меметического алгоритма меньше, чем у DR (примерно в 7 раз). Однако DR показывает немного меньшее время, чем лимит ресурсов $\mathcal{MA}_{ODEC}$ на небольших примерах ftv33, ftv35, ftv38, ftv44, ftv47, ftv55, ftv64, ftv90 и p43.

Локальный поиск с чередующимися окрестностями (VNS) – один из известных подходов к решению оптимизационных задач. VNS – это метаэвристика, которая систематически изменяет окрестность в рамках рандомизированного локального поиска. В работе [64] представлены VNS для АЗК. В стратегии VNS с управляемыми процедурами встряски ($GVNS$) [64] сочетаются известные подходы локального поиска HyperOpt и 3-opt.

$GVNS$ выполнялся на процессоре Pentium II 400 МГц с 128 МБ памяти. Сравниваются результаты $\mathcal{MA}_{ODEC}$ из таблицы 7 с экспериментальными результатами $GVNS$ из [64]. Оценена производительность компьютера, использованного в [64], как примерно 4% от производительности нашего компьютера (этот масштабирующий коэффициент выбран на

основе таблицы производительности [86]). Среднее относительное отклонение решения $\Delta_{\text{fin}} = 0.0002$ для $\mathcal{MA}_{ODEC}$ меньше, чем среднее относительное отклонение $\Delta_{\text{fin}} = 0.506$ для $GVNS$. Средние вычислительные ресурсы меметического алгоритма примерно в 1.75 раза меньше, чем у $GVNS$.

Обобщая полученные экспериментальные результаты, заключаем, что $\mathcal{MA}_{ODEC}$ демонстрирует частоты успеха и относительное отклонение, равное или лучшее, чем у $MA_{SAX/RAI}$ [63], GA_{GAPX} [228], HA [243], DR [206] и $GVNS$ [64]. Более того, $MA_{SAX/RAI}$ превосходит $\mathcal{MA}_{ODEC}$ только на одном примере, но разница между частотами нахождения оптимума не является статистически значимой. GA_{EAX} [190] показывает лучшие частоты нахождения оптимума, чем $\mathcal{MA}_{ODEC}$, но последний достигает меньшего среднего относительного отклонения.

Алгоритм ветвей и границ. Алгоритм $BnB(SCS)$ [231] является одним из лучших среди точных алгоритмов для АЗК. Также проведено сравнение результатов $\mathcal{MA}_{ODEC}$ из предыдущего эксперимента с $MA_{SAX/RAI}$ (таблица 7) с экспериментальными результатами $BnB(SCS)$ из [231]. Грубое сравнение на основе таблицы производительности компьютеров [86] приводит к заключению, что используемый компьютер примерно в 2.2 раза производительнее компьютера из исследования [231]. $\mathcal{MA}_{ODEC}$ демонстрирует как минимум 99% успеха в нахождении оптимума при вычислительных ресурсах в среднем примерно в 155 раз меньших, чем у $BnB(SCS)$, за исключением 6 относительно простых примеров: ftv33, ftv44, rbg323, rbg358, rbg403 и rbg443. В эксперименте [231] для p43 и kro124p решение не было найдено $BnB(SCS)$ за 3600 сек.

Модификации предложенного меметического алгоритма. В этом подразделе рассматривается три модификации $\mathcal{MA}_{ODEC}$, где меняется стратегия управления популяцией, оператор скрещивания и правило перезапуска. Алгоритм содержит много компонентов и параметров, и существуют иные способы их настройки, но рассматриваемые три варианта интересны тем, что они успешно использовались в аналогичных алгоритмах.

Элитная рекомбинация. Рассмотрим экспериментальное сравнение стратегии управления популяцией $\mathcal{MA}$ с *элитарной рекомбинацией* [118], которая ранее использовалась в генетических алгоритмах с оптимальной рекомбинацией [244]. Далее $\mathcal{MA}_{ER}$ обозначает меметический алгоритм, в котором стационарная схема воспроизведения заменена на элитарную рекомбинацию. Эта модификация заключается в равномерном выборе родителей $\mathbf{p}_1, \mathbf{p}_2$ на шаге 2.1 и замене потомком худшего из двух родителей $\mathbf{p}_1, \mathbf{p}_2$ на шаге 2.4.

Алгоритмам $\mathcal{MA}_{ODEC}$ и $\mathcal{MA}_{ER}$ было установлено одинаковое ограничение по времени T, что и в эксперименте с $MA_{SAX/RAI}$. Результаты также приведены в таблице 7. Оба алгоритма демонстрируют 100% успех на 7 из 27 задач. На оставшихся 20 задачах средняя частота нахождения оптимальных решений для $\mathcal{MA}_{ER}$ оценивается примерно как 64% от средней частоты для $\mathcal{MA}_{ODEC}$. На каждом из примеров разница между частотами статистически значима.

Отметим, что элитарная рекомбинация направлена на поддержание разнообразия популяции. По этой причине меметический алгоритм с элитарной рекомбинацией показывается лучшие результаты, чем стационарная схема воспроизведения в серии экспериментов без перезапусков. Перезапуски, выполняемые в $\mathcal{MA}_{ODEC}$, позволяют избежать локализации поиска и восстановить разнообразие популяции, что приводит к лучшим результатам.

Рандомизированный оператор скрещивания. Был проведен дополнительный эксперимент для сравнения оптимизированного оператора скрещивания ODEC с его рандомизированным аналогом DEC. В этом эксперименте сравнивались $\mathcal{MA}_{ODEC}$ с $\mathcal{MA}_{DEC}$, где оператор DEC заменяет ODEC, при том же ограничении на время вычислений T, что и в эксперименте с $MA_{SAX/RAI}$.

На всей серии примеров $\mathcal{MA}_{DEC}$ в среднем давал только 65.2% частоту нахождения оптимума, тогда как для $\mathcal{MA}_{ODEC}$ такая частота составила 99.9%. Алгоритмы дали идентичные результаты на 11 примерах. Однако для задач большой размерности, таких как ftv120, ftv130, ftv140, ftv150 и ftv170, $\mathcal{MA}_{DEC}$ находил оптимальное решение не более чем в 10

случаях из 1000 запусков. Этот эксперимент показывает преимущество ODEC перед DEC в $\mathcal{MA}$.

Самоадаптивное правило перезапуска на основе метода переписи Шнабеля. Каждый перезапуск $\mathcal{MA}$ вызывает интенсивное выполнение процедур локального поиска со случайными начальными точками. Поэтому продолжительность запуска (определяемая правилом перезапуска) – это важный параметр, влияющий на баланс между локальным поиском и эволюционной процедурой поиска. Последняя включает оператор оптимизированного скрещивания, который также может рассматриваться как оператор локального поиска, работающий с двумя входными решениями, а не с одним.

Помимо классического правила перезапуска, также протестировано правило перезапуска из [102], основанное на методе Шнабеля в сочетании с принципом максимального правдоподобия. Меметический алгоритм с правилом перезапуска из [102], обозначенный как $\mathcal{MA}_{SC}$, имеет все остальные процедуры и параметры идентичными $\mathcal{MA}_{ODEC}$.

Сравнение $\mathcal{MA}_{SC}$ с $\mathcal{MA}_{ODEC}$ в тех же условиях, что и в эксперименте с $MA_{SAX/RAI}$ (см. подраздел 4.3.2), не показало статистически значимых различий в частоте нахождения оптимальных решений: для обоих алгоритмов эти частоты были очень близки к 100%.

В следующем эксперименте установлено то же ограничение на время вычислений для $\mathcal{MA}_{SC}$, что и в эксперименте с GA_{EAX}, и проведено тестирование на Intel Xeon E5420 2.5 ГГц, как и $\mathcal{MA}_{ODEC}$. Результаты представлены в таблице 8 и на рис. 43. $\mathcal{MA}_{SC}$ демонстрирует лучшие результаты, чем $\mathcal{MA}_{ODEC}$, на 12 примерах, причем разница между частотами F_{opt} статистически значима в 5 случаях. $\mathcal{MA}_{ODEC}$ немного превосходит $\mathcal{MA}_{SC}$ на четырех тестовых задачах, но разница статистически значима только для примера kro124p.

В 11 из 27 случаев $\mathcal{MA}_{SC}$ находит оптимум чаще, чем GA_{EAX} (в 3 из этих случаев (отмечены «**») разница между частотами статистически значима при уровне значимости $\alpha \leq 0.05$). GA_{EAX} превосходит $\mathcal{MA}_{SC}$ на четырех примерах, но разница между частотами нахождения оптимума не является статистически значимой. Средняя частота F_{opt} на-

хождения оптимума для $\mathcal{MA}_{SC}$ составляет 0.972, тогда как для GA_{EAX} — 0.953. Средняя относительная ошибка решения для $\mathcal{MA}_{SC}$ примерно в 5 раз меньше, чем для GA_{EAX}.

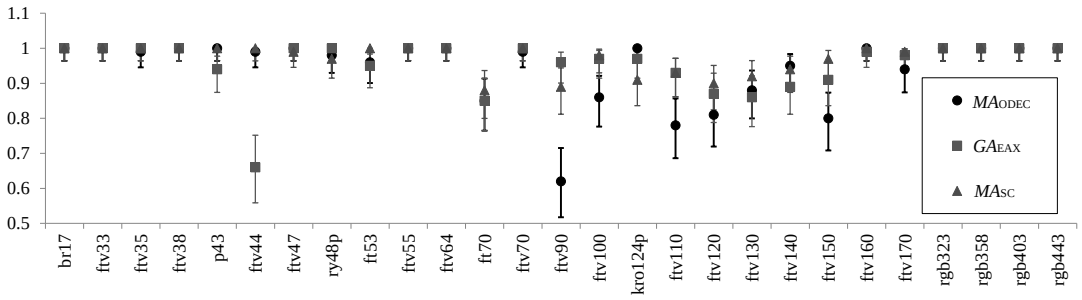


Рисунок 43 — Частота нахождения оптимума и 95%-доверительный интервал.

4.4 Задача составления расписаний выполнения заказов клиентов

Рассматривается задача обслуживания m клиентов на одном производственном устройстве (станке) [128, 158]. Заказы клиентов включают подмножества из n различных продуктов. Обозначим через M множество клиентов, а через J – множество продуктов. Длительность производства продукта j в заказе клиента i обозначается через $p_{ij} \geq 0$, $i \in M,\ j \in J$.

При переключении производства с продукта j на продукт j' требуется переналадка длительностью $s_{jj'} \geq 0,\ j, j' \in J,\ j \neq j'$. Если продукт j является первым на устройстве, то требуется начальная переналадка $s'_j,\ j \in J$. Клиенты также характеризуются директивными сроками d_i и весами q_i для их заказов, $i \in M$. Веса означают степень важности клиентов.

Определим операцию как пару клиент-продукт (i, j), $i \in M$, $j \in J$. Операции должны выполняться без прерываний. Множество допустимых решений Π состоит из всех возможных перестановок операций.

Рассматривается два критерия. Первый – минимизация суммы моментов завершения заказов, второй – максимизация суммы весов заказов, выполненных в срок, то есть до директивных сроков. Пусть $C_i(\pi)$ – время завершения производства последнего продукта клиента $i \in M$ в

перестановке π, а $I(\pi)$ - подмножество клиентов, для которых $C_i(\pi) \leq d_i$ в π. Тогда первый критерий $f_1(\pi) = \sum_{i=1}^{m} C_i(\pi)$, а второй критерий $f_2(\pi) = \sum_{i \in I(\pi)} q_i$.

Время завершения каждого заказа и последовательность операций влияют на значение обоих критериев. Первый критерий имеет интерпретацию на практике как общая стоимость ресурса, занятого заказами в процессе обслуживания. Во втором критерии учитываются директивные сроки работ и максимизируется общая «дополнительная прибыль» от своевременно выполненных заказов. Такие задачи возникают, например, в химическом производстве, фармацевтике и сервисных системах [66, 218, 242].

Задача с первым критерием NP-трудна даже в случае, когда длительности переналадок не зависят от последовательности, как доказано в [94]. Нижняя оценка для первого критерия может быть вычислена как

$$\sum_{i=1}^{m} \left(\sum_{j=1}^{n} p_{\sigma_i,j} \right) (m - i + 1) + m \sum_{j=1}^{n} \min\{\min_{j' \neq j} s_{j',j};\ s'_j\}, \tag{220}$$

где σ соответствует неубывающей последовательности суммарных длительностей заказов клиентов. Отметим, что эта оценка будет «хорошей» для примеров с близкими значениями длительностей переналадок на продукты. Случай со вторым критерием также NP-труден даже при нулевых длительностях переналадок (см., например, [170]).

Базовые версии метаэвристик (алгоритмы муравьиной колонии, методы поиска с запретами, имитация отжига, генетические алгоритмы) были экспериментально протестированы для критерия суммарного времени завершения в [128]. Модель целочисленного линейного программирования и анализ сложности приведены в [94]. Задача составления расписаний выполнения заказов клиентов в постановке, основанной на обработке продуктов (одинаковые продукты из разных заказов формируют партию и обрабатываются последовательно без других продуктов), исследовалась в [66]. Были предложены и протестированы ЦЛП-модели и алгоритм поиска с запретами. Нечеткая постановка задачи и методы ее решения приведены в [158]. Эвристики для постановки с зависящими от последовательности длительностями переналадки предложены в [242].

Версии задачи со специализированными машинами и различными критериями исследовались в [54,106,177,199,218,219,238]. Был проведен анализ вычислительной сложности, а также предложены конструктивные приближенные алгоритмы, метаэвристики и матэвристики.

4.4.1 Операторы

Решения кодируются перестановками операций $\pi = (\pi_1, \ldots, \pi_{nm})$, где π_i - i-я операция в последовательности. Начальная популяция строится двумя алгоритмами: случайным образом и методом «присоединения к подпоследовательности» [244]. Используется s-турнирная селекция, которая случайным образом выбирает s особей из текущей популяции и отбирает лучшую среди них.

Используются классические операторы кроссинговера на перестановках: порядковый кроссинговер (OX) и одноточечный кроссинговер (1PX). (1,1) эволюционная стратегия на основе мутаций сдвига и обмена (операторы используются для мутации с равной вероятностью) интегрирован на шаге мутации (см., например, [44]). Количество шагов для (1,1) эволюционной стратегии выбирается случайно от 1 до $\alpha\% \times nm$, где α - настраиваемый параметр.

Также применяются оптимизированные операторы кроссинговера, где решается задача оптимальной рекомбинации. Отметим, что мутация применяется к выбранным родителям, а затем родители скрещиваются (попытка придать решениям новые свойства, а затем использовать кроссинговер для рекомбинации этих свойств). Предыдущие исследования из [84] подтверждают, что такая техника может успешно использоваться в случае оптимизированных операторов кроссинговера.

Теорема 72 *Задача оптимальной рекомбинации (I)–(II) NP-трудна для задачи составления расписаний выполнения заказов клиентов с критерием суммы весов работ, выполненных в срок.*

Доказательство. Доказательство основано на полиномиальном сведении задачи *Упорядоченное разбиение* [111]: дано упорядоченное множество $A = \{a_1, a_2, \ldots, a_{2m_0}\}$ и вес e_i каждого элемента $a_i \in A$, где $\sum_{a_i \in A} e_i = 2E$ и $e_i < e_{i+1},\ i = 1, \ldots, 2m_0 - 1$. Вопрос заключается в

том, можно ли разбить множество A на два подмножества A_1 и A_2 так, что $\sum_{a_i \in A_1} e_i = \sum_{a_i \in A_2} e_i = E$, $|A_1| = |A_2| = m_0$, и подмножество A_1 содержит только один элемент из каждой пары $a_{2i-1},\ a_{2i},\ i = 1, \ldots, m_0$.

Построим серию индивидуальных ЗОР. Пусть $n = m = 2m_0$ и каждый клиент требует индивидуальный продукт. Длительность производства $p_i = e_i$ и вес $q_i = e_i$ для клиента $i = 1, \ldots, m$. Все длительности переналадок считаются пренебрежимо малыми. Заказы имеют общий директивный срок $d_i = E$. Зададим родительские перестановки:

$$\pi^1 = (1, \ldots, 2m_0 - 1, 2, \ldots, 2m_0);$$

$$\pi^2 = (2, \ldots, 2m_0, 1, \ldots, 2m_0 - 1).$$

Перестановка π, удовлетворяющая $\sum_{i \in I(\pi)} q_i \geq E$ и условию (I), существует тогда и только тогда, когда задача Упорядоченное разбиение имеет положительный ответ. Действительно, общий директивный срок E для всех заказов означает, что $\sum_{i \in I(\pi)} p_i \leq E$, следовательно $\sum_{i \in I(\pi)} e_i = E$ выполняется в случае, когда имеют место подмножества A_1 и A_2 такие, что $\sum_{a_i \in A_1} e_i = \sum_{a_i \in A_2} e_i = E$, и каждое подмножество содержит один элемент из каждой пары $a_{2i-1},\ a_{2i},\ i = 1, \ldots, m_0$. Таким образом, доказана NP-полнота распознавательной версии ЗОР, следовательно, исходная оптимизационная версия ЗОР NP-трудна. □

Статус сложности ЗОР для критерия суммы моментов завершения остается открытым вопросом.

Задача оптимальной рекомбинации решается в операторе кроссинговера двумя способами: перебором допустимых решений ЗОР или решением подзадачи целочисленного программирования из [254] с фиксированным подмножеством переменных, соответствующих условию (I). В первом случае применяется алгоритм, основанный на перечислении совершенных паросочетаний в специальном двудольном графе (см. раздел 3.1.2). Во втором случае подзадача решается с помощью ЦЛП-решателя или специализированных методов ветвей и границ. Также тестируются усеченные версии операторов, когда максимальное число просматриваемых решений ограничено и/или время счета решателя ограничено.

4.4.2 Вычислительный эксперимент

Все вычисления проводились на сервере с 128×AMD EPYC 7502 32-Core CPU, 500 GB RAM. В качестве языка программирования использовалась Java версии 11.0.6. В экспериментах каждый алгоритм запускался на каждом примере один раз.

Сначала меметический алгоритм запускался на серии построенных случайным образом примеров, где длительности переналадок и длительности операций (как основные конкурирующие элементы в целевой функции) выбираются случайно из одного интервала. Случайным образом генерируется 30 примеров с 50 клиентами и 25 продуктами. Числовые параметры генерируются случайно с равномерным распределением: интервал $(1, 40)$ используется для длительностей операций и переналадок, интервал $(20000, 40000)$ задан для директивных сроков, а интервал $(1, 5)$ – для весов.

MA тестировался с различными операторами кроссинговера на построенной серии примеров. Установлен размер популяции N равным 100, параметр мутации $\alpha = 30\%$ и размер турнира $s = 5$ на основе предварительных экспериментов и предыдущих исследований генетических алгоритмов такого типа (см., например, [55, 103, 248]). Значения выбирались из следующих множеств: $N \in \{100, 200, 300\}$, $s \in \{2, 5, 7, 10, 12\}$, $\alpha \in \{10, 15, 20, 25, 30, 35, 40\}$. Критерий остановки установлен в 10 минут для времени работы согласно ориентирам из практики (см., например, [55, 66, 218]).

Используются следующие обозначения:
MA_{GOX} – это MA с оптимизированным оператором кроссинговера на основе переборного алгоритма решения ЗОР, где задана верхняя граница на число перебираемых решений через верхнюю границу на число циклов, а все значения позиций из невыбранных циклов назначаются в соответствии с родительскими решениями;
MA_{ROX} – это MA с оптимизированным оператором кроссинговера на основе переборного алгоритма решения ЗОР, где задана верхняя граница на число циклов, а значения позиций из каждого невыбранного цикла назначаются случайным образом из значений одного из родителей;
MA_{CX} – это MA с циклическим кроссинговером;

Алгоритм	MA_{GOX}	MA_{ROX}	MA_{CX}	MA_{1PX}	MA_{B-MIP}	MA_{MIP}
среднее, %	6,1	6,9	8,9	9,2	34,2	39,7
ст.откл., %	1,6	1,4	1,0	1,5	3,4	6,2

Таблица 9 — Результаты MA для различных операторов кроссинговера (критерий суммы моментов завершения, относительное отклонение от рекорда, %).

Алгоритм	MA_{GOX}	MA_{ROX}	MA_{CX}	MA_{1PX}	MA_{B-MIP}	MA_{MIP}
среднее, %	16,1	17,9	21,9	23,2	57,4	49,7
ст.откл., %	3,5	4,4	7,2	6,8	10,1	8,4

Таблица 10 — Результаты MA для различных операторов кроссинговера (критерий взвешенной суммы заказов, выполненных в срок, относительное отклонение от рекорда, %).

MA_{1PX} – это MA с одноточечным кроссинговером;
MA_{B-MIP} – это MA с оптимизированным оператором кроссинговера на основе решения ЦЛП-подзадачи с верхней границей на число циклов и ограничением времени счета для решателя;
MA_{MIP} – это MA с оптимизированным оператором кроссинговера на основе решения ЦЛП-подзадачи с ограничением времени счета для решателя.

В таблицах 9, 10 показаны средние относительные отклонения результатов MA от лучшего найденного значения по всем запускам MA и решателей (Gurobi, CPLEX) на моделях целочисленного программирования из [254]. MA_{GOX} и MA_{ROX} демонстрируют статистически значимо лучшие результаты по сравнению с другими версиями MA на рассматриваемой серии (статистический анализ экспериментальных данных выполнен с использованием критерия Вилкоксона).

MA также сравнивался с алгоритмом муравьиной колонии (ACO), алгоритмом поиска с запретами (TS), имитацией отжига (SA) и генетическим алгоритмом (GA), предложенными в [128] для рассматриваемой задачи с критерием суммарного времени завершения. Для этой цели была построена серия примеров по тем же правилам, что и в [128], где $n,\ m \in \{5,\ 10,\ 15,\ 20\}$, длительности переналадок генерируют-

Алг.	MA_{GOX}	MA_{ROX}	MA_{CX}	MA_{1PX}	TS	ACO	GA	SA
ср., %	0,9	1,8	1,4	3,0	1,3	3,5	11,6	12,1
ст.от., %	0,1	0,4	0,6	0,4	0,02	0,2	4,1	8,1

Таблица 11 — Сравнение MA с известными метаэвристиками (критерий суммы моментов завершения, относительное отклонение от рекорда, %).

ся случайным образом с равномерным распределением из интервалов $(10, 20)$, $(1, 30)$, $(25, 35)$, $(15, 45)$, длительности операций генерируются случайным образом из интервала $(1, 15)$. Проведен вычислительный эксперимент на этих примерах с критериями суммы моментов завершения и взвешенной суммы заказов, выполненных в срок, используя MA и Gurobi. Таблицы 11–12 содержат относительные отклонения от рекорда для MA и Gurobi.

MA тестировался с ограничением по времени, эквивалентным [128] согласно соотношению тактовых частот процессоров. Размер популяции N полагался равным 100, параметр мутации $\alpha = 30\%$ и размер турнира $s = 5$ для обоих критериев на основе первой части эксперимента. Результаты ACO, TS, SA и GA взяты из [128]. Как видно из таблицы 11 MA_{GOX} показывает лучшие результаты среди всех сравниваемых алгоритмов, и различие статистически значимо. Алгоритмы MA_{CX}, MA_{ROX} и TS демонстрируют близкие результаты.

В таблице 12 показаны результаты запусков MA и Gurobi с теми же параметрами для критерия взвешенной суммы заказов, выполненных в срок. В [128] авторы не проводили эксперимент с этим критерием, поэтому сравнение невозможно. Здесь MA_{CX} демонстрирует немного лучший результат по сравнению с другими вариантами MA. Аналогично критерию суммы моментов завершения все варианты MA показывают результаты лучше, чем Gurobi.

Алгоритм	MA_{GOX}	MA_{ROX}	MA_{CX}	MA_{1PX}	Gurobi
среднее, %	5,4	5,2	4,6	5,6	34,2
ст.откл., %	0,7	1,2	1,2	1,5	12,1

Таблица 12 — Результаты MA и Gurobi для критерия взвешенной суммы заказов, выполненных в срок (относительное отклонение от рекорда, %).

4.5 Задача составления расписаний с учетом ограничения на расход энергии

Рассматривается задача $P2|size_j, energy|\sum C_j$ составления расписаний жестких распараллеливаемых работ на двух процессорах (или ядрах) с выбором скоростей. Множество работ как и ранее обозначается через $\mathcal{J} = \{1, \ldots, n\}$. Для каждой работы $j \in \mathcal{J}$ заданы объем V_j и количество $size_j$ требуемых процессоров. Значение $size_j$ для работы $j \in \mathcal{J}$ указывает, что для выполнения j требуется это количество процессоров. Все работы выполняются без прерываний.

Если процессор работает со скоростью s, то мгновенная мощность (потребление энергии в единицу времени) равна s^α, где $\alpha > 1$. Общий объем V_j жесткой работы $j \in \mathcal{J}$ равномерно распределяется между назначенными процессорами, т.е. если работа j использует $size_j$ процессоров, то объемы одинаковы для всех этих процессоров (обозначаются как $W_j := \frac{V_j}{size_j}$), и процессоры работают одновременно с одинаковой скоростью.

Цель – минимизировать сумму моментов завершения $\sum C_j$ при заданном бюджете на расход энергии E. Задача является NP-трудной [252].

Теоретический анализ задачи с однопроцессорными работами представлен в [20, 62, 202, 216]. Предложены полиномиальные алгоритмы для частных случаев, когда расписания строятся для различных уровней на расход энергии и выбирается лучшее из них [62, 202]. Точный полиномиальный алгоритм был представлен в [216] для общего многопроцессорного случая. Агрегированный критерий, включающий время выполнения и энергопотребление, исследовался в [20]. Предложен алгоритм динамического программирования для работ с одинаковыми объемами.

4.5.1 Операторы

Для рассматриваемой задачи предлагается адаптивный эволюционный алгоритм, основанный на схеме алгоритма 7. Решения кодируются перестановками работ. Для построения расписания и вычисления значения целевой функции согласно перестановке π сначала вычисляются длительности p_j работ путем решения следующей задачи выпуклого про-

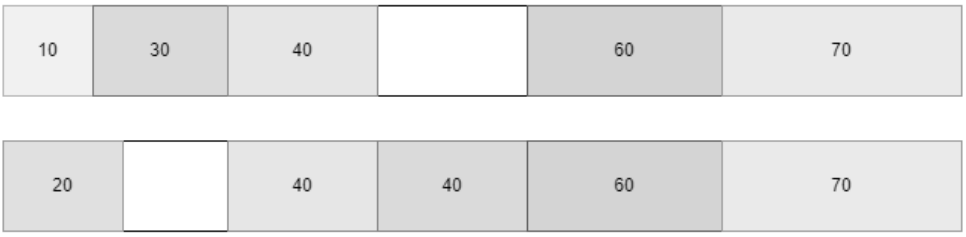


Рисунок 44 — Допустимое решение для задачи с двумя процессорами.



Рисунок 45 — Нижняя оценка на сумму моментов завершения.

граммирования:

$$\sum_{j\in\mathcal{J}} C_j(\pi) = \sum_{j=1}^{n}(n-j+1)p_{\pi_j} \to \min, \tag{221}$$

$$\sum_{j\in\mathcal{J}} (2p_j)^{1-\alpha}(V_j)^{\alpha} = E. \tag{222}$$

Затем работы назначаются в расписание в порядке, заданном перестановкой π (см. рис. 44). Решение задачи (221)–(222) также дает нижнюю оценку на целевую функцию, когда работы упорядочены по невозрастанию объемов W_j (см. рис. 45).

Особи начальной популяции строятся случайным образом или с использованием правила «присоединение к перестановке» [103]: заданное количество работ n_J размещается в случайном порядке, а каждая следующая работа выбирается случайно и вставляется в позицию среди уже размещенных, где значение целевой функции частичного решения минимально (такой способ применяется с вероятностью P_{IP}).

Критерий остановки алгоритма – количество вычислений целевой функции (обозначаемое через n_{max}). Используется классическое правило перезапуска: ЭА перезапускается, как только номер вычисления целевой функции становится равным номеру вычисления целевой функции, на котором было достигнуто крайнее лучшее найденное решение, плюс заданное количество вычислений целевой функции n_{FC}.

В операторе кроссинговера точно или приближенно решается задача оптимальной рекомбинации, которыя основывается на наследовании значений в позициях перестановок и отношения предшествования между работами. В первом случае имеет место детерминированная оптимизированная версия оператора CX, а во втором случае – детерминированная оптимизированная версия кроссинговера OX. Обе задачи оптимальной рекомбинации являются NP-трудными.

Теорема 73 *Задача оптимальной рекомбинации (I)–(II) для задачи составления расписаний* $P2|size_j, energy|\sum C_j$ *является NP-трудной.*

Доказательство. Доказательство NP-трудности основано на полиномиальной сводимости задачи Упорядоченное разбиение к распознавательной версии задачи составления расписаний. Представим доказательство для случая наследования значений в позициях перестановок. При наследовании отношений предшествования между работами доказательство аналогично.

Рассмотрим следующую NP-трудную задачу Упорядоченное разбиение [252]: Дано упорядоченное множество $A = \{a_1, a_2, \ldots, a_{2n_0}\}$ положительных целых чисел, таких что $\sum_{a_i \in A} a_i = 2C$, $a_i < a_{i+1},\ i = 1, \ldots, 2n_0 - 1$ и $a_{2i+1} > 3a_{2i}$ для $i = 1, \ldots, n_0 - 1$. Требуется определить, можно ли разбить множество A на два подмножества A_1 и A_2 так, чтобы

$$\sum_{a_i \in A_1} a_i = \sum_{a_i \in A_2} a_i = C,\ |A_1| = |A_2| = n_0,$$

и подмножество A_1 содержало только один элемент из каждой пары $a_{2i-1},\ a_{2i},\ i = 1, \ldots, n_0$.

Построим семейство индивидуальных задач составления расписаний с $n_1 = 2n_0$ однопроцессорными работами и одной двухпроцессорной работой, $n = 2n_0 + 1$. Положим $V_{2j-1} = a_{2j-1}(n_0 - j + \frac{3}{2})^{\frac{1}{\alpha}},\ size_{2j-1} = 1$ и $V_{2j} = a_{2j}(n_0 - j + \frac{3}{2})^{\frac{1}{\alpha}},\ size_{2j} = 1$ для $j = 1, \ldots, n_0$ и $V_{2n_0+1} = 2^{\frac{\alpha-1}{\alpha}} M,\ size_{2n_0+1} = 2$. Здесь M – большая константа, например,

$$M := \max\left\{(2n_0+1)^{\frac{\alpha+1}{\alpha}} \left(\frac{3}{2}\right)^{\frac{1}{\alpha}} a_{2n_0}; \sum_{j=1}^{n_0} \left(n_0 - j + \frac{3}{2}\right)(a_{2j} + a_{2j-1})\right\} + 1.$$

Установим бюджет на расход энергии $E = \sum_{j=1}^{n_0}(n_0 - j + \frac{3}{2})(a_{2j-1} + a_{2j}) + M$.

Зададим родительские перестановки для процессоров:
первый родитель
$\pi_1^1 = (1,\ 3,\ \ldots,\ 2n_0 - 1,\ 2n_0 + 1)$,
$\pi_2^1 = (2,\ 4,\ \ldots,\ 2n_0,\ 2n_0 + 1)$,
второй родитель
$\pi_1^2 = (2,\ 4,\ \ldots,\ 2n_0,\ 2n_0 + 1)$,
$\pi_2^2 = (1,\ 3,\ \ldots,\ 2n_0 - 1,\ 2n_0 + 1)$.

Заметим, что объемы V_j упорядочены по возрастанию, как и a_j, и $\frac{V_{2j+1}}{V_{2j}} = \frac{a_{2j+1}}{a_{2j}} \left(\frac{n-j+1/2}{n-j+3/2}\right)^{\frac{1}{\alpha}} > 3^{\frac{\alpha-1}{\alpha}} > 1$.

Установим пороговое значение для целевой функции, равное E. Покажем, что допустимое расписание S с $\sum_j C_j(S) \le E$ существует тогда и только тогда, когда задача Упорядоченное разбиение имеет положительный ответ.

Вычислим нижнюю оценку на сумму моментов завершения работ при заданной последовательности работ $\pi = (\pi_1, \ldots, \pi_{2n_0}, 2n_0 + 1)$ (см. рис. 46):

$$\sum_{j=1}^{n_0} (n_0 - j + 1)\,(p_{\pi_{2j-1}} + p_{\pi_{2j}}) + \frac{1}{2}\sum_{j=1}^{n_0}(p_{\pi_{2j-1}} + p_{\pi_{2j}}) + p_{2n_0+1} \to \min, \quad (223)$$

$$\sum_{j=1}^{n_0}(p_{2j-1}^{1-\alpha} V_{2j-1}^{\alpha} + p_{2j}^{1-\alpha} V_{2j}^{\alpha}) + 2 \cdot p_{2n_0+1}^{1-\alpha} \left(\frac{V_{2n_0+1}}{2}\right)^{\alpha} = E. \quad (224)$$

Представленная модель имеет линейную целевую функцию и выпуклое ограничение на потребление энергии. Используя необходимые и достаточные условия Каруша-Куна-Таккера [166], получаем оптимальное решение

$$p_{\pi_{2j}} = \left(n_0 - j + \frac{3}{2}\right)^{-\frac{1}{\alpha}} \cdot V_{\pi_{2j}} \cdot G,$$

$$p_{\pi_{2j-1}} = \left(n_0 - j + \frac{3}{2}\right)^{-\frac{1}{\alpha}} \cdot V_{\pi_{2j-1}} \cdot G,$$

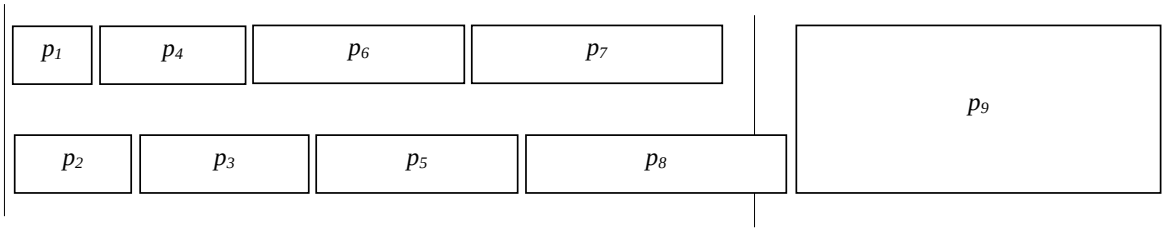


Рисунок 46 — Иллюстративный пример расписания.

$$p_{2n_0+1} = 2^{\frac{1}{\alpha}} \cdot \left(\frac{V_{2n_0+1}}{2}\right) \cdot G,$$

$$G = \left(\frac{\sum_{j=1}^{n_0} \left(n_0 - j + \frac{3}{2}\right)^{\frac{\alpha-1}{\alpha}} \left(V_{\pi_{2j}} + V_{\pi_{2j-1}}\right) + V_{2n_0+1} 2^{\frac{1-\alpha}{\alpha}}}{E}\right)^{\frac{1}{\alpha-1}}.$$

Оптимальное значение целевой функции (нижняя оценка) равно

$$\left(\frac{\left(\sum_{j=1}^{n_0} \left(n_0 - j + \frac{3}{2}\right)^{\frac{\alpha-1}{\alpha}} \left(V_{\pi_{2j}} + V_{\pi_{2j-1}}\right) + V_{2n_0+1} 2^{\frac{1-\alpha}{\alpha}}\right)^{\alpha}}{E}\right)^{\frac{1}{\alpha-1}}. \qquad (225)$$

Последовательность $\left\{n_0 - j + \frac{3}{2}\right\}$ убывает, поэтому упорядочение работ по неубыванию значений V_j дает нижнюю оценку:

$$\sum_{j=1}^{n_0} (n_0 - j + 1)\left(a_{2j-1} + a_{2j}\right) + \frac{1}{2}\sum_{j=1}^{n_0}(a_{2j-1} + a_{2j}) + M.$$

Заметим, что в этом случае $G = 1$, $p_j = a_j$ для $j = 1, \ldots, 2n_0$. Следовательно, допустимое расписание S с последней работой $2n_0 + 1$, согласующееся с родительскими решениями и имеющее целевую функцию $\sum C_j(S) \leq E$, существует тогда и только тогда, когда задача Упорядоченное разбиение имеет положительный ответ. Распознавательная версия ЗОР NP-полна, значит оптимизационная версия ЗОР NP-трудна. □

Использование жадных методов для решения ЗОР практически оправдано ввиду ее трудноре шаемости, так как оператор кроссинговера применяется итеративно в ЭА. В экспериментальных исследованиях будут использоваться оптимизированные операторы, основанные на дерандо-

мизации кроссинговера $1PX$:

Частично оптимизированный одноточечный кроссинговер (PO_1PX). Вместо случайного выбора одной точки в $1PX$, выбирается $x\%$ всех возможных точек. Одноточечный кроссинговер применяется к родителям с каждой из выбранных точек, и вычисляется значение целевой функции для потомков. Лучший найденный потомок является результатом частично оптимизированного $1PX$.

Оптимизированный одноточечный кроссинговер (O_1PX). Это полная версия кроссинговера PO_1PX с $x = 100$.

Оператор кроссинговера применяется с вероятностью P_{Cross}, а оператор мутации — с вероятностью P_{Mut}.

Также предлагается адаптивная схема выбора оператора кроссинговера (см. Алгоритм 8), которая применяется на каждой итерации эволюционного алгоритма [49]. Здесь $w_i,\ i = 1, \ldots, 3$ –настраиваемые параметры, λ – так называемый параметр затухания, который контролирует, насколько чувствительны веса к изменениям в работе операторов.

Алгоритм 8 Адаптивная схема выбора оператора кроссинговера.

1: Выбрать оператор кроссинговера a. Вероятность выбора каждого оператора пропорциональна его весу.
2: Применить оператор кроссинговера a к родительским особям.
3: Вычислить оценку выбранного оператора кроссинговера a:

$$\phi_a = \begin{cases} w_1, & \text{если найденное решение является новым лучшим найденным решением (среди всех перезапусков),} \\ w_2, & \text{если найденное решение лучше, чем лучшее найденное решение с момента последнего перезапуска,} \\ w_3, & \text{если найденное решение лучше одного из родителей,} \\ 0 & \text{иначе.} \end{cases}$$

4: Вес оператора кроссинговера a обновляется следующим образом:

$$\rho_a := \lambda\rho_a + (1 - \lambda)\phi_a.$$

Следует обратить внимание, что в алгоритме присутствует период обучения, задаваемый как процент от количества вычислений целевой функции, в который все операторы кроссинговера имеют равные веса.

Название параметра	Значение параметра
k	200
n_e	2
P_{IP}	0.2
$Selection$	ранговая
P_{Cross}	0.8
$Crossover$	1PX
P_{Mut}	0.2
$Mutation$	сдвига
n_{FC}	1000 * 200
w_1, w_2, w_3	1, 0.8, 0.3

Таблица 13 — Параметры ЭА, найденные в предварительном вычислительном эксперименте.

4.5.2 Вычислительный эксперимент

Все вычисления, описанные в этом разделе, выполнялись на компьютере с 16 ГБ ОЗУ и процессором Intel Core i7-8565U 1.80 ГГц. В качестве языка программирования использовалась Java версии 11.0.6. Было сгенерировано 30 тестовых примеров с $n = 100$ работами с равными вероятностями выбора малого или большого объема и типа работы (однопроцессорная или двухпроцессорная). Набор вариантов больших объемов: $LARGE = \{200, 275, 350, 425, 500, 575, 650, 725, 800, 875\}$ и набор вариантов малых объемов: $SMALL = \{10, 20, 30, 40, 50, 60, 70, 80, 90,\ 100\}$.

Значения параметров эволюционного алгоритма были выбраны в предварительном эксперименте. Значения выбирались из следующих множеств: $k \in \{100, 200\}$, $n_e \in \{0, 2, k-2\}$, $P_{IP} \in \{0.2, 0.5, 0.8\}$, $Crossover \in \{1PX, CX, OX, PMX\}$. Конфигурация значений, демонстрирующая наилучшие результаты, представлена в таблице 13. Отметим, что кроссинговер $1PX$ демонстрирует лучшие результаты, чем другие операторы, поскольку сохраняет начальную часть перестановки, где работы с малыми объемами имеют тенденцию быть размещенными в исследуемой задаче. Адаптивная версия ЭА с рандомизированными операторами кроссинговера $1PX$, CX, OX и PMX показала конкуренцию операторов на начальном этапе, но на конечном этапе $1PX$ также становится одним из лидеров.

Название параметра	Значение параметра
k	244
n_e	146
P_{IP}	0.43
$Selection$	rank
P_{Cross}	0.7
$Crossover$	1PX
P_{Mut}	0.63
$Mutation$	swap

Таблица 14 — Параметры ЭА, найденные пакетом IRACE.

Также протестирован адаптивный ЭА с $1PX$ и его оптимизированными версиями PO_1PX и O_1PX. Дополнительно использовался пакет IRACE [179] для настройки параметров алгоритма. Значения параметров, найденные пакетом IRACE, представлены в таблице 14. Эволюционный алгоритм также сравнивался с известными жадными алгоритмами из [251], разработанными для той же задачи.

Таблица 15 показывает среднее, минимальное и максимальное относительное отклонение результатов от нижней оценки целевой функции для различных алгоритмов (критерий остановки – ограничение на число вычислений целевой функции, равное 15000×200):

- EA_{custom} – ЭА с параметрами из таблицы 13;
- EA_{adapt} – ЭА с адаптацией для рандомизированных операторов кроссинговера $1PX$, CX, OX, PMX;
- $EA_{optimized}$ – ЭА с адаптацией для оптимизированных операторов кроссинговера $1PX$, PO_1PX, O_1PX;
- EA_{irace} – ЭА с параметрами, найденными пакетом IRACE (таблица 14);
- GR — жадная эвристика из [251];
- GR_{LI} — жадная эвристика с локальными улучшениями из [251].

Также был проведен статистический анализ экспериментальных данных (целевых функций для найденных решений) с использованием кри-

	EA_{custom}	EA_{adapt}	$EA_{optimized}$	EA_{irace}	GR	GR_{LI}
avg	2.03%	2.06%	1.95%	1.99%	8.33%	4.56%
min	0.83%	0.83%	0.78%	0.82%	2.19%	1.67%
max	3.83%	3.88%	3.57%	3.86%	16.38%	7.74%

Таблица 15 — Относительные отклонения результатов от нижней оценки целевой функции для алгоритмов.

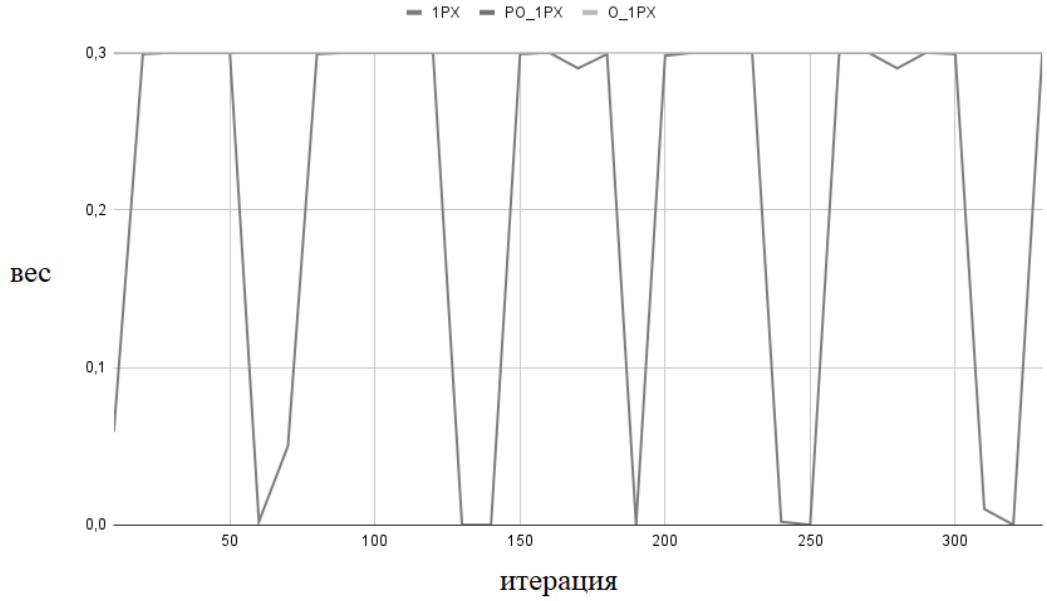


Рисунок 47 — Динамика весов операторов кроссинговера во время работы $EA_{optimized}$.

терия Вилкоксона. $EA_{optimized}$ демонстрирует статистически значимо лучшие результаты, чем остальные алгоритмы на рассматриваемой серии. EA_{custom} и EA_{adapt} не имеют статистически значимых различий в результатах. EA_{irace} показывает статистически значимо худшие результаты, чем $EA_{optimized}$, но статистически значимо лучшие результаты, чем EA_{custom} и EA_{adapt}. Все предложенные эволюционные алгоритмы демонстрируют среднее отклонение около 2% и не более 5%, т.е. строят решения хорошего качества на тестируемой серии.

Динамика весов операторов кроссинговера во время работы ЭА показана на рис. 47. Веса оптимизированных операторов держатся на максимальном уровне, успешно улучшая решения. Вес рандомизированного оператора изменяется во время выполнения алгоритма, впадины на графике соответствуют перезапускам.

	$GA_{optimized}$	GA_{irace}
avg	1.73%	1.8%
min	0.66%	0.76%
max	3.17%	3.22%

Таблица 16 — Относительные отклонения для тестовой выборки.

Отметим, что все результаты, представленные в таблице 15, статистически значимо лучше результатов, найденных эвристическими алгоритмами из [251], ранее предложенными для рассматриваемой задачи.

Для тестирования были сгенерированы так же новые 30 примеров с $n = 100$ работами, и на них проведено экспериментальное сравнение $GA_{optimized}$ и GA_{irace}. Результаты представлены в таблице 16. $GA_{optimized}$ демонстрирует статистически значимо лучшие результаты, чем GA_{irace}, как и на обучающей выборке.

4.6 Задача составления энергетически эффективных расписаний

Рассматривается задача составления расписаний множества работ $J = \{1, \ldots, n\}$ на одном процессоре. Каждая работа характеризуется моментом поступления r_j, директивным сроком d_j и объемом W_j, $j \in J$. Если процессор выполняет работу j со скоростью s_j, то мгновенная мощность (потребление энергии в единицу времени) составляет $(s_j)^\alpha$, $\alpha > 1$. Требуется составить допустимое расписание выполнение работ, где каждая работа $j \in J$ выполняется в интервале $[r_j, d_j)$, при котором общее потребление энергии $E = \sum_{j \in J} \frac{W_j}{s_j}(s_j)^\alpha$ минимально.

Допустимые решения кодируются перестановками работ. Для заданной перестановки исходная задача может быть сведена к согласованному случаю и решена за полиномиальное время. Действительно, пусть имеется перестановка $\pi = (\pi_1, \ldots, \pi_n)$ работ. Рассмотрим пару индексов $i < j$ и скорректируем времена поступления и директивные сроки следующим образом: $r'_{\pi_j} = \max\{r_{\pi_i},\ r_{\pi_j}\}$ и $d'_{\pi_i} = \min\{d_{\pi_i},\ d_{\pi_j}\}$. Повторим эту процедуру для всех пар работ и получим полиномиально разрешимый согласованный частный случай. Таким образом, значение целевой функции $E(\pi)$ для перестановки π может быть вычислено за полиномиальное

время.

Предыдущие исследования посвящены различным конструктивным приближенным алгоритмам с гарантированными оценками точности (см., например, [30,32,115]). В работе [27] предложен $2^{4\alpha-3}$-приближенный алгоритм для примеров с вложенной структурой допустимых интервалов выполнения работ и $2^{5\alpha-4}$-приближенный алгоритм для общего случая. Последний результат был улучшен для малых значений α до $2^{\alpha-1}\tilde{B}_\alpha$-приближенного алгоритма [32], где $\tilde{B}_\alpha$ – обобщенное число Белла. В [30] авторы показали, что отношение между оптимальным расписанием задачи без прерываний и оптимальным расписанием с прерываниями может быть очень большим в общем случае. Авторы также исследовали свойства оптимальных расписаний с прерываниями и предложили $\left(1+\frac{W_{\max}}{W_{\min}}\right)^{\alpha}$-приближенный алгоритм для задачи без прерываний, где $W_{\max}=\max_{j\in\mathcal{J}} W_j$ и $W_{\min}=\min_{j\in\mathcal{J}} W_j$.

В работах [147,173,174] предложены приближенные алгоритмы и эвристики для энергоэффективного планирования с общим временем поступления и директивным сроком для всех работ. Авторы рассматривают однопроцессорные работы с отношениями предшествования и независимые многопроцессорные работы. Используются методы, основанные на уровнях, и трехэтапные методы: разбиение на подзадачи, составление расписания работ и учет ресурсов.

Отметим, что ранее в литературе не было представлено экспериментальное исследование эвристических алгоритмов для $1|r_j,d_j|E$. В диссертации предлагается алгоритм локального поиска [180,229] с двумя типами окрестностей для $1|r_j,d_j|E$. Первый тип строится на основе одного решения и использует стратегии обмена и сдвига с адаптацией к специфике задачи. Второй тип определяется на основе пар решений и основан на наследовании значений в позициях перестановок решений как в задаче оптимальной рекомбинации ЭА. Экспериментальное исследование на примерах различной структуры показывает, что алгоритм демонстрирует конкурентоспособные результаты. Также приводится сравнение с эволюционным алгоритмом, основанным на стационарной схеме воспроизведения. Исследуется вычислительная сложность задачи оптимальной рекомбинации.

4.6.1 Операторы

Окрестности. Методы локального поиска успешно применяются для решения задач составления расписаний на перестановках (см., например, [114, 157, 180, 239]). В текущем исследовании предлагается алгоритм локального поиска, основанный окрестностях малой и большой размерности для рассматриваемой задачи. Основная идея заключается в применении стратегии «малая окрестность → большая окрестность → малая окрестность» к популяции решений.

Используются такие классические малые окрестности для перестановок, как сдвиг и обмен. В случае окрестности обмена рассматриваются решения, в которых позиции двух работ меняются местами. Окрестность сдвига строится путем вставки работы в некоторую другую позицию. Отметим, что моменты поступления и директивные сроки задают частичный порядок между работами: если $d_i < r_j$, то работа i должна предшествовать работе j. Поэтому в окрестностях переставляются только независимые с точки зрения частичного порядка работы.

Оптимальная рекомбинация будет играть роль большой окрестности. Напомним, что оптимальную рекомбинацию можно также рассматривать как улучшающий шаг в большой окрестности, определяемой двумя решениями.

Для $1|r_j, d_j|E$ будем использовать оптимальную рекомбинацию (I)–(II), основанную на наследовании значений в позициях перестановки. Преимущество такого подхода при решении $1|r_j, d_j|E$ заключается в том, что ЗОР с кодировкой на основе значений в позициях направлена на сохранение работ в позициях из родительских решений. Это важно для согласованности с моментами поступления и директивными сроками. Оптимальная рекомбинация в свою очередь будет использоваться как тунель между локальными оптимумами, построенными при малых окрестностях [229].

Теорема 74 *Задача оптимальной рекомбинации (I)-(II) для* $1|r_j, d_j|E$ *является NP-трудной даже в случае, когда все работы, кроме одной, имеют одинаковые моменты поступления и директивные сроки.*

Доказательство. Для доказательства NP-полноты распознователь-

ной версии ЗОР для $1|r_j, d_j|E$ используем задачу *Упорядоченное разбиение* [111]: дано упорядоченное множество $A = \{a_1, a_2, \ldots, a_{2n_0}\}$ и вес e_i каждого элемента $a_i \in A$, где

$$\sum_{a_i \in A} e_i = 2C,\ e_i > e_{i+1},\ i = 1, \ldots, 2n_0 - 1, \tag{226}$$

Заметим, что $C = \frac{1}{2}\sum_{i=1}^{2n_0} e_i$. Вопрос заключается в том, можно ли разбить множество A на два подмножества A_1 и A_2 таких, что $\sum_{a_i \in A_1} e_i = \sum_{a_i \in A_2} e_i = C,\ |A_1| = |A_2| = n_0$ и подмножество A_1 содержит только один элемент из каждой пары $a_{2i-1},\ a_{2i}$.

Пусть количество работ $n = 2n_0 + 1$. Работы $j \in \{1, \ldots, 2n_0\}$ имеют объемы $W_j = e_j$, моменты поступления $r_j = 0$ и директивные сроки $d_j = 3$. Последняя работа имеет объем $W_{2n_0+1} = 2$, момент поступления $r_{2n_0+1} = 1$ и директивный срок $d_{2n_0+1} = 2$.

Зададим родительские перестановки для ЗОР:
$\pi^1 = (1, 3, \ldots, 2n_0 - 1, 2n_0 + 1, 2, 4, \ldots, 2n_0)$,
$\pi^2 = (2, 4, \ldots, 2n_0, 2n_0 + 1, 1, 3, \ldots, 2n_0 - 1)$.

Покажем, что перестановка π', для которой выполняется условие (I) и $E(\pi') \leq C^\alpha(2 + 2^\alpha)$, существует тогда и только тогда, когда задача Упорядоченное разбиение имеет положительный ответ.

Действительно, в силу строгой выпуклости функции расчета мгновенной мощности s^α, легко видеть, что индивидуальная задача Упорядоченное разбиение имеет требуемое разбиение тогда и только тогда, когда расписание использует одинаковую скорость, равную B, имея одинаковый суммарный объем B для подмножеств работ J_1 и J_2 в интервалах $[0, 1)$ и $[2, 3)$. Такое расписание характеризуется суммарным потреблением энергии $E = 1 \cdot C^\alpha + 1 \cdot (2C)^\alpha + 1 \cdot C^\alpha = C^\alpha(2 + 2^\alpha)$ (см. рис. 48). Таким образом, распознавательная версия ЗОР NP-полна, значит оптимизационная версия ЗОР NP-трудна. □

Для решения ЗОР для задачи $1|r_j, d_j|E$ будет использоваться метод на основе перебора совершенных паросочетаний в специальном двудольном графе (см. раздел 3.1.2). Здесь перебираются допустимые перестановки и для них вычисляется значение целевой функции с использованием известного алгоритма, предложенного в [246] для задачи с прерыва-

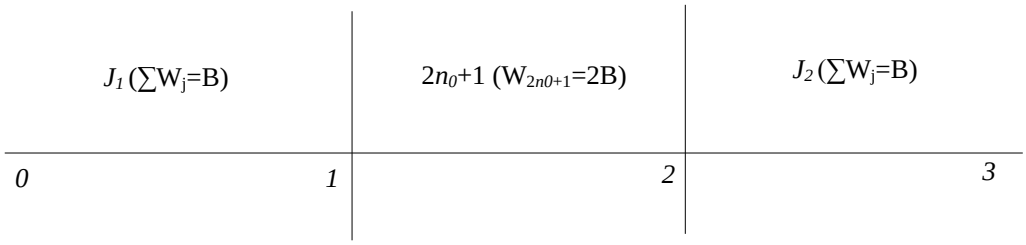


Рисунок 48 — Иллюстрация к доказательству NP-трудности.

ниями при согласованных моментах поступления и директивных сроках. В результате оптимальное решение ЗОР может быть найдено за время $O(1.42^n n^2)$. Более того, «почти все» пары родительских перестановок дают ЗОР (I)-(II), имеющие не более n допустимых решений, и могут быть решены за время $O(n^3)$.

Алгоритмы локального поиска. Предлагаемые алгоритмы локального поиска основываются на улучшении решений, полученных после локальных улучшений в окрестностях обмена или сдвига, с помощью оператора оптимальной рекомбинации. Начальная популяция строится случайным образом и эти решения улучшаются с помощью классической техники локального поиска с окрестностями обмена или сдвига и стратегией «иди в первый улучшающий». Затем решения случайным образом перемешиваются и скрещиваются по следующему правилу. Сначала оптимальная рекомбинация $OR(\cdot)$ применяется к первым двум локальным оптимумам в последовательности. Затем потомок скрещивается с третьим локальным оптимумом из последовательности, и так далее. Формальное описание представлено в Алгоритме 9. Обозначим этот алгоритм как $PLS - One$.

Отметим, что оптимальная рекомбинация позволяет получить более значительную перестройку решения за счет обмена блоками родительских решений.

Алгоритм 9 Алгоритм популяционного локального поиска.

1: Построить начальную популяцию из k перестановок (допустимых в соответствии с моментами поступления и директивными сроками).
2: Применить локальный поиск на основе окрестности обмена или сдвига к каждой перестановке.
3: Сгенерировать случайную последовательность перестановок $\pi^1, \ldots, \pi^k$. Положить $\pi' = OR(\pi^1, \pi^2)$.
4: Для $i = 3$ до k построить $\pi' = OR(\pi', \pi^i)$.
5: Улучшить π' с помощью локального поиска в окрестности обмена или сдвига.
6: Вернуть π' и $E(\pi')$.

Алгоритм 10 Алгоритм генетического локального поиска.

1: Построить начальную популяцию из k перестановок (допустимых в соответствии с моментами поступления и директивными сроками).
2: Применить локальный поиск на основе окрестности обмена или сдвига к каждой перестановке.
3: Пока не выполнится условие останова, выполнять

3.1 Выбрать две родительские перестановки π^1 и π^2.

3.2 Применить мутацию обмена или сдвига к перестановкам π^1 и π^2.

3.3 Положить $\pi' = OR(\pi^1, \pi^2)$.

3.4 Заменить худшую перестановку популяции на π'.

4: Улучшить рекордное решение с помощью локального поиска с окрестностями обмена или сдвига.
5: Вернуть наилучшее найденное решение.

Также применялся несколько иной подход, где оптимальная рекомбинация используется для скрещивания различных комбинаций решений, полученных в локальном поиске. Здесь запускается Алгоритм 9 несколько раз с разными случайными последовательностями, и наилучшие ре-

шения, найденные в каждом запуске, затем скрещиваются по схеме Алгоритма 9. Такой алгоритм обозначается через $PLS - Many$.

Для сравнения также разработан алгоритм генетического локального поиска с оптимальной рекомбинацией и операторами мутации обмена или сдвига в рамках стационарной схемы воспроизведения (см. Алгоритм 10). Используется турнирная селекция. Этот генетический алгоритм обозначается через GLS.

4.6.2 Вычислительный эксперимент

Алгоритмы протестированы на сгенерированных случайным образом тестовых примерах с $n = 50$ и $n = 100$ работами. Момент поступления r_j выбирался случайным образом из интервала $[0,\ 20]$, директивный срок d_j – из $[r_j + 1,\ r_j + 11]$, объем W_j – из $[5,\ 15]$ для каждой работы $j \in \mathcal{J}$ независимо. Проведено тестирование на двух сериях $S^n_{\alpha=2}$ с $\alpha = 2$ и $S^n_{\alpha=3}$ с $\alpha = 3$ для $n = 50,\ n = 100$. Каждая серия содержит 30 примеров. Во всех экспериментах установлены размер популяции $k = 100$, размер турнира $s = 5$ на основе предварительных вычислительных экспериментов. Алгоритмы были реализованы на языке Java. Вычислительный эксперимент проводился на компьютере с процессором Intel Core i3-10100F 3.6 ГГц, 16 Гб ОЗУ. Будут представлены результаты для случая выбора окрестности сдвига в качестве малой окрестности. Результаты для окрестности обмена аналогичны.

Относительные отклонения от нижней оценки целевой функции представлены в таблице 17. Здесь нижняя оценка берется как оптимальное значение целевой функции соответствующей задачи с прерываниями. Для GLS был установлен критерий остановки по времени, равному времени работы $PLS - Many$.

Таблица 18 содержит вспомогательные результаты для анализа предложенных алгоритмов:
столбец $Init$: результаты, соответствующие наилучшим решениям в начальной популяции (шаг 1 алгоритмов);
столбец LS: результаты, полученные в Алгоритме 9 после шага 2, когда применены только локальные улучшения на основе малых окрестностей;
столбец $All - To - All$: результаты, когда локальные оптимумы, полу-

Серия	$PLS-One$			$PLS-Many$			GLS		
	min	aver	max	min	aver	max	min	aver	max
$S^{50}_{\alpha=2}$	0	3,1	15,5	0	0,9	6,1	0	1,7	8,2
$S^{50}_{\alpha=3}$	0	34,9	9,8	0	2,1	10,1	0	3,2	12,7
$S^{100}_{\alpha=2}$	9,7	14,2	20,0	1,7	3,1	6,1	2,8	5,2	7,9
$S^{100}_{\alpha=3}$	16,7	63,1	32,7	0,5	5,8	15,1	0,8	8,8	17,3

Таблица 17 — Относительное отклонение от нижней оценки (в процентах).

Серия	$Init$			LS			$All-To-All$		
	min	aver	max	min	aver	max	min	aver	max
$S^{50}_{\alpha=2}$	25	60	111	8	19	40	0,1	6	15
$S^{50}_{\alpha=3}$	114	191	383	15	50	115	0,3	15	40
$S^{100}_{\alpha=2}$	53	72	87	37	51	66	24	16	31
$S^{100}_{\alpha=3}$	143	232	326	105	155	221	37	49	66

Таблица 18 — Относительное отклонение от нижней оценки (в процентах).

ченные на шаге 2, скрещиваются с помощью оптимальной рекомбинации друг с другом, и возвращается наилучшее найденное решение.

Как видно из представленных таблиц, $PLS-Many$ демонстрирует лидирующие результаты, GLS занимает второе место, а последнее место принадлежит $PLS-One$. Также проведен статистический анализ экспериментальных данных с использованием критерия Вилкоксона. Результаты показывают, что разница между рекордными значениями $PLS-Many$ и GLS (а также между GLS и $PLS-One$) статистически значима при уровне значимости $\alpha = 0.05$ для всех серий. Отметим, что все разработанные алгоритмы локального поиска значительно улучшают начальную популяцию локальных оптимумов с помощью оптимальной рекомбинации. Таким образом, решения после локального поиска в малых окрестностях далеки от глобального оптимума, а оптимальная рекомбинация значительно улучшает их при поиске в больших окрестностях.

Простое попарное скрещивание локальных оптимумов ($All-To-All$) приводит к худшим результатам, чем $PLS-One$, при использовании

большего числа вызовов оператора рекомбинации с большой окрестностью. Результаты экспериментов также показали, что решения, полученные на последней итерации оптимальной рекомбинации (шаг 4), были локально оптимальными по крайней мере в 88% случаев для $PLS - Many$ и по крайней мере в 50% случаев для $PLS - One$. Таким образом, оптимальная рекомбинация позволяет получать новые локальные оптимумы. Здесь локальную оптимальность понимается в контексте окрестности сдвига.

Также алгоритмы тестировались на примерах, сгенерированных по правилам из [30] с известными оптимальными решениями, но в этом случае оптимумы находились на этапе работы с малыми окрестностями (шаг 2 Алгоритма 9).

Заключение

В диссертации проведено исследование вычислительной сложности, построены новые математические модели и эффективные точные и приближенные алгоритмы оптимизации для актуальных задач составления расписаний с альтернативами выполнения работ, сложными технологическими связями и ресурсными ограничениями, возникающими на производстве и в компьютерных системах. Получены следующие основные результаты.

1. Предложен двухэтапный метод решения задачи составления энергетически эффективных расписаний с распараллеливаемыми и многопроцессорными работами. На первом этапе вычисляется нижняя оценка целевой функции и оцениваются характеристики работ. На втором этапе реализуется переход к задаче с фиксированными длительностями работ и строится допустимое расписание с гарантированной оценкой точности. Построены серии примеров, на которых оценки точности достигаются. Предложен метод доказательства NP-трудности на основе тех же принципов, что и двухэтапный метод решения.

2. Предложены методы построения нижних оценок на потребление энергии в задачах составления расписаний с распараллеливаемыми и многопроцессорными работами, которые основаны:
   – на сведении релаксации исходной задачи к специального вида задаче о максимальном потоке минимальной стоимости;
   – на представлении исходной задачи как задачи линейного программирования с экспоненциальным числом переменных и полиномиальным числом ограничений и решении ее методом эллипсоидов с использованием полиномиального оракула отсечения.

3. Впервые исследованы вопросы труднорешаемости задачи теории расписаний с многопроцессорными и распараллеливаемыми работами с классическими критериями при бюджетных ограничениях на расход энергии, выделены полиномиально разрешимые и NP-трудные частные случаи.

4. Для задач с ограничением на расход энергии предложены подходы к построению нижних оценок на целевую функцию, которые основаны:

   – на представлении исходной задачи как задачи выпуклого программирования и решении ее с использованием условий Каруша-Куна-Таккера;

   – на представлении исходной задачи как задачи линейного программирования с экспоненциальным числом переменных и полиномиальным числом ограничений и решении ее методом эллипсоидов с использованием полиномиального оракула отсечения.

   При построении допустимых решений применяются либо жадные алгоритмы, либо методы декомпозиции.

   Указанные результаты являются первыми приближенными алгоритмами с гарантированными оценками точности для многопроцессорных и распараллеливаемых работ с учетом расхода энергии и обобщают известные результаты для однопроцессорных работ.

5. Предложен метод доказательства NP-трудности для задач составления расписаний с запретами работ.

6. Разработан точный метод решения для класса задач составления расписаний с предписаниями работ мощности не превосходящей 2, трудоемкость которого полиномиальна для «почти всех» систем предписаний. На основе этого подхода предложены модели целочисленного линейного программирования с меньшим числом переменных, чем дает классический подход. Метод основан на переборе совершенных паросочетаний в специальном двудольном графе и обобщен на случай предписаний произвольной мощности.

7. Предложены эволюционные алгоритмы с оптимизированными операторами, которые показывают конкурентоспособные результаты на тестовых примерах известных библиотек. Исследована вычислительная сложность задачи оптимальной рекомбинации.

8. Экспериментально показана перспективность использования оптимальной рекомбинации, а также ее статистически значимое превос-

ходство над рандомизированными аналогами в составе эволюционных алгоритмов.

9. Разработаны стратегии управления популяцией при использовании оптимизированных операторов. Построены схемы алгоритма локального поиска с применением оптимальной рекомбинации.

Для дальнейшего исследования представляет интерес обобщение результатов на случай, когда учитывается не только расход энергии, но и такие характерные для многопроцессорных компьютерных систем ресурсы, как память, шина данных, ядра и другие. В алгоритмическом анализе задач с запретами работ открытым остается вопрос о полиномиальной разрешимости «почти всех» индивидуальных задач в случае систем предписаний произвольной мощности, как и о сложности задачи оптимальной рекомбинации при произвольном числе родительских решений. Также остаются открытыми вопросы о вычислительной сложности частных случаев задачи с критерием суммы моментов завершения при бюджетном ограничении на расход энергии, когда имеется два специализированных процессора и когда распараллеливаемые (гибкие) работы имеют идентичные объемы.

Литература

[1] Боброва Е.А., Романова А.А., Сервах В.В. Сложность задачи построения циклических расписаний обработки однотипных деталей // Дискретн. анализ и исслед. опер. – 2013. – Т. 20, № 4. – С. 3–14.

[2] Бондаренко В. А. Многогранники с высокой плотностью графа и полиномиальная разрешимость задач комбинаторной оптимизации // Автоматика и телемеханика. – 1993. – Вып. 4. – С. 21—26.

[3] Гимади Э.Х., Залюбовский В.В., Севастьянов С. В. Полиномиальная разрешимость задач календарного планирования со складируемыми ресурсами и директивными срока // Дискретн. анализ и исслед. опер. – 2000. – Сер. 2. – Т. 7, № 1. – С. 9–34.

[4] Еремеев А.В., Коваленко Ю.В. О задаче составления расписаний с группировкой машин по технологиям // Дискретный анализ и исследование операций. — 2011. — Т. 18, № 5. — С. 54–79.

[5] Еремеев А.В., Коваленко Ю.В. О сложности оптимальной рекомбинации для одной задачи составления расписаний с переналадками // Дискретный анализ и исследование операций. — 2012. — Т. 19, № 3. — С. 13–26.

[6] Захарова Ю.В. Приближенные алгоритмы для вариантов задачи open shop с учетом расхода энергии // Труды Института математики и механики УрО РАН. — 2024. — Т. 30, № 4. — С. 117–133. — DOI: 10.21538/0134-4889-2024-30-4-117-133.

[7] Захарова Ю.В., Евтина А.О. Конструктивные алгоритмы для задачи составления расписаний на двух процессорах с критерием максимального временного смещения при учёте распараллеливания и расхода энергии // Прикладная дискретная математика. — 2025. — № 67. — С. 118–128. — DOI: 10.17223/20710410/67/7.

[8] Зинов В.И., Картак В.М., Валиахметова Ю.И. Решение многокритериальной задачи рационального размещения несущих стен с помощью

генетического алгоритма // Информатика и автоматизация. – 2025. – Т. 24, № 2. – С. 464–491.

[9] Коваленко Ю.В. О сложности оптимальной рекомбинации для задач составления расписаний в многостадийной системе поточного типа // Дискретный анализ и исследование операций. — 2016. — Т. 23, № 2. — С. 41–62. — DOI: 10.1134/S1990478916020071.

[10] Колоколов А.А., Заозерская Л.А. Построение и анализ оценок числа итераций для алгоритмов целочисленного программирования с использованием метода регулярных разбиений // Изв. вузов. Матем. – 2014. – № 1. – С. 41–54.

[11] Сердюков А.И. О задаче коммивояжёра при наличии запретов // Управляемые системы. — 1978. — Вып. 17. — С. 80–86.

[12] Симанчёв Р.Ю., Уразова И.В. Целочисленная модель с условиями оптимальности для задачи минимизации суммарного взвешенного времени обслуживания требований одним прибором // Дискретн. анализ и исслед. опер. 2025. – Т. 32, № 2. – С. 122–136.

[13] Танаев В.С., Сотсков Ю.Н., Струсевич В.А. Теория расписаний: Многостадийные системы. — М.: Наука, 1989. — 327 с.

[14] Хамисов О.В. Оптимизация функции оптимального значения в задачах выпуклого параметрического программирования // Тр. Ин-та математики и механики УрО РАН. – 2023. – Т. 29, № 3. – С. 247-260.

[15] Чирков А.Ю., Шевченко В.Н., Золотых Н.Ю. О многокритериальной задаче целочисленного линейного программирования // Дискретн. анализ и исслед. опер. – 2005. – Сер. 2. – Т. 12, № 2. – С. 72–84.

[16] Avci S., Akturk M.S., Storer R.H. A problem space algorithm for single machine weighted tardiness problems // IIE Transactions. — 2003. — Vol. 35. — P. 479–486.

[17] Abitz D., Hartmann T., Middendorf M. A weighted population update rule for PACO applied to the single machine total weighted tardiness

problem // GECCO '20: Proceedings of the 2020 Genetic and Evolutionary Computation Conference. — 2020. — P. 4–12.

[18] Adak Z., Arioglu Akan M.O., Bulkan S. Multiprocessor open shop problem: literature review and future directions // Journal of Combinatorial Optimization. — 2020. — Vol. 40. — P. 547–569. — DOI: 10.1007/s10878-020-00591-3.

[19] Aggarwal C.C., Orlin J.B., Tai R.P. An optimized crossover for maximum independent set // Operations Research. — 1997. — Vol. 45. — P. 225–234.

[20] Albers S., Fujiwara H. Energy-efficient algorithms for flow time minimization // ACM Transactions on Algorithms. — 2007. — Vol. 3, No. 4. — 17 p.

[21] Albers S., Antoniadis A., Greiner G. On multi-processor speed scaling with migration // Journal of Computer and System Sciences. — 2015. — Vol. 81. — P. 1194–1209.

[22] Albers S., Müller F., Schmelzer S. Speed scaling on parallel processors // Algorithmica. — 2014. — Vol. 68, No. 2. — P. 404–425.

[23] Albers S., Bampis E., Letsios D., Lucarelli G., Stotz R. Scheduling on power-heterogeneous processors // Information and Computation. — 2017. — Vol. 257. — P. 22–33.

[24] Angel E., Bampis E., Kacem F., Letsios D. Speed scaling on parallel processors with migration // Proceedings of the 18th International European Conference on Parallel and Distributed Computing. Euro-Par 2012. Lecture Notes in Computer Science. — Berlin, Heidelberg: Springer, 2012. — Vol. 7484. — P. 128–140.

[25] Anghinolfi D., Paolucci M. A new ant colony optimization approach for the single machine total weighted tardiness scheduling problem // International Journal of Operations Research. — 2008. — Vol. 5, No. 1. — P. 44–60.

[26] Antczak T. Optimality and duality for nonsmooth multiobjective programming problems with V-r-invexity // Journal of Global Optimization. — 2009. — Vol. 45, No. 2. — P. 319–334.

[27] Antoniadis A., Huang C.C. Non-preemptive speed scaling // Journal of Scheduling. — 2013. — Vol. 16, No. 4. — P. 385–394.

[28] Arkhipov D., Battaia O., Lazarev A. A. An efficient pseudo-polynomial algorithm for finding a lower bound on the makespan for the Resource Constrained Project Scheduling Problem // European Journal of Operational Research. — 2018. — Vol. 275, no. 1. — P. 35–44.

[29] Balas E., Niehaus W. Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems // Journal of Heuristics. — 1998. — Vol. 4, No. 2. — P. 107–122.

[30] Bampis E., Kononov A., Letsios D., Lucarelli G., Nemparis I. From preemptive to non-preemptive speed-scaling scheduling // Discrete Applied Mathematics. — 2015. — Vol. 181. — P. 11–20.

[31] Bampis E., Letsios D., Lucarelli G. A note on multiprocessor speed scaling with precedence constraints // SPAA '14: Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures. — 2014. — P. 138–142.

[32] Bampis E., Kononov A., Letsios D., Lucarelli G., Sviridenko M. Energy-efficient scheduling and routing via randomized rounding // Journal of Scheduling. — 2018. — Vol. 21. — P. 35–51.

[33] Bampis E., Kononov A., Letsios D., Lucarelli G., Nemparis I. From preemptive to non-preemptive speed-scaling scheduling // Discrete Applied Mathematics. — 2015. — Vol. 181. — P. 11–20.

[34] Bampis E., Letsios D., Lucarelli G. Green scheduling, flows and matchings // Theoretical Computer Science. — 2015. — Vol. 579. — P. 126–136.

[35] Bampis E., Letsios D., Milis I., Zois G. Speed scaling for maximum lateness // Theory of Computing Systems. — 2016. — Vol. 58. — P. 304–321.

[36] Bansal N., Kimbrel T., Pruhs K. Dynamic speed scaling to manage energy and temperature // Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science. — 2004. — P. 520–529.

[37] Bansal N., Pruhs K. Speed scaling to manage temperature // STACS 2005. Lecture Notes in Computer Science. — Berlin, Heidelberg: Springer, 2005. — P. 460–471.

[38] Bansal N., Bunde D.P., Chan H.L., Pruhs K. Average rate speed scaling // LATIN 2008: Theoretical Informatics. Lecture Notes in Computer Science. — 2008. — Vol. 4957. — P. 240–251.

[39] Bansal N., Pruhs K., Stein C. Speed scaling for weighted flow time // SIAM Journal on Computing. — 2009. — Vol. 39, No. 4. — P. 1294–1298.

[40] Baptiste P. A note on scheduling multiprocessor tasks with identical processing times // Computers and Operations Research. — 2003. — Vol. 30, No. 13. — P. 2071–2078.

[41] Barany I., Fiala T. Nearly optimum solution of multimachine scheduling problems // Szigma Mathematika Kozgazdasagi Folyoirat. — 1982. — Vol. 15. — P. 177–191. (In Hungarian)

[42] Belkhale K., Banerjee P. An approximate algorithm for the partitionable independent task scheduling problem // Proceedings of International Conference on Parallel Processing (ICPP-90). — 1990. — Vol. 1. — P. 72–75.

[43] den Besten M., Stutzle T., Dorigo M. Ant colony optimization for the total weighted tardiness problem // Parallel Problem Solving from Nature (PPSN-2000). Lecture Notes in Computer Science. — 2000. — Vol. 1917. — P. 611–620.

[44] Beyer H.-G., Schwefel H.-P. Evolution strategies – a comprehensive introduction // Natural Computing. — 2002. — Vol. 1. — P. 3–52.

[45] Bilge U., Kurtulan M., Kirac F. A tabu search algorithm for the single machine total weighted tardiness problem // European Journal of Operational Research. — 2007. — Vol. 176. — P. 1423–1435.

[46] Bingham B.D., Greenstreet M.R. Energy optimal scheduling on multiprocessors with migration // Proceedings of International Symposium on Parallel and Distributed Processing with Applications (ISPA 2008). — 2008. — P. 153–161.

[47] Blazewicz J., Drabowski M., Weglarz J. Scheduling multiprocessor tasks to minimize schedule length // IEEE Transactions on Computers. — 1986. — Vol. 35, No. 5. — P. 389–393.

[48] Blazewicz J., Dror M., Weglarz J. Mathematical programming formulations for machine scheduling: a survey // European Journal of Operational Research. — 1991. — Vol. 51, No. 3. — P. 283–300.

[49] Blum C., Eremeev A., Zakharova Y. Hybridizations of evolutionary algorithms with Large Neighborhood Search // Computer Science Review. — 2022. — Vol. 46. — 22 p. — DOI: 10.1016/j.cosrev.2022.100512.

[50] Bonomo F., Duran G., Marenco J. Exploring the complexity boundary between coloring and list-coloring // Annals of Operations Research. — 2009. — Vol. 169, No. 1. — P. 3–16.

[51] Borisovsky P.A., Eremeev A.V. Comparing evolutionary algorithms to the (1+1)-EA // Theoretical Computer Science. — 2008. — Vol. 403, No. 1. — P. 33–41.

[52] Borisovsky P., Dolgui A., Eremeev A. Genetic algorithms for a supply management problem: MIP-recombination vs greedy decoder // European Journal of Operational Research. — 2009. — Vol. 195, No. 3. — P. 770–779.

[53] Borisovsky P. Exact solution of one production scheduling problem // Optimization Problems and Their Applications. OPTA 2018. Communications in Computer and Information Science. — Cham: Springer, 2018. — Vol. 871. — P. 56–67. — DOI: 10.1007/978-3-319-93800-4_5.

[54] Borisovsky P., Eremeev A., Kallrath J. Multi-product continuous plant scheduling: combination of decomposition, genetic algorithm, and constructive heuristic // International Journal of Production Research. — 2020. — Vol. 58, No. 9. — P. 2677–2695.

[55] Borisovsky P., Kovalenko Y. A memetic algorithm with parallel local search for flowshop scheduling problems // Bioinspired Optimization Methods and Their Applications. BIOMA 2020. Lecture Notes in Computer Science. — Cham: Springer, 2020. — Vol. 12438. — P. 201–213.

[56] Bouchra D., Chaabane L. An enhanced genetic algorithm for the single machine total weighted tardiness problem // Romanian Journal of Information and Computer Science. — 2007. — Vol. 9, No. 1. — P. 66–77.

[57] Boyd S., Vandenberghe L. Convex Optimization. — Cambridge: Cambridge University Press, 2009. — 714 p.

[58] Brodtkorb A.R., Dyken C., Hagen T.R., Hjelmervik J.M., Storaasli O.O. State-of-the-art in heterogeneous computing // Scientific Programming. — 2010. — Vol. 18. — P. 1–33.

[59] Brown B.W., Hollander M. Statistics: A Biomedical Introduction. — New York: John Wiley & Sons, Inc., 1977.

[60] Brucker P., Knust S., Roper D., Zinder Y. Scheduling UET task systems with concurrency on two parallel identical processors // Mathematical Methods of Operations Research. — 2000. — Vol. 52, No. 3. — P. 369–387.

[61] Brucker P. Scheduling algorithms // Journal of the Operational Research Society. — 1999. — Vol. 50. — P. 774.

[62] Bunde D. Power-aware scheduling for makespan and flow // Journal of Scheduling. — 2009. — Vol. 12. — P. 489–500.

[63] Buriol L.S., Franca P.M., Moscato P. A new memetic algorithm for the asymmetric traveling salesman problem // Journal of Heuristics. — 2004. — Vol. 10. — P. 483–506.

[64] Burke E.K., Cowling P.I., Keuthen R. Effective local and guided variable neighborhood search methods for the asymmetric travelling salesman problem // Applications of Evolutionary Computing. EvoWorkshop 2001. Lecture Notes in Computer Science. — Berlin, Heidelberg: Springer, 2001. — Vol. 2037. — P. 203–212.

[65] Cai X., Lee C.-Y., Li C.-L. Minimizing total completion time in two-processor task systems with prespecified processor allocations // Naval Research Logistics. — 1998. — Vol. 45, No. 2. — P. 231–242.

[66] Cetinkaya F.C., Yeloglu P., Catmakas H.A. Customer order scheduling with job-based processing on a single-machine to minimize the total completion time // International Journal of Industrial Engineering Computations. — 2021. — Vol. 12, No. 3. — P. 273–292.

[67] Chen J., Hsu H., Chuang K., Yang C., Pang A., Kuo T. Multiprocessor energy-efficient scheduling with task migration considerations // Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS 2004). — IEEE, 2004. — P. 101–108.

[68] Chernykh I., Kononov A., Sevastyanov S. Efficient approximation algorithms for the routing open shop problem // Computers & Operations Research. — 2013. — Vol. 40, No. 3. — P. 841–847.

[69] Chernykh I. Routing open shop with unrelated travel times // Discrete Optimization and Operations Research. DOOR 2016. Lecture Notes in Computer Science. — Cham: Springer, 2016. — Vol. 9869. — P. 272–283.

[70] Chicano F., Ochoa G., Whitley D., Tinos R. Quasi-optimal recombination operator // Evolutionary Computation in Combinatorial Optimization. EvoCOP 2019. Lecture Notes in Computer Science. — Cham: Springer, 2019. — Vol. 11452. — P. 131–146.

[71] Chicano F., Ochoa G., Whitley D., Tinos R. Dynastic potential crossover operator // Evolutionary Computation. — 2022. — Vol. 30, No. 3. — P. 409–446.

[72] Chunga T.P., Fua Q., Liaoa C.J., Liua Y.T. On hybrid evolutionary algorithms for scheduling problem with tardiness criterion // Engineering Optimization. — 2017. — Vol. 49, No. 7. — P. 1133–1147.

[73] Chung T.-P., Fu Q., Liao C.-J., Liu Y.-T. Multiple-variable neighbourhood search for the single-machine total weighted tardiness problem // Engineering Optimization. — 2016. — Vol. 49, No. 7. — P. 1133–1147.

[74] Coffman E., Garey M., Johnson D., Lapaugh A. Scheduling file transfers // SIAM Journal on Computing. — 1985. — Vol. 14, No. 3. — P. 744–780.

[75] Cohen-Addad V., Li Z., Mathieu C., Milis I. Energy-efficient algorithms for non-preemptive speed-scaling // Approximation and Online Algorithms. WAOA 2014. Lecture Notes in Computer Science. — Cham: Springer, 2015. — Vol. 8952. — P. 107–118.

[76] Congram R., Potts C., van de Velde S. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem // INFORMS Journal on Computing. — 2002. — Vol. 14, No. 1. — P. 52–67.

[77] Conway R.W., Maxwell W.L., Miller L.W. Theory of Scheduling. — Reading, Mass.: Addison-Wesley, 1967. — [Russian translation: Теория расписаний. — М.: Наука, 1975. — 360 с.]

[78] Cook W., Seymour P. Tour merging via branch-decomposition // INFORMS Journal on Computing. — 2003. — Vol. 15, No. 2. — P. 233–248.

[79] Cormen T.H., Leiserson C.E., Stein C. Introduction to Algorithms. — Cambridge, MA: MIT Press, 2022.

[80] Cotta C., Alba E., Troya J.M. Utilizing dynastically optimal forma recombination in hybrid genetic algorithms // Parallel Problem Solving from Nature — PPSN V. Lecture Notes in Computer Science. — Berlin, Heidelberg: Springer, 1998. — Vol. 1498. — P. 305–314.

[81] Dempe S., Zemkoho A. On the Karush–Kuhn–Tucker reformulation of the bilevel optimization problem // Nonlinear Analysis: Theory, Methods and Applications. — 2012. — Vol. 75, No. 3. — P. 1202–1218.

[82] Dempster A.H., Lenstra J.K., Rinnooy Kan A.H.G. Deterministic and stochastic scheduling // Proceedings of the NATO Advanced Study Institute Series. — Dordrecht: Springer, 1982. — Vol. 84. — P. 181–196.

[83] Ding J., Lu Z., Cheng T.C.E., Xu L. A hybrid evolutionary approach for the single machine total weighted tardiness problem // Computers Industrial Engineering. — 2017. — Vol. 108. — P. 70–80.

[84] Doerr B., Doerr C., Ebel F. From black-box complexity to designing new genetic algorithms // Theoretical Computer Science. — 2015. — Vol. 567. — P. 87–104. — DOI: 10.1016/j.tcs.2014.11.028.

[85] Dolgui A., Eremeev A., Guschinskaya O. MIP-based GRASP and genetic algorithm for balancing transfer lines // Matheuristics. Annals of Information Systems. — 2009. — Vol. 10. — P. 189–208.

[86] Dongarra J.J. Performance of various computers using standard linear equations software // Technical Report CS-89-85. — Manchester: University of Manchester, 2014. — 110 p.

[87] Drozdowski M. On complexity of multiprocessor tasks scheduling // Bulletin of the Polish Academy of Sciences. Technical Sciences. — 1995. — Vol. 43, No. 3. — P. 381–392.

[88] Drozdowski M., Kubiak W. Scheduling parallel tasks with sequential heads and tails // Annals of Operations Research. — 1999. — Vol. 90. — P. 221–246.

[89] Drozdowski M., Dell'Olmo P. Scheduling multiprocessor tasks for mean flow time criterion // Computers and Operations Research. — 2000. — Vol. 27, No. 6. — P. 571–585.

[90] Drozdowski M. Scheduling for Parallel Processing. — London: Springer, 2009.

[91] Du J., Leung J.T. Complexity of scheduling parallel task systems // SIAM Journal on Discrete Mathematics. — 1989. — Vol. 2, No. 4. — P. 472–478.

[92] Edmonds J. Maximum matching and a polyhedron with 0,1-vertices // Journal of Research of the National Bureau of Standards. — 1965. — Vol. 69B. — P. 125–130.

[93] Eppstein D. The traveling salesman problem for cubic graphs // Journal of Graph Algorithms and Applications. — 2007. — Vol. 11, No. 1. — P. 61–81.

[94] Erel E., Ghosh J.B. Customer order scheduling on a single machine with family setup times: complexity and algorithms // Applied Mathematics and Computation. — 2007. — Vol. 185, No. 1. — P. 11–18.

[95] Eremeev A.V. A genetic algorithm with a non-binary representation for the set covering problem // Operations Research Proceedings 1998: Selected Papers of the International Conference on Operations Research Zurich, August 31 – September 3, 1998. — Berlin, Heidelberg: Springer, 1999. — P. 175–181.

[96] Eremeev A. On complexity of optimal recombination for binary representations of solutions // Evolutionary Computation. — 2008. — Vol. 16, No. 1. — P. 127–147.

[97] Eremeev A. On complexity of the optimal recombination for the travelling salesman problem // Evolutionary Computation in Combinatorial Optimization. EvoCOP 2011. Lecture Notes in Computer Science. — Berlin, Heidelberg: Springer, 2011. — Vol. 6622. — P. 215–225.

[98] Eremeev A.V., Kovalenko J.V. Optimal recombination in genetic algorithms for combinatorial optimization problems: Part II // Yugoslav Journal of Operations Research. — 2014. — Vol. 24, No. 2. — P. 165–186. — DOI: 10.2298/YJOR131030041E.

[99] Eremeev A.V., Kovalenko J.V. Experimental evaluation of two approaches to optimal recombination for permutation problems // Evolutionary Computation in Combinatorial Optimization. EvoCOP 2016. Lecture Notes in Computer Science. — Cham: Springer, 2016. — Vol. 9595. — P. 138–153. — DOI: 10.1007/978-3-319-30698-8_10.

[100] Eremeev A., Kovalenko Yu. On solving travelling salesman problem with vertex requisitions // Yugoslav Journal of Operations Research. — 2017. — Vol. 27, No. 4. — P. 415–426. — DOI: 10.1134/S1990478916020071.

[101] Eremeev A.V., Kovalenko Y.V. Genetic algorithm with optimal recombination for the asymmetric travelling salesman problem // Large-Scale Scientific Computing. LSSC 2017. Lecture Notes in Computer Science. — Cham: Springer, 2017. — Vol. 10665. — P. 341–349. — DOI: 10.1007/978-3-319-73441-5_36.

[102] Eremeev A.V. A restarting rule based on the Schnabel census for genetic algorithms // Learning and Intelligent Optimization. LION 2019. Lecture Notes in Computer Science. — Cham: Springer, 2019. — Vol. 11353. — P. 337–351.

[103] Eremeev A.V., Kovalenko Y.V. A memetic algorithm with optimal recombination for the asymmetric travelling salesman problem // Memetic Computing. — 2020. — Vol. 12, No. 1. — P. 23–36. — DOI: 10.1007/s12293-019-00291-4.

[104] Ernst A.T., Jiang H., Krishnamoorthy M., Sier D. Staff scheduling and rostering: a review of applications, methods and models // European Journal of Operational Research. — 2004. — Vol. 153, No. 1. — P. 3–27.

[105] Fisher M. Vehicle routing // Network Routing. Handbooks in Operations Research and Management Science. — Amsterdam: Elsevier, North Holland, 1995. — Vol. 8. — P. 1–33.

[106] Framinan J.M., Perez-Gonzalez P. New approximate algorithms for the customer order scheduling problem with total completion time objective // Computers & Operations Research. — 2017. — Vol. 78. — P. 181–192.

[107] Freisleben B., Merz P. A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems // Proceedings of IEEE International Conference on Evolutionary Computation. — IEEE Press, 1996. — P. 616–621.

[108] Fu Q., Chung T.P. On hybrid evolutionary algorithms for scheduling problem with tardiness criterion // Proceedings of 2016 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM). — IEEE, 2016. — Vol. 16560230.

[109] Gao M., He K., Li L., Wang Q., Liu C. A review on energy consumption, energy efficiency and energy saving of metal forming processes from different hierarchies // Processes. — 2019. — Vol. 7, No. 6. — P. 357–381. — DOI: 10.3390/pr7060357.

[110] Gao K., Huang Y., Sadollah A., Wang L. A review of energy-efficient scheduling in intelligent production systems // Complex and Intelligent Systems. — 2020. — Vol. 6. — P. 237–249.

[111] Garey M., Johnson D. Computers and Intractability: A Guide to the Theory of NP-Completeness. — San Francisco, CA: W.H. Freeman and Company, 1979. — 339 p.

[112] Geiger M.J. On heuristic search for the single machine total weighted tardiness problem – some theoretical insights and their empirical verification // European Journal of Operational Research. — 2010. — Vol. 207. — P. 1235–1243.

[113] Gen M., Lin L. Genetic Algorithms and Their Applications // Springer Handbook of Engineering Statistics. — London: Springer, 2023.

[114] Gendreau M., Potvin J.-Y. Handbook of Metaheuristics. — New York: Springer, 2010.

[115] Gerards M.E.T., Hurink J.L., Holzenspies P.K.F. A survey of offline algorithms for energy minimization under deadline constraints // Journal of Scheduling. — 2016. — Vol. 19. — P. 3–19. — DOI: 10.1007/s10951-015-0463-8.

[116] Giaro K., Kubale M., Maiafiejski M., Piwakowski K. Chromatic scheduling of dedicated 2-processor UET tasks to minimize mean flow time // Proceedings of the 7th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA-99). — 1999. — P. 343–347.

[117] Glasgow J., Shachnai H. Channel based scheduling of parallelizable tasks // Proceedings of the 5th IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS-97). — Los Alamitos, CA: IEEE Computer Society, 1997. — P. 11–16.

[118] Goldberg D., Thierens D. Elitist recombination: an integrated selection recombination GA // Proceedings of the First IEEE World Congress on Computational Intelligence. — Piscataway, NJ: IEEE Service Center, 1994. — Vol. 1. — P. 508–512.

[119] Gonzalez T., Sahni S. Open shop scheduling to minimize finish time // Journal of the ACM. — 1976. — Vol. 23, Iss. 4. — P. 665–679. — DOI: 10.1145/321978.321985.

[120] Graham R.L. Bounds for certain multiprocessor anomalies // SIAM Journal on Applied Mathematics. — 1966. — Vol. 17, No. 2. — P. 416–429.

[121] Graham R.L., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G. Optimization and approximation in deterministic sequencing and scheduling: a survey // Annals of Discrete Mathematics. — Amsterdam: Elsevier, 1979. — Vol. 5. — P. 287–326.

[122] Greiner G., Nonner T., Souza A. The bell is ringing in speed-scaled multiprocessor scheduling // Theory of Computing Systems. — 2014. — Vol. 54, No. 1. — P. 24–44.

[123] Grötschel M., Lovász L., Schrijver A. Geometric Algorithms and Combinatorial Optimization. — Berlin: Springer, 1993. — 362 p. — DOI: 10.1007/978-3-642-78240-4.

[124] Gupta A., Krishnaswamy R., Pruhs K. Scalably scheduling power-heterogeneous processors // Proceedings of the International Colloquium on Automata, Languages, and Programming. — 2010. — P. 312–323.

[125] Gupta A., Krishnaswamy R., Pruhs K. Nonclairvoyantly scheduling power-heterogeneous processors // Proceedings of the International Green Computing Conference. — 2010. — P. 165–173.

[126] Gupta A., Im S., Krishnaswamy R., Moseley B., Pruhs K. Scheduling heterogeneous processors isn't as easy as you think // Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms. — 2012. — P. 1242–1253.

[127] Hanafi S., Todosijevic R. Mathematical programming based heuristics for the 0–1 MIP: a survey // Journal of Heuristics. — 2017. — Vol. 23, No. 4. — P. 165–206.

[128] Hazır O., Gunalay Y., Erel E. Customer order scheduling problem: a comparative metaheuristics study // International Journal of Advanced Manufacturing Technology. — 2008. — Vol. 37. — P. 589–598.

[129] He L.J., Cao Y.L., Li W.F., Cao J.J., Zhong L.C. Optimization of energy-efficient open shop scheduling with an adaptive multi-objective differential evolution algorithm // Applied Soft Computing. — 2022. — Vol. 118. — 19 p. — DOI: 10.1016/j.asoc.2022.108459.

[130] Holland J.H. Adaptation in Natural and Artificial Systems. — Ann Arbor, MI: University of Michigan Press, 1975.

[131] Holyer I. The NP-completeness of edge-coloring // SIAM Journal on Computing. — 1981. — Vol. 10. — P. 718–720.

[132] Hoogeveen J., van de Velde S., Veltman B. Complexity of scheduling multiprocessor tasks with prespecified processor allocations // Discrete Applied Mathematics. — 1994. — Vol. 55, No. 3. — P. 259–272.

[133] Huang W., Wang Y. An optimal speed control scheme supported by media servers for low-power multimedia applications // Multimedia Systems. — 2009. — Vol. 15, No. 2. — P. 113–124.

[134] Jansen K., Porkolab L. Preemptive parallel task scheduling in O(n) + Poly(m) time // Algorithms and Computation. ISAAC 2000. Lecture Notes in Computer Science. — Berlin, Heidelberg: Springer, 2000. — Vol. 1969. — P. 398–409.

[135] Jansen K., Porkolab L. Linear-time approximation schemes for scheduling malleable parallel tasks // Algorithmica. — 2002. — Vol. 32, No. 3. — P. 507–520.

[136] Jansen K., Porkolab L. Preemptive scheduling with dedicated processors: applications of fractional graph coloring // Journal of Scheduling. — 2004. — Vol. 7. — P. 35–48.

[137] Johannes B. Scheduling parallel jobs to minimize the makespan // Journal of Scheduling. — 2006. — Vol. 9. — P. 433–452.

[138] Johnson D.S., McGeoch L.A. The traveling salesman problem: a case study // Local Search in Combinatorial Optimization. — New York: John Wiley & Sons Ltd., 1997. — P. 215–336.

[139] Jurisch B., Kubiak W. Two-machine open shops with renewable resources // Operations Research. — 1997. — Vol. 45, Iss. 4. — P. 544–552. — DOI: 10.1287/opre.45.4.544.

[140] Kanellakis P.C., Papadimitriou C.H. Local search for the asymmetric traveling salesman problem // Operations Research. — 1980. — Vol. 28. — P. 1086–1099.

[141] Karzanov A. Determining the maximal flow in a network by the method of preflows // Soviet Mathematics — Doklady. — 1974. — Vol. 15, No. 2. — P. 434–437.

[142] Katal A., Dahiya S., Choudhury T. Energy efficiency in cloud computing data centers: a survey on software technologies // Cluster Computing. — 2023. — Vol. 26, No. 3. — P. 1845–1875.

[143] Kellegoz T., Toklu B., Wilson J. Comparing efficiencies of genetic crossover operators for one machine total weighted tardiness problem // Applied Mathematics and Computation. — 2008. — Vol. 199, No. 2. — P. 590–598.

[144] Khachay M.Yu., Neznakhina E.D., Ryzhenko K.V. Constant-factor approximation algorithms for a series of combinatorial routing problems

based on the reduction to the asymmetric traveling salesman problem // Proceedings of the Steklov Institute of Mathematics (Supplement). — 2022. — Vol. 319, Suppl. 1. — P. S140–S155.

[145] Khramova A.P., Chernykh I. A new algorithm for the two-machine open shop and the polynomial solvability of a scheduling problem with routing // Journal of Scheduling. — 2021. — Vol. 24, Iss. 4. — P. 405–412. — DOI: 10.1007/s10951-021-00694-7.

[146] Kochetov Y.A., Panin A.A., Plyasunov A.V. Genetic local search for the servers load balancing problem // CEUR Workshop Proceedings. – 2016. – Vol. 1623. – P. 453-463.

[147] F Kong F., Guan N., Deng Q., Yi W. Energy-efficient scheduling for parallel real-time tasks based on level-packing // Proceedings of the 2011 ACM Symposium on Applied Computing. — 2011. — P. 635–640. — DOI: 10.1145/1982185.1982326.

[148] Kononov A., Kovalenko Y. On speed scaling scheduling of parallel jobs with preemption // Discrete Optimization and Operations Research. DOOR 2016. Lecture Notes in Computer Science. — Cham: Springer, 2016. — Vol. 9869. — P. 309–321. — DOI: 10.1007/978-3-319-44914-2_25.

[149] Kononov A., Kovalenko Y. An approximation algorithm for preemptive speed scaling scheduling of parallel jobs with migration // Learning and Intelligent Optimization. LION 2017. Lecture Notes in Computer Science. — Cham: Springer, 2017. — Vol. 10556. — P. 351–357. — DOI: 10.1007/978-3-319-69404-7_30.

[150] Kononov A., Kovalenko Y. Approximate schedules for non-migratory parallel jobs in speed-scaled multiprocessor systems // Siberian Electronic Mathematical Reports. — 2019. — Vol. 16. — P. 249–257. — DOI: 10.33048/semi.2019.16.016.

[151] Kononov A., Kovalenko Y. Makespan minimization for parallel jobs with energy constraint // Mathematical Optimization Theory and Operations Research. MOTOR 2020. Lecture Notes in Computer Science.

— Cham: Springer, 2020. — Vol. 12095. — P. 289–300. — DOI: 10.1007/978-3-030-49988-4_20.

[152] Kononov A., Kovalenko Y. Approximation algorithms for energy-efficient scheduling of parallel jobs // Journal of Scheduling. — 2020. — Vol. 23. — P. 693–709. — DOI: 10.1007/s10951-020-00653-8.

[153] Kononov A., Kovalenko Y. Minimizing total completion time in multiprocessor job systems with energy constraint // Mathematical Optimization Theory and Operations Research. MOTOR 2021. Lecture Notes in Computer Science. — Cham: Springer, 2021. — Vol. 12755. — P. 267–279. — DOI: 10.1007/978-3-030-77876-7_18.

[154] Kononov A., Zakharova Y. Speed scaling scheduling of multiprocessor jobs with energy constraint and makespan criterion // Journal of Global Optimization. — 2022. — Vol. 83, No. 3. — P. 539–564. — DOI: 10.1007/s10898-021-01115-x.

[155] Kononov A., Zakharova Y. Minimizing makespan for parallelizable jobs with energy constraint // Siberian Electronic Mathematical Reports. — 2022. — Vol. 19, No. 2. — P. 586–600. — DOI: 10.33048/semi.2022.19.049.

[156] Kononov A., Zakharova Y. Speed scaling scheduling of multiprocessor jobs with energy constraint and total completion time criterion // International Journal of Artificial Intelligence. — 2023. — Vol. 21, No. 2. — P. 109–129.

[157] Kononova P.A., Kochetov Y.A. A local search algorithm for the single machine scheduling problem with setups and a storage // Journal of Applied and Industrial Mathematics. — 2019. — Vol. 13. — P. 239–249.

[158] Kovalenko Y.V., Zakharov A.O. The Pareto set reduction in bicriteria customer order scheduling on a single machine with setup times // Journal of Physics: Conference Series. — 2020. — Vol. 1546. — 8 p.

[159] Kovalyov M.Y., Werner F. Problem $F2||C_{\max}$ with forbidden jobs in the first or last position is easy // European Journal of Operational Research. — 2007. — Vol. 177, No. 2. — P. 1310–1311.

[160] Kozlova A., Nikolaev A. Simulated annealing approach to verify vertex adjacencies in the traveling salesperson polytope // Mathematical Optimization Theory and Operations Research / ed. by Khachay M., Kochetov Y., Pardalos P. — Cham : Springer, 2019. — P. 374–389. — (Lecture Notes in Computer Science ; vol. 11548). — DOI: 10.1007/978-3-030-22629-9_26.

[161] Krishnamurti R., Narahari B. An approximation algorithm for preemptive scheduling on parallel-task systems // SIAM Journal on Discrete Mathematics. — 1995. — Vol. 8, No. 4. — P. 669.

[162] Ku W.Y., Beck J.C. Mixed integer programming models for job shop scheduling: a computational analysis // Computers & Operations Research. — 2016. — Vol. 73. — P. 165–173.

[163] Kubale M. The complexity of scheduling independent two-processor tasks on dedicated processors // Information Processing Letters. — 1987. — Vol. 24, No. 3. — P. 141–147.

[164] Kubale M. Preemptive versus nonpreemptive scheduling of biprocessor tasks on dedicated processors // European Journal of Operational Research. — 1996. — Vol. 94, No. 2. — P. 242–251.

[165] Kubiak W. A Book of Open Shop Scheduling. — Cham: Springer International Publishing, 2022. — 290 p. — DOI: 10.1007/978-3-030-91025-9.

[166] Kuhn H., Tucker A. Nonlinear programming // Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability. — Berkeley, CA: University of California Press, 1951. — P. 481–492.

[167] Kunz K.S. Numerical Analysis. — New York: McGraw-Hill, 1957. — 381 p.

[168] Kwon W.C., Kim T. Optimal voltage allocation techniques for dynamically variable voltage processors // ACM Transactions on Embedded Computing Systems. — 2005. — Vol. 4, No. 1. — P. 211–230.

[169] Lee C.-Y., Cai X. Scheduling one and two-processor tasks on two parallel processors // IIE Transactions. — 1999. — Vol. 31, No. 5. — P. 445–455.

[170] Lenstra J.K., Rinnooy Kan A.H.G., Brucker P. Complexity of machine scheduling problems // Annals of Discrete Mathematics. — Amsterdam: Elsevier, 1977. — Vol. 1. — P. 343–362.

[171] Lepre R., Trystram D., Woeginger G. Approximation algorithms for scheduling malleable tasks under precedence constraints // International Journal of Foundations of Computer Science. — 2002. — Vol. 13, No. 4. — P. 613–627.

[172] Li K. Analysis of the list scheduling algorithm for precedence constrained parallel tasks // Journal of Combinatorial Optimization. — 1999. — Vol. 3. — P. 73–88.

[173] Li K. Energy efficient scheduling of parallel tasks on multiprocessor computers // Journal of Supercomputing. — 2012. — Vol. 60. — P. 223–247. — DOI: 10.1007/s11227-010-0416-0.

[174] Li K. Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers // IEEE Transactions on Computers. — 2012. — Vol. 61, No. 12. — P. 1668–1681.

[175] Li D., Wu J. Energy-Aware Scheduling on Multiprocessor Platforms. — New York: Springer Science and Business Media, 2012. — 59 p. — DOI: 10.1007/978-1-4614-5224-9.

[176] Li M., Yao F., Yuan H. An $O(n^2)$ algorithm for computing optimal continuous voltage schedules // arXiv:1408.5995. — 2014.

[177] Lin B.M.T., Kononov A.V. Customer order scheduling to minimize the number of late jobs // European Journal of Operational Research. — 2007. — Vol. 183, No. 2. — P. 944–948.

[178] Lodi A. The heuristic (dark) side of MIP solvers // Hybrid Metaheuristics. Studies in Computational Intelligence. — Berlin, Heidelberg: Springer, 2013. — Vol. 434. — P. 273–284.

[179] López-Ibáñez M., Dubois-Lacoste J., Cáceres L.P., Birattari M., Stützle T. The irace package: iterated racing for automatic algorithm configuration // Operations Research Perspectives. — 2016. — Vol. 3. — P. 43–58.

[180] Lourenço H.R., Martin O.C., Stützle T. Iterated local search: framework and applications // Handbook of Metaheuristics. — Cham: Springer International Publishing, 2018. — P. 129–168.

[181] Luna F., Blum C., Alba E., Nebro A. J. ACO vs EAs for solving a real-world frequency assignment problem in GSM networks // Proceedings of the 9th annual conference on Genetic and evolutionary computation. – 2007. – P. 94–101.

[182] Makarychev K., Sviridenko M. Solving optimization problems with diseconomies of scale via decoupling // Journal of the ACM. — 2018. — Vol. 65, No. 6. — P. 1–27.

[183] Mansouri S.A., Aktas E., Besikci U. Green scheduling of a two-machine flow-shop: trade-off between makespan and energy consumption // European Journal of Operational Research. — 2016. — Vol. 248, No. 3. — P. 772–788. — DOI: 10.1016/j.ejor.2015.08.064.

[184] McNaughton R. Scheduling with deadlines and loss functions // Management Science. — 1959. — Vol. 6, No. 1. — P. 1–12.

[185] Mitchell M. An Introduction to Genetic Algorithms. — Cambridge, MA: MIT Press, 1998. — 226 p.

[186] Hansen P., Mladenovic N., Moreno Perez, J.A. Variable neighbourhood search: methods and applications // Ann Oper Res. – 2010. – Vol. 175. P. 367–407.

[187] Mood A.M., Graybill F.A., Boes D.C. Introduction to the Theory of Statistics. — New York: McGraw-Hill, 1974.

[188] Mouzon G., Yildirim M.B., Twomey J. Operational methods for minimization of energy consumption of manufacturing equipment // International Journal of Production Research. — 2007. — Vol. 45, Iss. 18–19. — DOI: 10.1080/00207540701450013.

[189] Munkres J. Algorithms for the assignment and transportation problems // Journal of the Society for Industrial and Applied Mathematics. — 1957. — Vol. 5, No. 1. — P. 32–38.

[190] Nagata Y., Soler D. A new genetic algorithm for the asymmetric TSP // Expert Systems with Applications. — 2012. — Vol. 39, No. 10. — P. 8947–8953.

[191] Naroska E., Schwiegelshohn U. On an on-line scheduling problem for parallel jobs // Information Processing Letters. — 2002. — Vol. 81. — P. 297–304.

[192] Nearchou A. A hybrid metaheuristic for the single machine total weighted tardiness problem // Cybernetics and Systems. — 2012. — P. 651–668.

[193] Nesterov Y. Lectures on Convex Optimization. — Cham: Springer, 2018. — 603 p. — DOI: 10.1007/978-3-319-91578-4.

[194] Neri F., Cotta C. Memetic algorithms and memetic computing optimization: a literature review // Swarm and Evolutionary Computation. — 2012. — Vol. 2. — P. 1–14.

[195] Neri F., Cotta C., Moscato P. Handbook of Memetic Algorithms. — Berlin, Heidelberg: Springer-Verlag, 2012.

[196] Neri F., Toivanen J., Cascella G.L., Ong Y.S. An adaptive multimeme algorithm for designing HIV multidrug therapies // IEEE/ACM Transactions on Computational Biology and Bioinformatics. — 2007. — Vol. 4. — P. 264–278.

[197] Norman M., Moscato P. A competitive and cooperative approach to complex combinatorial search // Proceedings of the 20th Joint Conference on Informatics and Operations Research. — Buenos Aires, Argentina, 1991. — P. 3.15–3.29.

[198] Oliver I., Smith D., Holland J. A study of permutation crossover operators on the traveling salesman problem // Proceedings of the Second

International Conference on Genetic Algorithms and their Applications. — Mahwah, NJ: L.E. Associates, Inc., 1987. — P. 224–230.

[199] Prata B. de Ath., Rodrigues C.D., Framinan J.M. Customer order scheduling problem to minimize makespan with sequence-dependent setup times // Computers & Industrial Engineering. — 2021. — Vol. 151.

[200] Pruhs K., Sgall J., Torng E. Online scheduling // Handbook on Scheduling. — Boca Raton, FL: CRC Press, 2004.

[201] Pruhs K., van Stee R. Speed scaling of tasks with precedence constraints // Theory of Computing Systems. — 2007. — Vol. 43. — P. 67–80.

[202] Pruhs K., Uthaisombut P., Woeginger G. Getting the best response for your erg // ACM Transactions on Algorithms. — 2008. — Vol. 4, No. 3. — P. 1–17.

[203] Pyatkin A.V. On list incidentor coloring of a multigraph of degree 3 // Journal of Applied and Industrial Mathematics. — 2008. — Vol. 2. — P. 560–565.

[204] Radcliffe N.J. The algebra of genetic algorithms // Annals of Mathematics and Artificial Intelligence. — 1994. — Vol. 10, No. 4. — P. 339–384.

[205] Reeves C.R., Eremeev A.V. Statistical analysis of local search landscapes // Journal of the Operational Research Society. — 2004. — Vol. 55, No. 7. — P. 687–693.

[206] Rego C., Gamboa D., Glover F. Doubly-rooted stem-and-cycle ejection chain algorithm for the asymmetric traveling salesman problem // Networks. — 2016. — Vol. 68, No. 1. — P. 23–33.

[207] Reinelt G. TSPLIB – a traveling salesman problem library // ORSA Journal on Computing. — 1991. — Vol. 3, No. 4. — P. 376–384.

[208] Reingold E.M., Nievergelt J., Deo N. Combinatorial Algorithms: Theory and Practice. — Englewood Cliffs, NJ: Prentice-Hall, 1977.

[209] Rudolph G. Finite Markov chain results in evolutionary computation: a tour d'horizon // Fundamenta Informaticae. — 1998. — Vol. 35, No. 1–4. — P. 67–89.

[210] Saadani N.E.H., Baptiste P., Moalla M. The simple $F2||C_{\max}$ with forbidden tasks in first or last position: a problem more complex than it seems // European Journal of Operational Research. — 2005. — Vol. 161. — P. 21–31.

[211] Salido M.A., Escamilla J., Giret A., Barber F. A genetic algorithm for energy-efficiency in job-shop scheduling // The International Journal of Advanced Manufacturing Technology. — 2016. — Vol. 85. — P. 1303–1314. — DOI: 10.1007/s00170-015-7987.

[212] Sanches D., Whitley L.D., Tinós R. Building a better heuristic for the traveling salesman problem: combining edge assembly crossover and partition crossover // Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2017). — Berlin; New York, NY: Association for Computing Machinery, 2017. — P. 329–336.

[213] Sathya Sofia A., GaneshKumar P. Multi-objective task scheduling to minimize energy consumption and makespan of cloud computing using NSGA-II // Journal of Network and Systems Management. — 2018. — Vol. 26. — P. 463–485. — DOI: 10.1007/s10922-017-9425-0.

[214] Schwiegelshohn U., Ludwig W., Wolf J., Turek J., Yu P. Smart SMART bounds for weighted response time scheduling // SIAM Journal on Computing. — 1998. — Vol. 28. — P. 237–253.

[215] Shabtay D., Kaspi M. Minimizing the makespan in open-shop scheduling problems with a convex resource consumption function // Naval Research Logistics. — 2006. — Vol. 53, No. 3. — P. 204–216. — DOI: 10.1002/nav.20133.

[216] Shabtay D., Kaspi M. Parallel machine scheduling with a convex resource consumption function // European Journal of Operational Research. — 2006. — Vol. 173, No. 1. — P. 92–107.

[217] Sharma N., Sharma H., Sharma A. An effective solution for large scale single machine total weighted tardiness problem using lunar cycle inspired artificial bee colony algorithm // IEEE/ACM Transactions on Computational Biology and Bioinformatics. — 2019. — Vol. 17, No. 5. — P. 1573–1581.

[218] Shi Zh., Ma H., Ren M. A learning-based two-stage optimization method for customer order scheduling // Computers and Operations Research. — 2021. — Vol. 136, Art. 105488.

[219] Shi Zh., Wang L., Liu P. Minimizing completion time for order scheduling: formulation and heuristic algorithm // IEEE Transactions on Automation Science and Engineering. — 2017. — Vol. 14, No. 4. — P. 1558–1569.

[220] Shioura A., Shakhlevich N.V., Strusevich V.A. Machine speed scaling by adapting methods for convex optimization with submodular constraints // INFORMS Journal on Computing. — 2017. — Vol. 29, No. 4. — P. 724–736.

[221] Shmoys D.B., Stein C., Wein J. Improved approximation algorithms for shop scheduling problems // SIAM Journal on Computing. — 1994. — Vol. 23, Iss. 3. — P. 617–632. — DOI: 10.1137/S009753979222676X.

[222] Soper A.J. A cyclical search for the two machine flow shop and open shop to minimise finishing time // Journal of Scheduling. — Vol. 18, Iss. 3. — P. 311–314. — DOI: 10.1007/s10951-013-0356-7.

[223] Sotskov Y.N., Shakhlevich N.V. NP-hardness of shop-scheduling problems with three jobs // Discrete Applied Mathematics. — 1995. — Vol. 59, No. 3. — P. 237–266.

[224] Stefanini L., Arana-Jiménez M. Karush–Kuhn–Tucker conditions for interval and fuzzy optimization in several variables under total and directional generalized differentiability // Fuzzy Sets and Systems. — 2019. — Vol. 362. — P. 1–34.

[225] Stewart R., Raith A., Sinnen O. Optimising makespan and energy consumption in task scheduling for parallel systems // Computers and Operations Research. — 2023. — Vol. 154. — DOI: 10.1016/j.cor.2023.106212.

[226] Tasgetiren M., Liang Y.C., Sevkli M., Gencyilmaz G. Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem // International Journal of Production Research. — 2006. — Vol. 44, No. 22. — P. 4737–4754.

[227] Tasgetiren M., Sevkli M., Liang Y.C., Gencyilmaz G. Particle swarm optimization algorithm for single machine total weighted tardiness problem // Proceedings of the 2004 Congress on Evolutionary Computation. — IEEE, 2004. — Vol. 2. — P. 1412–1419.

[228] Tinós R., Whitley D., Ochoa G. Generalized asymmetric partition crossover (GAPX) for the asymmetric TSP // Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. — New York, NY: ACM, 2014. — P. 501–508.

[229] Tinos R., Whitley D., Ochoa G. A new generalized partition crossover for the traveling salesman problem: tunneling between local optima // Evolutionary Computation. — 2020. — Vol. 28, No. 2. — P. 255–288.

[230] Turek J., Ludwig W., Wolf J., Fleischer L., Tiwari P., Glasgow J., Schwiegelshohn U., Yu P. Scheduling parallelizable tasks to minimize average response time // Proceedings of the Sixth Annual ACM Symposium on Parallel Algorithms and Architectures. — 1994. — P. 200–209.

[231] Turkensteen M., Ghosh D., Goldengorin B., Sierksma G. Tolerance-based branch and bound algorithms for the ATSP // European Journal of Operational Research. — 2008. — Vol. 189. — P. 775–788.

[232] Uno T. Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs // Algorithms and Computation: 8th International Symposium, ISAAC'97 Singapore. — Berlin, Heidelberg: Springer, 1997. — P. 92–101.

[233] Vakhnin A., Sopov E. A novel self-adaptive cooperative coevolution algorithm for solving continuous large-scale global optimization problems // Algorithms. — 2022. — Vol. 15, No. 12.

[234] Varadarajan S., Whitley D. The massively parallel mixing genetic algorithm for the traveling salesman problem // Proceedings of GECCO-19. — New York, NY: ACM, 2019. — P. 872–879.

[235] Wagner B.J., Davis D.J., Kher H.V. The production of several items in a single facility with linearly changing demand rates // Decision Sciences. — 2002. — Vol. 33, No. 3. — P. 317–346.

[236] Wang Q., Cheng K.-H. List scheduling of parallel tasks // Information Processing Letters. — 1991. — Vol. 37, No. 5. — P. 291–297.

[237] Wang Q., Cheng K.-H. A heuristic of scheduling parallel tasks and its analysis // SIAM Journal on Computing. — 1992. — Vol. 21, No. 2. — P. 281–294.

[238] Wang G., Cheng T.C.E. Customer order scheduling to minimize total weighted completion time // Omega. — 2007. - Vol. 35, No. 5. — P. 623–626.

[239] Wang X., Tang L. A population-based variable neighborhood search for the single machine total weighted tardiness problem // Computers and Operations Research. — 2009. — P. 2105–2110.

[240] de Werra D. Graph-theoretical models for preemptive scheduling // Advances in Project Scheduling. — 1989. — P. 171–185.

[241] Whitley D., Starkweather T., Shaner D. The traveling salesman and sequence scheduling: quality solutions using genetic edge recombination // Handbook of Genetic Algorithms. — New York, NY: Van Nostrand Reinhold, 1991. — P. 350–372.

[242] Wu Ch.-Ch., Bai D., Zhang X. A robust customer order scheduling problem along with scenario-dependent component processing times and due dates // Journal of Manufacturing Systems. — 2021. — Vol. 58 (A). — P. 291–305.

[243] Xing L.N., Chen Y.W., Yang K.W., Hou F., Shen X.S., Cai H.P. A hybrid approach combining an improved genetic algorithm and optimization strategies for the asymmetric TSP // Engineering Applications of Artificial Intelligence. — 2008. — Vol. 21, No. 8. — P. 1370–1380.

[244] Yagiura M., Ibaraki T. The use of dynamic programming in genetic algorithms for permutation problems // European Journal of Operational Research. — 1996. — Vol. 92. — P. 387–401.

[245] Yahyaoui H., Krichen S., Derbel B., Talbi E.G. A variable neighborhood descent for solving the single machine total weighted tardiness problem // Proceedings of the 5th International Conference on Modeling, Simulation and Applied Optimization (ICMSAO). — IEEE, 2013. — Vol. 13641151.

[246] Yao F., Demers A., Shenker S. A scheduling model for reduced CPU energy // Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science. — IEEE, 1995. — P. 374–382.

[247] Zakharov A.O., Kovalenko Y.V. Pareto-based hybrid algorithms for the bicriteria asymmetric travelling salesman problem // Mathematical Optimization Theory and Operations Research. MOTOR 2019. Lecture Notes in Computer Science. — Cham: Springer, 2019. — Vol. 11548. — P. 358–373. — DOI: 10.1007/978-3-030-22629-9_25.

[248] Zakharova Yu. Hybrid evolutionary algorithm with optimized operators for total weighted tardiness problem // Mathematical Optimization Theory and Operations Research. MOTOR 2023. Lecture Notes in Computer Science. — Cham: Springer, 2023. — Vol. 13930. — P. 224–238. — DOI: 10.1007/978-3-031-35305-5_15.

[249] Zakharova Yu. Computational complexity and exact techniques of scheduling problems with job requisitions // Journal of Scheduling. — 2025. — 18 p. — DOI: 10.1007/s10951-025-00840-5.

[250] Zakharova Yu. Population local search for single processor energy efficient scheduling problem // Numerical Computations: Theory and Algorithms. NUMTA 2023. Revised Selected Papers, Part I. Lecture Notes

in Computer Science. — Cham: Springer, 2025. — Vol. 14476. — P. 400–408. — DOI: 10.1007/978-3-031-81241-5_35.

[251] Zakharova Y.V., Sakhno M.Y. Heuristics with local improvements for two-processor scheduling problem with energy constraint and parallelization // Numerical Computations: Theory and Algorithms. NUMTA 2023. Revised Selected Papers, Part I. Lecture Notes in Computer Science. — Cham: Springer, 2025. — Vol. 14476. — P. 241–256. — DOI: 10.1007/978-3-031-81241-5_17.

[252] Zakharova Yu., Sakhno M. Complexity and heuristic algorithms for speed scaling scheduling of parallel jobs with energy constraint // Journal of Computational and Applied Mathematics. — 2025. — Vol. 457. — DOI: 10.1016/j.cam.2024.116254.

[253] Zakharova Yu., Sakhno M. Evolutionary algorithm with optimized operators for multi-processor scheduling with resources // Siberian Electronic Mathematical Reports. — 2025. — Vol. 22, No. 2. — P. 85–103. — DOI: 10.33048/semi.2025.22.

[254] Zakharova Y.V., Zakharov A.O. Integer programming models and metaheuristics for customer order scheduling // Mathematical Optimization Theory and Operations Research: Recent Trends. MOTOR 2024. Communications in Computer and Information Science. — Cham: Springer, 2024. — Vol. 2239. — P. 276–290. — DOI: 10.1007/978-3-031-73365-9_19.

[255] Zhang W. Depth-first branch-and-bound versus local search: a case study // Proceedings of the 17th National Conference on Artificial Intelligence. — Austin, TX, 2000. — P. 930–935.