РОССИЙСКАЯ АКАДЕМИЯ НАУК
СИБИРСКОЕ ОТДЕЛЕНИЕ
ИНСТИТУТ МАТЕМАТИКИ им. С.Л. СОБОЛЕВА

На правах рукописи
УДК 519.21

Столяр Артем Александрович

# Алгоритмы локального поиска для задачи календарного планирования с ограниченными ресурсами

05.13.18 — математическое моделирование, численные методы и комплексы программ

Диссертация на соискание ученой степени
кандидата физико-математических наук

Научные руководители:
д.ф.-м.н., профессор
В.Л. Береснев
к.ф.-м.н., доцент
Ю.А. Кочетов

Новосибирск — 2004

# ОГЛАВЛЕНИЕ

# Введение

Задачи календарного планирования отражают процесс распределения во времени ограниченного числа ресурсов для выполнения проекта, состоящего из заданного множества взаимосвязанных работ. Это известная NP-трудная задача дискретной оптимизации [5], исследуемая более сорока лет. На сегодняшний день она широко используется в таких областях как строительство, военная промышленность, разработка программного обеспечения и т.д. Кроме того, данная задача представляет интерес с математической точки зрения, поскольку календарное планирование содержит достаточно богатый класс известных сложных задач теории расписаний, а также тесно связано с задачами раскроя и упаковки.

В задаче календарного планирования с ограниченными ресурсами (ЗКПОР) задано множество работ, связанных друг с другом условиями предшествования. Эти условия порождают на множестве работ частичный порядок. Для каждой работы задана длительность ее выполнения и объемы потребляемых ресурсов. Суммарный объем каждого ресурса считается известным в каждый момент времени. Все ресурсы являются возобновляемыми [1], неиспользованные ресурсы пропадают как, например, рабочее время. Требуется найти расписание выполнения работ, минимизирующее общее время выполнения всего комплекса работ и, при этом, удовлетворяющее условиям предшествования и ограничениям по ресурсам.

Известно, что для ЗКПОР маловероятно существование приближенного полиномиального алгоритма с гарантированной оценкой точности порядка $O(n^{1-\varepsilon})$ для любого $\varepsilon > 0$ [76]. Данное обстоятельство вызывает особый интерес к приближенным методам, среди которых следует отметить жадные алгоритмы, алгоритмы локального поиска, а также их вероятностные варианты.

Целью работы является разработка и экспериментальное исследование

приближенных алгоритмов, исследование качества получаемых решений.

Разработан новый вероятностный жадный алгоритм, использующий вспомогательную минимаксную задачу. Алгоритм содержит фазу локального улучшения, в ходе которой применяется процедура, основанная на итеративном переходе от активного расписания к T-позднему и обратно. Разработанный алгоритм является одним из лучших в классе жадных алгоритмов решения задачи календарного планирования с ограниченными ресурсами.

Разработан алгоритм поиска с запретами и чередованием окрестностей. В алгоритме использованы две удачно дополняющие друг друга окрестности, определенные, соответственно, для активных и T-поздних расписаний. Окрестности обладают экспоненциальной мощностью, но в алгоритме просматривается только полиномиальное число их элементов, которые выбираются с помощью решения задачи о многомерном рюкзаке.

Разработан новый эволюционный алгоритм, основанный на стратегии связывающих путей. Алгоритм содержит фазу локального улучшения, в ходе которой получаемые решения подлежат перестройке методом поиска с запретами. При формировании начального набора решений используется разработанный автором вероятностный жадный алгоритм.

Проведены численные эксперименты на примерах из международной библиотеки тестовых задач PSPLib [66]. Результаты показали, что разработанный алгоритм является лучшим среди опубликованных эвристических методов. Для 28 тестовых примеров этой библиотеки алгоритмом были получены новые рекордные значения целевой функции.

Предложенные в диссертационной работе алгоритмы ориентированы на широкий класс прикладных задач, связанных с распределением ресурсов. Результаты могут быть использованы для решения большинства моделей календарного планирования (в том числе неклассических), задач упаковки, раскроя, теории расписаний. Кроме того, разработанные методы могут использоваться в схеме гибридных алгоритмов.

Работа организована следующим образом. Во введении приводится математическая постановка задачи и обзор известных методов ее решения, как точных так и приближенных. Описываются некоторые способы кодирования решений и декодирующие процедуры.

В первой главе приводится численное исследование концентрации локальных оптимумов в задаче календарного планирования с ограниченными ресурсами. Исследуется число различных локальных оптимумов относительно трех рассматриваемых окрестностей в различных классах тестовых примеров. Обсуждается вопрос относительно взаимного расположения локальных оптимумов.

Во второй главе описываются три новые жадные стратегии, используемые в схеме вероятностного жадного алгоритма для решения рассматриваемой задачи. Две из них основаны на решении вспомогательных оптимизационных задач. Обсуждаются процедуры локального улучшения решений, получаемых вероятностными жадными алгоритмами.

Третья глава посвящена алгоритму, сочетающему в себе идеи поиска с запретами и чередования окрестностей. В алгоритме используются две окрестности, определяемые для активных и T-поздних расписаний. Вводится новый способ организации списка запретов.

В четвертой главе обсуждаются эволюционные стратегии. Предлагается алгоритм, который называется стратегией связывающих путей. Обсуждаются вопросы комбинирования подобных стратегий с локальным поиском.

В заключении приводится перечень основных результатов диссертационной работы.

Основные результаты диссертации опубликованы в работах [6, 7, 8, 9, 10, 11, 12, 53, 54, 55, 56, 57] и докладывались на Международной конференции "Дискретный анализ и исследование операций"(г. Новосибирск, 2000, 2002, 2004), на 12-й Международной Байкальской конференции (г. Иркутск, 2001), на Симпозиуме по исследованию операций (г. Дуйсбург, Германия, 2001), на конференции "Математическое программирование и приложения"(г. Екатеринбург, 2003), на Международной объединенной конференции по исследованию операций EURO/INFORMS (г. Стамбул, Турция, 2003), на 9-й международной конференции по задачам календарного планирования PMS'04 (г. Нанси, Франция, 2004), на научных семинарах в Институте математики им. С.Л. Соболева СО РАН.

## 0.1 Математическая постановка задачи

Обозначим через $J = \{1, \ldots, n\} \cup \{0, n+1\}$ множество работ, в котором 0-я и $(n+1)$-я работы являются фиктивными и задают начало и завершение всего проекта. Они имеют нулевую длительность. Для каждой работы $j \in J$ время ее выполнения обозначим через $p_j \geq 0,\ p_0 = p_{n+1} = 0$. Отношения предшествования на $J$ зададим совокупностью множеств $P_j$, содержащих всех непосредственных предшественников работы $j,\ j \in J$. Если $i \in P_j$, то работа $j$ не может начаться до завершения работы $i$. Предполагается, что $0 \in P_j$ и $j \in P_{n+1}$ для всех $j \in \{1, \ldots, n\}$.

Через $K = \{1, \ldots, \mathbf{K}\}$ обозначим множество ресурсов, необходимых для выполнения работ множества $J$. Все ресурсы предполагаются возобновляемыми и нескладируемыми [1], то есть в каждый момент времени выделяется определенный объем ресурсов каждого типа, а остатки ресурсов пропадают. В рассматриваемой постановке задачи выделяемый ресурс $R_k > 0, k \in K$ есть величина постоянная. Через $r_{jk} \geq 0$ обозначим объем $k$-го ресурса, необходимый для выполнения $j$-й работы в каждый момент ее выполнения.

Введем переменные задачи. Через $s_j \geq 0$ обозначим момент начала выполнения $j$-й работы. Так как работы выполняются без прерывания, то момент окончания работы $c_j \geq 0$ определяется равенством $c_j = s_j + p_j$. Через $A(t) = \{j \in J \mid s_j \leq t < c_j\}$ обозначим множество работ, находящихся в процессе выполнения в момент времени $t$. Время завершения проекта для расписания $S = \{s_j,\ j \in J\}$ обозначим через $T(S)$. Эта величина соответствует времени завершения последней работы проекта, $T(S) = c_{n+1}$. С использованием данных обозначений задача календарного планирования с ограниченными ресурсами записывается следующим образом.

Найти :

$$\min T(S)$$

при ограничениях

$$c_i \leq s_j,\ i \in P_j,\ j \in J$$

$$\sum_{j \in A(t)} r_{kj} \leq R_k,\ k \in K,\ t \geq 0,$$

$$s_j \geq 0.$$

Целевая функция задает время завершения проекта. Первое ограничение соответствует условиям предшествования, второе — выделяемым ресурсам.

Сформулированная задача, как обобщение известной задачи Job-Shop, относится к классу NP-трудных задач дискретной оптимизации [25]. Известно, что для задачи календарного планирования с ограниченными ресурсами маловероятно существование приближенного полиномиального алгоритма с гарантированной оценкой точности порядка $n^{1-\varepsilon}$ для любого $\varepsilon > 0$. Более точно это утверждение выглядит следующим образом.

**Теорема 1** *[76] Для ЗКПОР не существует полиномиального приближенного алгоритма с гарантированной оценкой точности порядка $n^{1-\varepsilon}$ для любого $\varepsilon > 0$, если $NP \neq ZPP$.*

Таким образом, применение точных алгоритмов целесообразно лишь в том случае, когда размерность задачи относительно невелика (до 50 работ). В задачах большой размерности применение приближенных методов остается единственным эффективным способом их решения. Ниже приводится обзор наиболее известных методов решения рассматриваемой задачи.

## 0.2 Точные методы

В основе большинства точных алгоритмов лежит метод ветвей и границ. В качестве вершин в дереве поиска рассматриваются частичные расписания. Процесс ветвления, как правило, состоит в достройке частичного расписания с помощью различных стратегий. Использование таких приемов как правила доминирования, нижние оценки, непосредственный отбор [29] позволяет в большинстве случаев сократить дерево поиска. В зависимости от конкретного метода используются различные схемы ветвления и способы отсечения.

### 0.2.1 Дерево предшествований

В работе [85] предлагается алгоритм, основанный на так называемом дереве предшествований. Процедура начинается с присвоения начальной фиктивной работе нулевого момента начала. На каждом уровне $m$ дерева ветвления рассматривается множество работ $J_m$, включенных в расписание, а

также множество допустимых работ $D_m = \{j \mid P_j \subseteq J_m\}$, предшественники которых принадлежат множеству $J_m$. Далее выбирается допустимая работа $j$, для которой вычисляется наиболее ранний момент начала $s_j$ в соответствии с условиями предшествования и ограничениям по ресурсам. Затем процесс ветвления переходит на следующий уровень. Как только в качестве допустимой работы выбирается последняя фиктивная работа $n+1$, получено полное расписание (висячая вершина) и процесс поднимается на уровень выше с переходом к следующей допустимой работе. Как только просмотрены все допустимые работы данного уровня, процесс переходит еще на уровень выше. В результате получается дерево, где всякий путь из корня в висячую вершину представляет собой допустимую по условиям предшествования перестановку работ. В последствии этот алгоритм был модернизирован с помощью эффективной технологии сокращения дерева поиска [92].

### 0.2.2 Выбор с задержкой

Этот алгоритм был предложен в работе [31]. В отличие от предыдущего алгоритма, на каждом уровне $m$ рассматривается момент времени $t_m$ как потенциальный момент начала некоторых работ. Не включенная в расписание работа $j$ является допустимой в момент $t_m$, если все ее предшественники $i \in P_j$ включены в расписание и заканчиваются до текущего момента, т.е. $c_i \leq t_m,\ i \in P_j$. Множество $A(t_m) = \{j \mid s_j \leq t_m < c_j\}$ содержит работы, находящиеся в процессе выполнения в момент времени $t_m$. Момент $t_m$ определяется как минимальный момент завершения работ, выполняемых в момент $t_{m-1}$, т.е. $t_m = \min\{c_j \mid j \in A(t_{m-1})\}$. Далее строится множество допустимых работ $D_m$, которое затем добавляется к множеству выполняемых работ $A(t_m)$. В результате могут быть нарушены ограничения по ресурсам. Следовательно, чтобы сохранить допустимость, какие-то работы необходимо задержать. Через $DA(t_m)$ обозначено подмножество $A(t_m)$, для которого выполняется условие $\sum_{j \in A(t_m) \setminus DA(t_m)} r_{jk} \leq R_k$ для всех $k \in K$. Множество $DA(t_m)$ выбирается минимальным по включению. Ветвление соответствует альтернативному выбору множеств $DA(t_m)$. Как только получено полное расписание, процесс возвращается на уровень выше и переходит к следующей альтернативе.

### 0.2.3 Выбор с расширением

Метод, предложенный в работе [96], отличается от предыдущего выбором альтернативы. По-прежнему рассматриваются множества $D_m$ и $A(t_m)$. Однако в отличие от предыдущего метода, они не объединяются. Вместо этого строится множество $EA(t_m)$ как подмножество допустимых работ, которые могут выполняться параллельно с работами из множества $A(t_m)$, не нарушая ограничений по ресурсам, т.е. $\sum_{j \in A(t_m) \bigcup EA(t_m)} r_{jk} \leq R_k$. Ветвление происходит по различным множествам $EA(t_m)$. Еще одно отличие состоит в том, что при переходе на следующий уровень $m+1$ все работы из множества $A(t_m)$ остаются в расписании, в то время как в предыдущем методе некоторые из них могут быть временно исключены из расписания.

### 0.2.4 Выбор на основе логической схемы

В методе, предложенном в работе [29], вместо частичных расписаний рассматриваются логические схемы, определяемые следующим образом. Схема $(C,\ D,\ N,\ F)$ состоит из четырех отношений на множестве $J$. Отношения в $C$ называются конъюнкциями и обозначаются через $(i \rightarrow j)$. Для таких работ выполнено $c_i \leq s_j$. Отношения в $D$ называются дизъюнкциями, и обозначаются через $(i - j)$. Сюда относятся работы, для которых выполнено либо $(i \rightarrow j)$, либо $(j \rightarrow i)$. Множество $N$ содержит пары работ, которые выполняются параллельно в течение некоторого периода времени. Все остальные пары лежат в $F$.

Если $C_0$ — множество пар, связанных условиями предшествования, $D_0$ — множество пар, которые не могут выполняться одновременно ввиду ограничений по ресурсам, а $F_0$ — множество всех оставшихся пар, то схема $(C_0,\ D_0,\ \emptyset,\ F_0)$ будет выступать в качестве корневой вершины в дереве поиска. Ветвление происходит путем перемещения пары из $F$ в $D$, либо в $N$. С помощью специальных правил [29] удается пополнить множества $C$, $D$, и $N$, сокращая тем самым дерево поиска.

### 0.2.5 Представление в виде задачи ЦЛП

Одним из способов решения ЗКПОР является применение пакета CPLEX [24] к формулировке в виде задачи целочисленного линейного программирова-

ния (ЦЛП) [88]. Для каждой работы $i \in J$ определяется окно ее выполнения $[es_i, ls_i]$. Границы окна обозначают, соответственно, наиболее раннее и наиболее позднее время начала работы $i$. Положив $es_0 = 0$ и $ls_{n+1} = T_{max}$, некоторой верхней оценке на длину расписания, значения $[es_i$ и $ls_i]$ определяются для всех $i \in J$ согласно условиям предшествования по рекуррентным формулам [39]. Расписание определяется значениями булевых переменных $\xi_{it} \in \{0, 1\}$: значение переменной равно единице тогда и только тогда, когда работа $i$ начинается в момент времени $t$, $i \in J$, $t = es_i, \ldots, ls_i$. С учетом введенных обозначений формулировка в задачи ЦЛП записывается следующим образом.

Найти :

$$\min \sum_{t=es_{n+1}}^{ls_{n+1}} t \cdot \xi_{n+1\ t}$$

при ограничениях

$$\sum_{t=es_i}^{ls_i} \xi_{it} = 1,\ i \in J,$$

$$\sum_{t=es_j}^{ls_j} t \cdot \xi_{jt} - \sum_{t=es_i}^{ls_i} t \cdot \xi_{it} \geq p_i,\ i \in P_j,$$

$$\sum_{i \in J} r_{ik} \sum_{\tau=\sigma(t,i)}^{t} \xi_{i\tau} \leq R_k,\ t = 0, \ldots, T_{max},\ k \in K,$$

$$\xi_{it} \in \{0, 1\}, i \in J,\ t = es_i, \ldots, ls_i,$$

где $\sigma(t, i) = \max\{0, t - p_i + 1\}$. Первое ограничение запрещает прерывания. Второе и третье соответствуют условиям предшествования и ограничениям по ресурсам.

## 0.3 Метод Т-поздних расписаний для задачи со складируемыми ресурсами

Ресурс называется *складируемым*, если он, будучи неистраченным в момент времени $t$, может быть использован в момент времени $t' > t$. Если все ресурсы являются складирумыми, удается построить точный полиномиальный алгоритм [1].

**Определение 1** *Допустимое расписание называется $T$-поздним, если для всех работ $j \in J$ и некоторого $T$ выполнено $s_j + p_j \leq T$ и увеличение любого из моментов $s_j$ приводит к нарушению этого неравенства либо условий предшествования.*

Очевидно, что длина $T$-позднего расписания не превосходит $T$. Такое расписание может быть построено с помощью следующих рекуррентных соотношений:

$$s_{n+1} = T,$$

$$s_j = \min\{s_i \mid i \in S_j\} - p_j,$$

где $S_j$ – множество непосредственных последователей работы $j$.

**Теорема 2** *[1] Пусть $T^*$ — длина оптимальноого расписания. Тогда существует допустимое $T^*$-позднее расписание.*

Если $p_j$ — целые числа, то данное утверждение позволяет искать оптимальное расписание среди $T$-поздних методом дихотомии. Пусть $T_1$ и $T_2$ соответственно нижняя и верхняя оценка длины допустимого расписания. Тогда оптимальное расписание может быть получено с помощью следующего алгоритма [2].

1. Положить $[T = (T_1 + T_2)/2]$.
2. Построить T-позднее расписание $\{s_j^T\}$.
3. Если расписание $\{s_j^T\}$ допустимо, то положить $T_1 = T$,
   иначе положить $T_2 = T$.
4. Если $T_2 - T_1 > 1$, то перейти на шаг 1,
   иначе получено оптимальное расписание $\{s_j^{T_1}\}$.

Данный алгоритм применим к задаче с директивными сроками [1]. Переход от возобновляемых ресурсов к складируемым расширяет множество допустимых решений задачи и, следовательно, позволяет находить нижнюю оценку длины оптимального расписания в исходной задаче.

## 0.4 Приближенные методы

### 0.4.1 Декодирующие процедуры

Декодирующие процедуры (ДП) являются основой большинства эвристических алгоритмов для решения задачи календарного планирования с ограниченными ресурсами. Они представляют собой процесс пошагового построения расписания. В зависимости от способа включения в расписание очередной работы, различают последовательную, Т-позднюю и параллельную декодирующую процедуру.

#### Последовательная процедура

Последовательная процедура состоит из $n+2$ шагов. На каждом шаге $m$ рассматривается множество $J_m$ работ, уже включенных в расписание, и остаточный объем имеющихся ресурсов в каждый момент времени $\widetilde{R}_k(t) = R_k - \sum\limits_{j \in A(t)} r_{jk}$. Шаг состоит в добавлении в расписание очередной работы из множества $D_m = \{j \in J \setminus J_m \mid P_j \subseteq J_m\}$, которое будем называть допустимым множеством. Алгоритм построения расписания может быть записан следующим образом.

Последовательный декодер:

0. Положить $c_0 = 0,\ J_0 = \{0\}$.
1. Для всех $m = 1, \ldots, n+1$ выполнять следующее:
   1.1 Вычислить $D_m,\ \widetilde{R}_k(t)$.
   1.2 Выбрать работу $j \in D_m$.
   1.3 Вычислить наиболее ранний момент $s_j$ начала работы $j, \in D_m$, допустимый по условиям предшествования и ограничениям по ресурсам
   1.4 Положить $J_m = J_{m-1} \bigcup \{j\}$.
2. Положить $T(S) = s_{n+1}$.

Трудоемкость процедуры оценивается величиной $O(n^2\mathbf{K})$. Полученное таким образом расписание относится к классу *активных* расписаний [60].

**Определение 2** *[93] Активным называется такое расписание, в котором ни одна работа не может быть начата раньше указанного ей срока без нарушения условий предшествования либо ограничений по ресурсам.*

**Теорема 3** *[60] Класс активных расписаний содержит оптимальное расписание.*

Данное утверждение позволяет ограничить пространство поиска множеством активных расписаний.

### T-поздняя процедура

T-поздняя процедура является аналогом последовательной ДП и строит расписание в обратной последовательности. На первом шаге фиксируется число $T$ и полагается $c_{n+1} = T$. Далее на каждом шаге $m = n, \ldots, 1$ рассматриваются аналогичные множества $J_m$ и $\widetilde{R}_k(t) = R_k - \sum\limits_{j \in A(t)} r_{jk}$. Допустимое множество $D_m = \{j \in J \setminus J_m \mid \quad S_j \subseteq J_m\}$ состоит из работ, последователи которых включены в расписание.

$\underline{T\text{-поздний декодер}(T)}$:

0. Положить $s_{n+1} := c_{n+1} := T$.
1. Для всех $m = n, \ldots, 0$ выполнять следующее:
   1.1 Вычислить $D_m$, $\widetilde{R}_k(t)$.
   1.2 Выбрать работу $j \in D_m$.
   1.3 Вычислить наиболее поздний момент $c_j$ завершения работы $j, \in D_m$, допустимый по условиям предшествования и ограничениям по ресурсам
   1.4 Положить $s_{j_m} = c_j - p_{j_m}$.
   1.5 Положить $J_m = J_{m+1} \bigcup \{j\}$.
3. Вычислить $T(S) = T - s_0$.

Трудоемкость процедуры также оценивается величиной $O(n^2\mathbf{K})$. Если ограничения по ресурсам не зависят от времени, то среди $T$-поздних расписаний найдется и оптимальное. Доказательство проводится аналогично случаю с активными расписаниями.

### Параллельная процедура

Последовательная ДП поочередно рассматривает работы проекта и для каждой из них подбирает момент ее начала. Параллельная процедура поступает иначе. На каждом шаге $m = 1, \ldots, n$ рассматривается момент времени $t_m$, и по нему строится множество работ, которые будут начинаться в указанный момент времени. Обозначим через $J(t_m) = \{j \in J \mid c_j = t_m\}$ множество работ, завершенных к моменту времени $t_m$ и через $D(t_m) = \{j \in J \setminus J(t_m) \mid P_j \subseteq J(t_m),\ r_{jk} \leq \widetilde{R}_k(t_m)\}$ — допустимое множество работ, где $\widetilde{R}_k(t_m)$ — остаточный объем ресурсов в момент $t_m$. С учетом введенных обозначений, параллельная процедура выглядит следующим образом.

Параллельный декодер:

0. Положить $m = 0,\ t_m = 0,\ A(0) = \{0\},\ J(0) = \{0\},\ \widetilde{R}_k(0) = R_k$.
1. Пока $|A(t_m) \bigcup J(t_m)| \leq n$ выполнять следующее:
   1.1 Положить $m = m + 1$.
   1.2 Положить $t_m = \min\{c_j \mid j \in A(t_{m-1})\}$.
   1.3 Вычислить $J(t_m),\ A(t_m),\ \widetilde{R}_k(t_m),\ D(t_m)$.
   1.4 Пока $D(t_m) \neq \emptyset$ выполнять следующее:
      1.4.1 Выбирать работу $j \in D(t_m)$.
      1.4.2 Положить $s_j = t_m$.
      1.4.3 Обновить $\widetilde{R}_k(t_m),\ A(t_m),\ D(t_m)$.
2. Положить $T(S) = \max\{c_j \mid j \in J\}$.

Трудоемкость процедуры также оценивается величиной $O(n^2\mathbf{K})$. Полученное таким образом расписание относится к классу *плотных* расписаний [60], который является подклассом активных расписаний.

**Определение 3** *[93] Активное расписание называется плотным, если при замене каждой работы $j \in J$ на $p_j$ работ единичной длительности оно остается активным.*

На практике параллельная декодирующая процедура способна строить достаточно хорошие расписания. Класс плотных расписаний является подклассом активных расписаний, однако известно, что он может не содержать оптимального решения [60].

### 0.4.2 Кодировки решений

Во многих приближенных алгоритмах удобно использовать кодирование решений. Ниже приводятся основные известные способы такого кодирования.

#### Список

Значительная часть работ, посвященных алгоритмам решения ЗКПОР, использует кодировку решения в виде списка $L = (j_1, \ldots, j_n)$. Предполагается, что списки согласованы с условиями предшествования, то есть для любой пары $(i, j)$, $i \in P_j$ позиция работы $j$ больше позиции работы $i$. По списку строится расписание с помощью вышеописанных декодирующих процедур. Последовательный декодер согласно списку последовательно вычисляет наиболее раннее время начала выполнения каждой работы, учитывая ресурсные ограничения. Для $T$-позднего декодера фиксируется длина расписания $T$, и для каждой работы согласно списку вычисляется наиболее позднее время ее окончания с учетом ресурсных ограничений. Как было сказано выше, условия предшествования учитываются при составлении списка, что позволяет избежать их проверки при декодировании. В схеме параллельного декодера на шаге 1.4.1 из допустимого множества выбирается работа с наименьшей позицией в списке.

#### Вектор значений приоритета

Для решения задач теории расписаний интенсивно исследовались быстрые полиномиальные алгоритмы, основанные на приоритетных правилах. Согласно этому подходу, среди множества работ, доступных к выполнению, выбирается одна работа по заданному критерию, например, работа с минимальной длительностью, минимальным числом предшественников или другому критерию. Понятно, что ни одно из таких правил не гарантирует получения оптимального решения, если задача является $NP$-трудной. Поэтому более эффективной стратегией является применение сразу нескольких приоритетных правил и использование одного из них в зависимости от уже построенного частичного расписания. Идея такой групповой стратегии применялась и для ЗКПОР [26, 99]. Кодировка решений задается

вектором $\pi = (\pi_1, \ldots, \pi_n)$, где $\pi_i$ определяет приоритетное правило выбора очередной работы. Отметим, что такая кодировка, как и предыдущая, позволяет за полиномиальное время получить расписание работ, но одному расписанию может соответствовать несколько разных векторов $\pi$.

### Вектор смещения

В работе [90] было предложено представление решений в виде вектора смещения. Решение задается вектором целых неотрицательных чисел $\sigma = (\sigma_1, \ldots, \sigma_n)$. Декодирующая процедура последовательно вычисляет величины $s_j$ как максимум по всем моментам завершения предшественников работы $j$, к которым добавляются некоторые величины задержки $\sigma_j$, т. е. $s_j = \max\{s_i + p_i \mid i \in P_j\} + \sigma_j,\ j = 1, \ldots, n$. Поскольку декодирующая процедура не учитывает ограничения по ресурсам, то полученное расписание может оказаться недопустимым. В этом случае к длине расписания добавляется величина штрафа, которая зависит от степени нарушения ресурсных ограничений.

### Логическая схема

Для переборных методов типа ветвей и границ была предложена специальная кодировка решений, ориентированная на анализ условий предшествования и ограничений по ресурсам [29]. Эта кодировка получила название *логическая схема*. Она представляет собой четверку непересекающихся отношений $(C,\ D,\ N,\ F)$. Множество $C$ задает исходные условия предшествования. Если $(i, j) \in D$, то работы $i$ и $j$ не могут пересекаться по времени. Если $(i, j) \in N$, то работы $i$ и $j$ должны выполняться параллельно в течение хотя бы одной единицы времени. Множество $F$ содержит все оставшиеся пары $(i, j)$, которые не противоречат множествам $C, D$ и $N$. Конкретная четверка $(C,\ D,\ N,\ F)$ является кодом всех расписаний, в которых выполняются все отношения из $C,\ D$, и $N$. В качестве декодирующей процедуры была разработана эвристика [29], строящая расписание, удовлетворяющее отношениям из $C$ и $D$ и большей части отношений из $N$. Следует отметить, что вопрос о существовании допустимого расписания, удовлетворяющего фиксированной схеме является $NP$-полной задачей [69].

Кроме перечисленных существует и кодировка в виде набора булевых переменных $\xi_{jt} \in \{0, 1\}$, $\xi_{jt} = 1 \Longleftrightarrow s_j = t$. Такая кодировка использовалась в алгоритме лагранжевой релаксации [76] для формулировки ЗКПОР в виде задачи ЦЛП.

### 0.4.3 Методы приоритетных правил

В основе методов этого класса как правило лежит одна из вышеописанных декодирующих процедур. Выбор очередной работы для включения в расписание осуществляется исходя из расстановки приоритетов среди работ допустимого множества $D_m$. Ниже приводится обзор наиболее известных приоритетных правил.

#### Приоритетные правила

Исследованиям в области приоритетных правил для ЗКПОР посвящено колоссальное число работ [18, 26, 34, 33, 35, 36, 40, 58, 59, 60, 70, 80, 82, 83, 84, 91, 97, 99, 102, 106, 108]. Приоритетное правило ставит в соответствие каждой работе $j \in D_m$ некоторое число $v_j$. В ходе построения расписания при выборе очередной работы из допустимого множества определяющим фактором является указанное значение приоритета.

Приоритетные правила обычно классифицируют по трем критериям. По типу информации, необходимой для вычисления значения приоритета правила делятся на сетевые, временные и использующие ограничения по ресурсам [18, 70]. По количеству используемой информации различают глобальные и локальные правила. В локальных правилах необходима лишь информация об одной работе, для которой определяется значение приоритета, например, длительность выполнения, в то время как в глобальных правилах необходим больший объем данных. Деление на статические и динамические правила основывается на том, зависит ли определяемое значение приоритета от предыстории построения текущего расписания или же способ его определения всегда одинаковый. Наконец, правило может предполагать использование как только одного декодера, так и нескольких.

Примеры наиболее известных приоритетных правил приведены в Приложении А: наибольший позиционный вес (GRPW — greatest rank positional

weight), наиболее поздний момент завершения (LFT: latest finish time), наиболее поздний момент начала (LST: latest start time), минимальный резерв (MSLK: minimum slack), максимум всех последователей (MTS: most total successors), метод распределения ресурсов (RSM: resource scheduling method), минимальное время выполнения (SPT: shortest processing time), резерв в худшем случае (WCS: worst case slack). Правило MTS использует множество $\overline{S}_j$ — транзитивное замыкание множества непосредственных последователей работы $j \in J$. Для определения правил WCS и RSM вводится множество пар работ $AP = \{(i,j) \in D_m \times D_m \mid i \neq j\}$ — декартово произведение допустимого множества за вычетом диагонали. Правило WCS использует величину $E(i,j)$ — наиболее ранний момент начала работы $J \in J$, допустимый по условиям предшествования и ограничениям по ресурсам, при условии, что работа $i \in J$ начата в момент времени $t_m$. Это правило предполагает использование только параллельного декодера.

Большинство эвристических алгоритмов, основанных на приоритетных правилах, используют их в декодирующих процедурах в ходе построения расписания. В зависимости о того, сколько расписаний предполагается построить такие эвристики делятся на одношаговые и многошаговые.

### Одношаговые методы

К классу одношаговых методов относится большинство детерминированных жадных алгоритмов. Их основная идея состоит в построении одного расписания, используя последовательную или параллельную декодирующую процедуру в соответствии с заранее определенным правилом приоритета. Примеры таких алгоритмов описаны в работах [18, 26, 33, 34, 35, 36, 40, 58, 60, 70, 83, 84, 97, 106, 108].

Значительно позже были разработаны более сложные приоритетные правила как, например, WCS для параллельного декодера [60]. В работах [82, 102] описывается приоритетное правило, получившее название анализа на основе локального ограничения (LCBA: local constraint based analysis). Правило LCBA предполагает использование параллельного декодера и на основе имеющейся информации о потреблении ресурсов в момент времени $t_m$ принимает решение о том, какие работы допустимого множества $D(t_m)$ необходимо начать в момент $t_m$.

### Многошаговые методы

Основная идея многошаговых эвристик состоит в построении нескольких расписаний с последующим выбором наилучшего из них. При этом очередное расписание может строиться как независимо от построенных ранее, так и опираясь на предысторию. Разнообразие генерируемых расписаний может достигаться использованием нескольких приоритетных правил, нескольких декодирующих процедур, внесением рандомизации и т.п.

**Методы, использующие несколько правил** Для ЗКПОР известен достаточно широкий спектр приоритетных правил. Тем не менее, нельзя утверждать, что одно правило заведомо лучше другого. Как показывают исследования [61, 94, 95], разные правила, используемые в схеме одного алгоритма, обладают переменным успехом в зависимости от специфики исходных данных тестовой задачи. Поэтому многие авторы предлагают использовать несколько правил для получения нескольких расписаний. Так, например, в работе [26] предлагается алгоритм, использующий 7 различных правил. Недостатком такого подхода является то, что число построенных таким образом расписаний не превосходит числа использованных правил. В качестве одного из способов преодолеть этот недостаток предлагается заменить использование $m$ правил для получения $m$ расписаний на их выпуклую комбинацию $v(j) = \sum_{i=1}^{m} w_i \cdot v_i(j)$, где $w_i \geq 0, \forall i$ и $\sum_{i=1}^{m} w_i = 1$, как, например в работах [99, 102]. Этот прием позволяет строить любое наперед заданное число расписаний.

**Методы с прямым и обратным проходом** Методы этого типа состоят из двух этапов. На первом этапе, подобно одношаговым методам, строится расписание с помощью одного декодера и какого-либо правила. Далее фиксируется время завершения проекта и строится новое расписание в обратном порядке. При этом новые значения приоритета обычно определяются по моментам начала или завершения работ в расписании, построенном на предыдущем этапе. Оба этапа могут повторяться многократно, пока не выполнен критерий остановки. Примеры таких алгоритмов можно найти в работах [71, 82].

**Вероятностные многошаговые методы** Данная категория алгоритмов включает большинство известных вероятностных жадных алгоритмов для решения ЗКПОР. Такие алгоритмы, как правило, представляют собой серию из независимых испытаний, в ходе каждого из которых строится расписание с внесением элемента случайности. В качестве результата выступает лучшее расписание из указанной выборки. В отличие от предыдущих методов для работ допустимого множества вместо значений приоритета строится вероятностное распределение $\{pr(j),\ j \in D_m\},\ \sum_{j \in D_m} pr(j) = 1$. На каждом шаге построения расписания очередная работа выбирается случайным образом согласно указанному распределению. Наиболее простым является равновероятный выбор работ, т.е. когда $pr(j) = 1/|D_m|$ для всех $j \in D_m$. В иностранной литературе такой способ известен под названием *Random sampling*. Недостатки такого подхода очевидны. Хотя он и обеспечивает необходимое разнообразие, но тем не менее практически не использует информацию об исходных данных задачи. В работах [19, 34] было предложено использовать распределение на основе значений приоритета. Вероятности $pr(j)$ определялись пропорционально величинам $v(j)$ с целью отдавать предпочтение работам с большими значениями приоритетов. Интересное сочетание приоритетов и разнообразия используется в алгоритме, предложенном в работе [95]. Для определения вероятностей $pr(j)$ авторы предлагают использовать нормальное распределение, равновероятно выбирая работы с минимальным и максимальным значением приоритета.

Согласно исследованиям, проведенным в [59, 95], наиболее часто наилучшими вероятностными многошаговыми методами оказываются эвристики, в которых при определении вероятностей $pr(j),\ j \in D_m$ используется так называемое *распределение на основе худшего* [38]. Для каждой работы $j \in D_m$ определяется разность $r(j) = v(j) - \min\limits_{i \in D_m} v(i)$ между ее значением приоритета и наименьшим значением среди всех работ допустимого множества. Далее, определяются величины $r'(j) = (r(j) + \epsilon)^\alpha$, пропорционально которым вычисляются соответствующие вероятности $pr(j),\ j \in D_m$. Добавление величины $\epsilon > 0$ гарантирует положительную вероятность для каждой работы допустимого множества и, тем самым, с ненулевой вероятностью позволяет построить любое допустимое расписание. Выбор значения параметра $\alpha$ контролирует степень рандомизации. При больших зна-

чениях $\alpha$ поведение алгоритмов подобно детерминированным, поскольку работы с наибольшим значением приоритета будут всегда получать несравнимо бóльшую вероятность выбора. При $\alpha = 0$ достигается наибольший произвол, т.к. в этом случае все работы выбираются равновероятно.

В работе [61] был предложен адаптивный многошаговый метод. За основу были взяты последовательный декодер с правилом LFT и параллельный декодер с правилом WCS. Адаптивность метода обусловлена выбором одного из двух указанных вариантов при каждом построении нового расписания. Этот выбор зависел как от исходных данных задачи, так и от числа уже построенных расписаний. Этот алгоритм был в дальнейшем модифицирован [94] путем динамического определения параметра $\epsilon$ и увеличения числа используемых приоритетных правил.

### 0.4.4 Метаэвристики

По своей сути метаэвристики представляют собой общие схемы построения алгоритмов, которые могут быть реализованы для большинства задач дискретной оптимизации. Все метаэвристики являются итерационными процедурами и для многих из них установлена асимптотическая сходимость наилучшего найденного решения к глобальному оптимуму [2, 13]. К этому классу относятся алгоритмы имитации отжига, поиск с запретами, генетические алгоритмы, муравьиные колонии и другие [49].

**Имитация отжига** Экзотическое название данного алгоритма связано с методами имитационного моделирования в статистической физике, основанными на технике Монте-Карло. Исследование кристаллической решетки и поведения атомов при медленном остывании тела привело к появлению на свет вероятностных алгоритмов, которые оказались чрезвычайно эффективными в комбинаторной оптимизации. Впервые это было замечено в 1983 году [52]. Сегодня этот алгоритм является популярным как среди практиков благодаря своей простоте, гибкости и эффективности, так и среди теоретиков, поскольку для данного алгоритма удается аналитически исследовать его свойства и доказать асимптотическую сходимость.

Алгоритм имитации отжига относится к классу пороговых алгоритмов локального поиска. На каждом шаге этого алгоритма в окрестности те-

кущего решения $S_k$ выбирается некоторое решение $S'$, и если разность по целевой функции между новым и текущим решением не превосходит заданного порога $T_k$, то новое решение $S'$ заменяет текущее. В противном случае выбирается новое соседнее решение.

Алгоритм, предложенный в работе [30], использует кодировку решения в виде вектора значений приоритета. Элемент окрестности определяется по двум работам $i, j \in J$, для которых значения приоритета в соседнем решении меняются друг с другом. При этом работа $i$ выбирается случайно из всего множества работ, а работа $j$ — только из некоторой его части, определяемой работой $i$.

Алгоритм [28] использует кодировку в виде списка. Элемент окрестности определяется для каждой работы $j \in J$. Для указанной работы в текущем списке определяется окно между ближайшим предшественником и последователем. В соседнем списке работа $j$ перемещается на произвольное место внутри выбранного окна. При этом работы, заключенные между старой и новой позициями работы $j$ перемещаются циклически.

**Поиск с запретами** Основоположником алгоритма поиска с запретами (Tabu search) является Ф. Гловер, который предложил принципиально новую схему локального поиска [46]. Она позволяет алгоритму не останавливаться в точке локального оптимума, как это предписано в стандартном алгоритме локального спуска, а переходить от одного локального оптимума к другому с целью найти среди них глобальный оптимум. Основным механизмом, позволяющим алгоритму выбираться из локального оптимума, является список запретов. Он строится по предыстории поиска, то есть по нескольким последним решениям $S_k, S_{k-1}, \ldots, S_{k-h+1}$, и запрещает часть окрестности текущего решения $N(S_k)$. Точнее на каждом шаге алгоритма очередное решение $S_{k+1}$ является оптимальным решением подзадачи

$$f(S_{k+1}) = \min\{f(S) \mid j \in N(S_k) \setminus Tabu_h(S_k)\},$$

где $f(S)$ — значение целевой функции решения $S$. Множество $Tabu_h(S_k) \subseteq N(S_k)$ определяется по предшествующим решениям. Список запретов учитывает специфику задачи и, как правило, запрещает использование тех "фрагментов" решения, которые менялись на последних $h$ шагах алгоритма. Константа $h \geq 0$ определяет его память. При "короткой памяти"($h = 0$)

получаем стандартный локальный спуск.

В работе [20] предлагается два алгоритма поиска с запретами. Один из них использует кодировку списком и последовательную декодирующую процедуру. Окрестность, применяемая в этом алгоритме, основана на понятии критической дуги в некотором ориентированном графе, определяемом по текущему расписанию. Критические дуги заключают в себе смысл узкого места в расписании. Поэтому при построении соседнего решения основной упор делается на локальную перестройку именно таких частей текущего расписания. Второй алгоритм использует кодировку в виде логической схемы со специально разработанной декодирующей процедурой. Оба алгоритма используют динамический список запретов и алгоритмы, основанные на приоритетных правилах при генерации начального решения.

В работе [77] предлагается алгоритм, также использующий список в качестве кодировки. В текущем расписании с помощью некоторого ориентированного графа выявляются пары работ, конфликтующие из-за ресурсов. Процедура построения элемента окрестности ставит такие работы в новое место. Помимо классической ЗКПОР этот алгоритм используется для решения многих других моделей календарного планирования.

**Генетический алгоритм** Идея генетических алгоритмов заимствована у живой природы и состоит в моделировании эволюционного процесса, конечной целью которого является получение оптимального решения сложной комбинаторной задачи. Разработчик генетических алгоритмов выступает в данном случае как "создатель", который должен правильно установить законы эволюции, чтобы достичь желаемой цели как можно быстрее. Стандартный генетический алгоритм начинает свою работу с формирования начальной *популяции* $POP_0 = \{S_1, S_2, \ldots, S_k\}$ — конечного набора допустимых решений задачи. Эти решения могут быть выбраны как случайным образом так и с помощью приближенных алгоритмов, например вероятностных жадных. На каждом шаге эволюции с помощью вероятностного оператора *селекции* выбираются два решения, *родители* $S_i, S_j$. Оператор *скрещивания* по решениям $S_i, S_j$ строит новое решение $S'$, которое затем подвергается небольшим случайным модификациям, которые

принято называть *мутациями*. Затем решение добавляется в популяцию, а решение с наихудшим значением целевой функции удаляется из популяции.

В работе [50] было предложено три варианта генетического алгоритма, использующие соответственно кодировки в виде списка, вектора значений приоритета и вектора приоритетных правил. Во всех трех использовалась последовательная ДП, а также три оператора скрещивания: одноточечный, двуточечный и равномерный. Результаты экспериментов показали, что алгоритм проявляет себя наилучшим образом с использованием двуточечного оператора скрещивания и кодировки списком. В работе [51] этот алгоритм был модернизирован путем введения чередования последовательно декодера с параллельным, а также фазы локального поиска.

**Муравьиные колонии** Алгоритм муравьиной колонии (МК) возник при моделировании поведения муравьев [37]. Известно, что муравьи фактически не имеют зрения, но способны каким-то образом находить кратчайший путь от источника пищи до муравейника. Двигаясь по местности, они оставляют за собой след в виде сильно пахнущего вещества – феромона. Именно запах позволяет муравьям ориентироваться на местности. При выборе направления с большей вероятностью выбирается направление с более сильным запахом.

Основная идея алгоритма МК состоит в реализации принципа *коллективного разума*. Для поиска экстремума целевой функции алгоритм использует параллельно несколько агентов (искусственных муравьев), которые в ходе поиска накапливают статистическую информацию. Эта информация аккумулируется в общедоступном банке данных и используется агентами независимо друг от друга. Каждый агент действует по правилам вероятностного алгоритма и при выборе направления ориентируется не только на приращение целевой функции, но и на статистическую информацию, отражающую предысторию коллективного поиска.

Метод МК является итеративным. На каждой его итерации определенное количество агентов строят такое же количество допустимых решений задачи. Среди этих решений выбирается часть наилучших по целевой функции и в этой части отыскиваются повторяющиеся компоненты реше-

ний. Полученная информация запоминается и на последующих итерациях данные компоненты будут иметь большую вероятность войти в решение, чем это было на предыдущих итерациях.

В работе [74] предлагается алгоритм муравьиной колонии для ЗКПОР. Решение представляется в виде списка, а в качестве декодера используется последовательная процедура. На первой итерации алгоритм строит набор списков подобно вероятностному жадному алгоритму с приоритетным правилом. После отбора наилучших решений формируется матрица $\{\tau_{ij} \mid i, j \in J\}$ частоты попадания работы $j$ на позицию $i$ в наилучших найденных списках. На последующих итерациях эта информация используется при построении новых решений подобно значениям приоритета.

**Гибридные алгоритмы** Несколько нестандартный подход к решению задачи предлагается в работе [86]. Рассматривается комбинация переборного алгоритма и локального поиска. На каждой итерации множество переменных (в данном случае работ) делится на $p$ свободных и $n-p$ фиксированных. После фиксации $n-p$ работ в некотором расписании оставшиеся $p$ работ образуют подпроект размерности $p$, $(p \leq n)$. Эта подзадача решается методом полного перебора с заранее установленным временем работы. Если решение найдено менее чем за указанное время $T$, то значение параметра $p$ увеличивается. В противном случае его значение уменьшается. Затем процесс переходит к следующей итерации.

В работе [104] предлагается эволюционный алгоритм, состоящий из двух фаз. В ходе первой фазы происходит развитие популяции в течение некоторого периода. Управление эволюцией осуществляется путем последовательного применения эффективной процедуры локального улучшения использования ресурсов и техники комбинирования расписаний использующей идеи рассеивающего поиска [46]. Цель второй фазы – исследовать области, содержащие наилучшие найденные решения.

### 0.4.5 Прочие алгоритмы

Кроме перечисленных существует много других алгоритмов для решения ЗКПОР. В некоторых работах предлагается использовать методы ветвей и границ, в которых число генерируемых расписаний ограничивается поли-

номом от входных данных или временем работы процедуры [87, 92]. В некоторых алгоритмах вводятся так называемые дополнительные дуги, расширяющие условия предшествования. Основная идея состоит в формировании минимальных по включению множеств работ, не связанных условиями предшествования, но при этом не способных выполняться одновременно из-за ограничений по ресурсам. [18, 23, 91]. Часть алгоритмов основана на решении задачи, сформулированной в терминах целочисленного программирования [88, 78]. В алгоритме [73] используются блочные структуры в расписании. Текущее расписание делится на несколько частей. Затем алгоритм оптимизирует расписание работ внутри каждого блока с целью сократить длину расписания. В работе [76] предлагается алгоритм, основанный на лагранжевой релаксации задачи. Показано, что оптимальному решению релаксированной модели однозначно соответствует оптимальное решение задачи о минимальном разрезе. Вычисление множителей Лагранжа осуществляется стандартными методами субградиентной оптимизации. В работе также показано, что полученная нижняя оценка совпадает с оценкой линейного программирования.

Результаты экспериментов, приведенные в обзорных статьях [63, 64] показывают, что в большинстве случаев метаэвристики выигрывают у методов с приоритетными правилами. С увеличением числа итераций разрыв в относительной погрешности между этими алгоритмами растет. По-видимому, это происходит потому, что методы с приоритетными правилами не используют информацию, полученную ранее, в то время как метаэвристики на каждой итерации опираются на предысторию.

# Глава 1

# О концентрации локальных оптимумов

В данной главе проводится численное исследование расположения локальных оптимумов в допустимой области для трех окрестностей: линейной, квадратичной и экспоненциальной мощности.

В разделе 1.1 вводятся определения окрестностей и обсуждаются их свойства. Первая окрестность была предложена в работе [20] для алгоритма поиска с запретами. Вторая окрестность является в некотором смысле расширением первой. Третья окрестность основана на решении вспомогательной задачи о многомерном рюкзаке.

В разделе 1.2 приводится описание электронной библиотеки тестовых примеров PSPLib [65]. В разделе 1.3 обсуждаются результаты экспериментов, связанных с исследованием зависимости между числом различных локальных оптимумов и входными данными задачи. Обсуждается вопрос о связи между концентрацией локальных оптимумов и разбросом значений целевой функции.

## 1.1 Окрестности

Для расписания $S$ рассматриваются три окрестности $N_1(S)$, $N_2(S)$ и $N_3(S)$, которые будут использоваться в стандартном алгоритме локального спуска. Первая из них была предложена в работе [20] и строится по так называемым *критическим дугам*. Она имеет линейную мощность относительно числа работ. Вторая окрестность является в некотором смысле расширением первой и строится по всевозможным парам работ, не связанных условиями предшествования. Она имеет квадратичную мощность. Третья окрестность строится на основе локальной перестройки текущего списка.

Окрестность обладает экспоненциальной мощностью, но тем не менее выбор элемента осуществляется из ее линейной части, выделение которой основано на решении вспомогательной задачи о многомерном рюкзаке.

### 1.1.1 Окрестность $N_1(S)$

Окрестность $N_1(S)$ определяется для активного расписания $S$ и соответствующего ему списка работ $L$. Рассмотрим взвешенный ориентированный граф $G = (V, E)$ с множеством вершин $V = J$ и множеством дуг $E$, состоящим из пар $(i, j)$ таких, что $s_i + p_i = s_j$ в расписании $S$. Дуге $(i, j)$ припишем вес $w_{ij} = p_i$. Граф $G$ является сетью с одним источником и, быть может, несколькими стоками. По построению все пути из источника в сток, соответствующий последней фиктивной работе, имеют одинаковую длину. Такие пути будем называть *критическими.*

**Определение 4** *[20] Дугу $(i, j) \in E$ будем называть критической, если она принадлежит некоторому критическому пути и $i \notin P_j$.*

Найдем критический путь $CP$ с минимальным числом дуг. Идея построения окрестности $N_1(S)$ состоит в том, чтобы изменить список $L$, сделав хотя бы одну критическую дугу некритической для пути $CP$. Окрестность $N_1(S)$ определяется с помощью трех операторов.

1) $Shift_{ij}$ — оператор, который определяется для критической дуги $(i, j)$ пути $CP$, если работа $i$ стоит в списке $L$ раньше работы $j$. Оператор предъявляет новый список $L_1$, перемещая работу $i$ вместе со всеми ее последователями $u_1, \ldots, u_m$, стоящими перед $j$ в $L$, непосредственно за работу $j$:

$$L = (\ldots, i, \ldots, u_1, \ldots, u_m, \ldots, j, \ldots) \Rightarrow L_1 = (\ldots, j, i, u_1, \ldots, u_m, \ldots).$$

2) $BShift_{ij}$ — оператор, который определяется для критической дуги $(i, j)$ пути $CP$, если $i$ стоит в $L$ позже $j$. Оператор предъявляет новый список $L_2$, перемещая ближайшую справа от $i$ работу $u$, не связанную с $i$ условиями предшествования, непосредственно перед $i$:

$$L = (\ldots, j, \ldots, i, \ldots, l, \ldots, u, \ldots) \Rightarrow L_2 = (\ldots, j, \ldots, u, i, \ldots, l, \ldots).$$

3) $FShift_{ij}$ — оператор, аналогичный оператору $BShift_{ij}$. Он предъявляет список $L_3$, перемещая ближайшую слева от $j$ работу $u$, не связанную с $j$

условиями предшествования, непосредственно за элемент $j$:

$$L = (\dots, u, \dots, l, \dots, j, \dots, i, \dots) \Rightarrow L_3 = (\dots, l, \dots, j, u, \dots, i, \dots).$$

Каждый из указанных списков порождает активное расписание. Множество расписаний, порожденных таким образом с помощью указанных операторов по всем критическим дугам пути $CP$, образует окрестность $N_1(S)$. Она имеет мощность порядка $O(n)$.

Важным свойством любой окрестности является свойство *оптимальной связности.*

**Определение 5** *Графом смежности относительно окрестности $N(S)$ называется такой граф $G = (V, E)$, в котором множество вершин есть множество допустимых решений задачи, а множество дуг — совокупность таких пар $(S_1,\ S_2)$, для которых $S_2 \in N(S_1),\ S_1, S_2 \in V$.*

**Определение 6** *Окрестность $N(S)$ будем называть оптимально связной, если в графе смежности относительно $N(S)$ существует путь из произвольной вершины в вершину, соответствующую оптимальному решению. Соответственно, окрестность будем называть связной, если в графе смежности существует путь между двумя произвольными вершинами.*

**Теорема 4** *Окрестность $N_1(S)$ не является оптимально связной.*

**Доказательство.** Рассмотрим следующий контрпример. Пусть $n = 5,\ |K| = 1,\ P_1 = P_2 = P_3 = P_4 = \emptyset,\ P_5 = \{2, 3, 4\},\ p_1 = p_5 = 7,\ p_2 = p_3 = p_4 = 2,\ r_{11} = r_{51} = 3,\ r_{21} = r_{31} = r_{41} = 2,\ R_1 = 6$. Рассмотрим список $L = (1, 2, 3, 4, 5)$. Ему соответствует расписание $S:\ s_1 = 0,\ s_2 = 0,\ s_3 = 2,\ s_4 = 4,\ s_5 = 6$ (см. рисунок 1.1).

Длина расписания $S$ равна 13. В этом расписании существует две критические дуги, (2,3) и (3,4), к которым можно применить только оператор $Shift$ ввиду порядка работ в списке $L$. Заметим, что пара (4,5) не является критической дугой, поскольку $4 \in P_5$. Оператор $Shift$, действуя на список $L$ по указанным критическим дугам, позволяет строить только расписания, отличающиеся от текущего лишь взаимным расположением работ 2, 3 и 4.

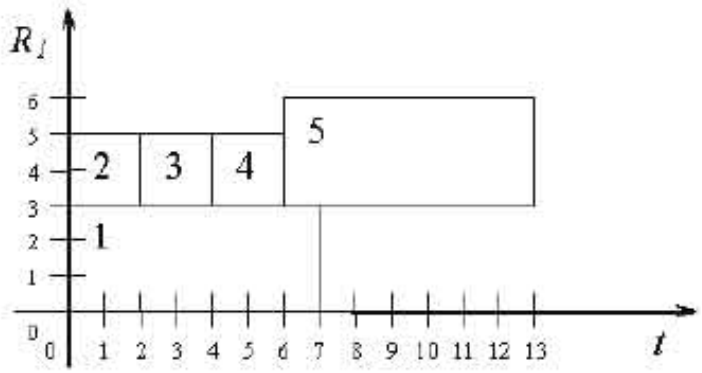


Рис. 1.1: Текущее решение

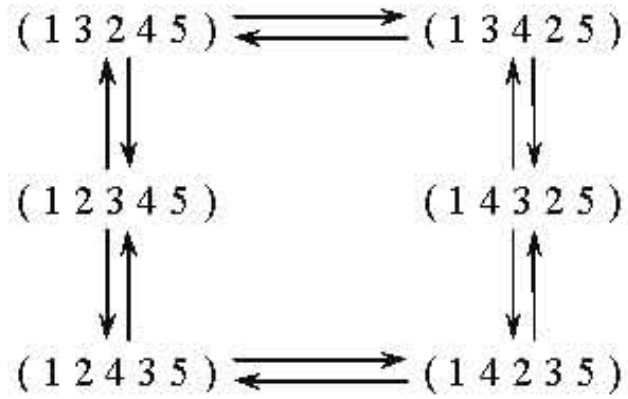


Рис. 1.2: Одна из компонент связности графа смежности окрестности $N_1$

На рисунке 1.2 приведена компонента связности, содержащая решение $L = (1, 2, 3, 4, 5)$, графа смежности окрестности $N_1$. Оптимальное же расписание приведено на рисунке 1.3. Один из соответствующих ему списков — $L^* = (2, 3, 4, 1, 5)$. Теорема доказана.

### 1.1.2 Окрестность $N_2(S)$

Окрестность $N_2(S)$ также как и окрестность $N_1(S)$ определяется для активного расписания $S$ и соответствующего ему списка $L$. Для каждой пары $(i, j)$, не связанной условиями предшествования, соседний список $L'$ получается из списка $L$ переносом работы $i$ вместе со всеми ее последователями, стоящими перед $j$ в $L$, непосредственно за работу $j$. Отличие от оператора $Shift_{ij}$ состоит в том, что пара $(i, j)$ не обязана соответствовать критической дуге и может не принадлежать $E$. Окрестность $N_2(S)$ состоит из активных расписаний, получаемые из соседних списков. Она имеет мощ-

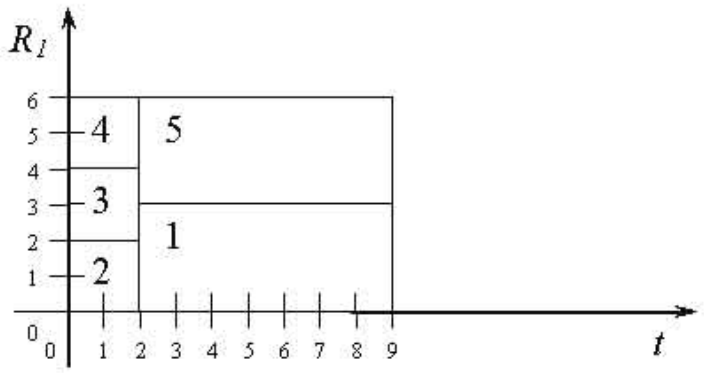


Рис. 1.3: Оптимальное решение

ность порядка $O(n^2)$.

Следует отметить, что такой способ построения соседнего решения позволяет построить любой допустимый список. Следовательно, связность окрестности $N_2(S)$ не вызывает сомнения.

### 1.1.3 Окрестность $N_3(S)$

При определении окрестности $N_3(S)$ используется приближенное решение задачи о многомерном рюкзаке. Эта окрестность успешно применялась в алгоритме поиска с запретами (см. главу 3). Пусть $S$ — текущее активное расписание. Построим по нему список $L$, упорядочив работы по возрастанию моментов их начала в расписании $S$. Работы с одинаковыми моментами начала сортируются в порядке возрастания их номеров. Далее, по каждой работе $j \in J$ в списке $L$ выделяется сегмент $L(j)$, в результате чего список $L$ делится на три части. Соседнее решение $L'$ отличается от текущего решения $L$ порядком расположения работ в средней части. Понятно, что существует много способов организации этого порядка и в этом смысле каждая работа $j$ порождает целый набор соседних решений $N^j(S)$, число элементов которого, вообще говоря, экспоненциально. Таким образом, окрестность $N_3(S)$, представляющая собой объединение $\bigcup_{j=1}^{n} N^j(S)$, имеет экспоненциальную мощность.

Итак, множество соседних решений $N^j(S)$, соответствующих работе $j$, задается сегментом $L(j)$ в текущем списке $L$. Для построения сегмента нам потребуются следующие определения.

**Определение 7** *Блоком работы $j$ в расписании $S$ будем называть мно-*

*жество работ, выполняющихся параллельно с работой $j$, либо начинающих выполняться сразу за ней или заканчивающихся непосредственно перед ней:*

$$Bl_j = \{i \in J \mid [s_i, c_i] \cap [s_j, c_j] \neq \emptyset\}.$$

Наряду с блоком введем понятие *исходящей сети*. Рассмотрим орграф $G_S(V, E)$ с множеством вершин $V = J$ и множеством дуг $E = \{(i, j) \mid c_i = s_j,\ i \in P_j\}$.

**Определение 8** *Исходящей сетью работы $j$ для расписания $S$ будем называть максимальный по включению связный подграф графа $G_S$, единственным источником которого является вершина, соответствующая работе $j$.*

Исходящая сеть позволяет найти все работы в расписании $S$, которые по условиям предшествования не могут начаться раньше указанного им срока, если время выполнения работы $j$ не меняется.

**Определение 9** *Входящей сетью работы $j$ для расписания $S$ будем называть максимальный по включению связный подграф графа $G_S$, единственным стоком которого является вершина, соответствующая работе $j$.*

Если блок работы $j$ не содержит ее предшественников, т.е. $Bl_j \cap P_j = \emptyset$, то началом сегмента $L(j)$ полагается работа блока $Bl_j$ с минимальной позицией в списке $L$, которая обозначается через $First_j$. Конец сегмента, обозначаемый через $Last_j$, определяется как максимум из двух чисел: максимальной позиции в списке $L$ для работ блока $Bl_j$ и исходящей сети. Заметим, что если блок работы $j$ содержит предшественника $i \in P_j$, то это означает, что $j \in L(i)$. В этом случае работа $j$ может быть без ущерба исключена из рассмотрения, поскольку множество сегментов $\{L(j) \mid Bl_j \cap P_j = \emptyset,\ j \in J\}$ покрывает весь список $L$. Этот факт в сочетании с тем, что всякий допустимый порядок работ в сегменте $L(j)$ охватывается множеством соседних решений $N^j(S)$, означает, что окрестность $N_3(S)$ связна.

**Лемма 1** *Пусть $L = (j_1, \ldots, j_n)$ — произвольный допустимый список и $S = \{s_j\}_{j=j_1}^{j_n}$ — соответствующее ему активное расписание, полученное последовательным декодером. Тогда для любой работы $j_k$, $k = 2, \ldots, n$ найдется работа $j_m$, такая что $s_{j_m} + p_{j_m} = s_{j_k}$ и $m < k$.*

**Доказательство.** Последовательный декодер строит расписание шаг за шагом, последовательно включая очередную работу в частичное расписание в наиболее ранний допустимый момент согласно условиям предшествования и ограничениям по ресурсам. Рассмотрим произвольную работу $j_k,\ k = 2, \ldots, n$ и частичное расписание, образованное работами $j_1, \ldots, j_k$. Поскольку оно является активным, работа $j_k$, будучи начатой в момент времени $s_{j_k}$, не может быть начата в момент времени $s_{j_k} - 1$. Это означает, что либо существует работа $j_m$, являющаяся предшественником работы $j_k$, которая заканчивается в момент $s_{j_k}$ и тогда справедливо $m < k$ в силу допустимости списка $L$, либо объем ресурсов, имеющихся в момент времени $s_{j_k} - 1$ не достаточен для выполнения работы $j_k$, то есть ресурс используется другой работой, которая должна закончиться к моменту $s_{j_k}$, поскольку ресурс освобождается. То есть существует работа $j_m$, такая что $s_{j_m} + p_{j_m} = s_{j_k}$. Так как рассматривается частичное расписание, образованное работами $j_1, \ldots, j_k$ и очевидно $m \neq k$, справедливо $m < k$. Лемма доказана.

**Следствие 1** *Если работа $j \in J$ занимает позицию $k > 1$ в списке $L$, то $First_j < k$.*

**Теорема 5** *Окрестность $N_3(S)$ является связной.*

**Доказательство.** Пусть $L' = (j_1, \ldots, j_n)$ — произвольный список. Рассмотрим другой произвольный список $L$. Покажем, что существует цепочка конечной длины $L = L_0,\ L_1, \ldots, L_g = L'$, в которой каждый последующий элемент является соседним решением предыдущего. Рассмотрим работу $j_1$, занимающую первую позицию в списке $L'$. Если в списке $L$ она также занимает первую позицию, то переходим к работе $j_2$. Пусть работа $j_1$ занимает в списке $L$ позицию $m > 1$. Рассмотрим ее блок. Поскольку работа $j_1$, занимая первую позицию в списке $L'$, не может иметь предшественников, то ее блок также не содержит предшественников. Поэтому ей соответствует некоторый набор соседних решений. Если $s_{j_1} = 0$, то поскольку $m > 1$, то найдется работа $i,\ s_i = 0$, принадлежащая блоку работы $j_1$ и занимающая первую позицию в списке $L$. Очевидно, что в этом случае $First_{j_1} = 1$. Тогда найдется соседнее решение $L_1$, в котором работа $j_1$ занимает позицию $First_{j_1} = 1$. Если $s_{j_1} > 0$, то найдется такая (быть может не одна) работа

$i$, для которой $s_i + p_i = s_{j_1}$. Рассмотрим такую работу $i$, занимающую минимальную позицию $m_1$ в списке $L$. Очевидно, что $i \in Bl_{j_1}$ и $m_1 = First_{j_1}$. Тогда найдется соседнее решение $L_1$, в котором работа $j_1$ занимает позицию $m_1$, а по следствию из леммы 1 $m_1 < m$. Если $m_1 > 1$, то повторим вышеизложенные рассуждения и аналогичным образом определим позицию $m_2 < m_1 < m$ и так далее пока работа $j_1$ на займет первую позицию в текущем списке. Далее переходим к работе $j_2$ и так далее. Допустим, что в текущем решении $L_k$ работы $j_1, \ldots, j_k$ занимают первые $k$ позиций. Рассмотрим работу $j_{k+1}$. Пусть она в списке $L_k$ занимает позицию $m > k+1$. Если блок работы $j_{k+1}$ не содержит ее предшественников, то рассмотрим соответствующую ему позицию $First_{j_{k+1}}$. Если $First_{j_{k+1}} \leq k$, то найдется такое соседнее решение $L_{k+1}$, в котором первые $k - First_{j_{k+1}} + 1$ работ блока $Bl_{j_{k+1}}$ сохраняют свои позиции, а позиция $k+1$ соответствует работе $j_{k+1}$. Если $First_{j_{k+1}} > k$, то в качестве очередного элемента $L_{k+1}$ искомой цепочки рассматривается соседнее решение, в котором работа $j_{k+1}$ занимает позицию, равную $First_{j_{k+1}}$. По следствию 1 $First_{j_{k+1}} < m$. Если при этом все еще $First_{j_{k+1}} > k+1$, то повторим рассуждения, иначе переходим к работе $j_{k+2}$. Осталось рассмотреть последний случай когда блок работы $j_{k+1}$ содержит ее предшественников. Рассмотрим работу $i \in P_{j_{k+1}}$, являющуюся одним из источников входящей сети работы $j_{k+1}$. Понятно, что $Bl_i \cap P_i = \emptyset$ и работа $j_{k+1}$ принадлежит исходящей сети работы $i$. Следовательно, $First_i < k+1 < m \leq Last_i$. Тогда найдется такое соседнее решение $L_{k+1}$, определяемое по работе $i$, в котором работы, занимающие позиции $First_i, \ldots, k$, сохраняют их, а работа $j_{k+1}$ занимает позицию $k+1$. Таким образом, все работы последовательно перемещаются на позиции, которые они должны занимать в списке $L^*$. Теорема доказана.

Поскольку окрестность $N_3(S)$ обладает экспоненциальной мощностью, то ее просмотр является достаточно трудоемким процессом. Поэтому из каждого множества $N^j(S)$ выбирается одно соседнее решение, исходя из условия наиболее эффективного использования имеющихся ресурсов работами сегмента $L(j)$. Для определения порядка работ в $L(j)$ будем применять модификацию параллельного декодера, в которой на шаге 2.3. будет выбираться подмножество работ из допустимого множества $D(t_m)$ по критерию максимизации использованных ресурсов [104]. Введем переменные

$x_i \in \{0,1\},\ i \in D(t_m)$. Если $x_i = 1$, то для $i$-й работы из множества $D(t_m)$ положим $s_i := t_m$. Если же $x_i = 0$, то $s_i > t_m$. Формально, выбор нужного подмножества означает решение следующей задачи о многомерном рюкзаке.

Найти:

$$\max \sum_{i \in D(t_m)} x_i \sum_{k \in K} \frac{r_{ik}}{R_k},$$

при ограничениях

$$\sum_{i \in D(t_m)} r_{ik} x_i \leq R_k - \sum_{i \in A(t_m)} r_{ik},\ k \in K,$$

$$x_i \in \{0,1\},\ i \in D(t_m).$$

Целевая функция задачи требует максимизации суммарной доли использованных ресурсов в момент времени $t_m$. Ограничения задачи устанавливают границы потребления ресурсов. Для решения данной задачи применяется вероятностный жадный алгоритм, о котором пойдет речь в следующем разделе.

С помощью указанной модификации параллельного декодера получаем новое расписание, составленное для работ списка $L(j)$. Порядок работ в этом расписании задает новый порядок работ сегмента $L(j)$ в соседнем решении.

Заметим, что множество вершин соответствующего графа окрестностей не охватывает всего множества активных расписаний, поскольку расписание, составленное из работ $L(j)$ параллельным декодером относится к классу *плотных* [93] расписаний. Класс плотных расписаний является подклассом активных расписаний, однако для него известно, что он может не содержать оптимального расписания. Вообще говоря, процедура построения расписания для работ $L(j)$ может быть адаптирована путем внесения так называемых "пустых" моментов времени когда не начинается ни одна работа, вследствие чего может быть построено любое активное расписание, но это, в свою очередь, не гарантирует конечности процедуры. Поэтому вопрос относительно оптимальной связности окрестности $N_3(S)$ пока остается открытым.

### 1.1.4 Задача о многомерном рюкзаке

При определении окрестности $N_3(S)$ существенно использовалась задача о многомерном рюкзаке. Известно [72], что она $NP$-трудна даже при $\mathbf{K} = 1$. В связи с этим рассмотрим вероятностный жадный алгоритм [3], который позволяет быстро получать для нее приближенные решения.

Разделим каждое ограничение на $R_k,\ k \in K$. При $\mathbf{K} = \mathbf{1}$ получаем одинаковые коэффициенты в целевой функции и ограничениях, т. е. удельные стоимости одинаковы и равны 1. Если $\mathbf{K} > 1$, то удельных стоимостей несколько, но их сумма равна 1:

$$\sum_{k \in K} \frac{r_{ik}/R_k}{\sum\limits_{k' \in K} r_{ik'}/R_{k'}} = 1.$$

Таким образом, для решения возникающей задачи о многомерном рюкзаке целесообразно использовать аналог жадных алгоритмов GS и MTGS (см. [72], гл.4), разработанных для задачи о камнях. Эти алгоритмы используют только коэффициенты целевой функции при выборе очередного элемента.

Итак, пусть $D(t_m)$ — допустимое множество, соответствующее моменту $t_m$. Зададимся величиной $0 < q \leq 1$ и сформируем случайное подмножество $D(q) \subseteq D(t_m)$. Каждый элемент из $D(t_m)$ включается в множество $D(q)$ с вероятностью $q$ независимо от других элементов.

Найдем в $D(q)$ элемент с максимальным весом $\sum_{k \in K} r_{ik}/R_k,\ i \in D(q)$. Положим значение соответствующей переменной $x_i$ равным 1, уменьшим правые части ограничений и удалим из $D(t_m)$ выбранный элемент и все элементы, которые уже не помещаются в "рюкзак". Будем повторять эту процедуру до тех пор, пока множество $D(t_m)$ не станет пустым. На выходе имеем набор значений булевых переменных $x_i \in \{0, 1\},\ i \in D$. Формально этот алгоритм может быть представлен следующим образом.

Алгоритм BMP$(q, t_m, A(t_m), D(t_m))$

1. Положить $\overline{D} := D(t_m),\ \widetilde{R}_k := R_k - \sum_{i \in A(t_m)} r_{ik},\ x_i := 0,\ i \in \overline{D}$.
2. Выбрать в $\overline{D}$ случайное подмножество $D(q)$.
   Если $D(q) = \emptyset$, то добавить в $D(q)$ любой элемент из $\overline{D}$.

3. Найти $i_0 \in D(q)$ с максимальным весом $\sum_{k\in K} r_{i_0 k}/R_k = \max\limits_{i\in D(q)} \sum_{k\in K} r_{ik}/R_k$.

4. Положить $x_{i_0} := 1$; $\overline{D} := \overline{D} \setminus \{i_0\}$, $\widetilde{R}_k := \widetilde{R}_k - r_{i_0 k}$, $k \in K$.

5. Для всех $i \in \overline{D}$: если $r_{ik} > \widetilde{R}_k$ для некоторого $k \in K$,
то положить $\overline{D} := \overline{D} \setminus \{i\}$.

6. Если $\overline{D} \neq \emptyset$, то вернуться на шаг 2.

Трудоемкость процедуры оценивается величиной $O(n^2\mathbf{K})$. Алгоритм BMP можно применять несколько раз и лучшее из полученных решений использовать при построении окрестности $N_3(S)$. Величина $q$ является параметром алгоритма. При $q = 1$ получаем детерминированный алгоритм GS. При малых значениях $q$ алгоритм многократно формирует множество $D(q)$ и тем самым использует идею алгоритма MTGS, пропуская некоторые элементы с большими весами.

## 1.2 Библиотека тестовых задач

Для тестирования алгоритмов решения задач календарного планирования с ограниченными ресурсами в 1996 году создана специализированная библиотека тестовых задач PSPLib [65]. Ее авторы — германские ученые R. Kolisch и A. Sprecher. Библиотека содержит широкий спектр примеров разной степени сложности как для рассматриваемой здесь модели, так и для многих других моделей календарного планирования, таких как мультимодальная ЗКПОР, классическая и мультимодальная ЗКПОР с минимальными и максимальными временными задержками и задача инвестирования ресурсов с временными задержками [107]. Примеры получены с помощью генератора ProGen [66]. Библиотека имеет сайт в ИНТЕРНЕТ с электронным адресом http://www.bwl.uni-kiel.de/Prod/psplib/index.html, где размещаются файлы исходных данных в свободном доступе. Описание библиотеки, типы рассматриваемых моделей, детальное описание построения примеров, описание параметров задачи и другая необходимая информация содержится в работе [66].

Помимо исходных данных, библиотека содержит оптимальные решения для примеров размерности 30 работ. Для остальных примеров в ней содержатся наилучшие найденные значения целевой функции с указанием ав-

тора и года. Также для этих примеров содержатся наилучшие полученные значения нижних оценок. Для некоторых примеров нижняя оценка совпадает с верхней, что свидетельствует о получении оптимума, но число таких примеров невелико относительно общего числа примеров. Информация о получаемых рекордах регулярно обновляется.

Для ЗКПОР в этой библиотеке имеется по 480 примеров размерности $n$=30, 60 и 90 работ и 600 примеров для $n$=120. Число типов ресурсов равно 4. Примеры разбиты на классы. В каждом классе содержится по 10 примеров, порожденных с помощью датчика псевдослучайных чисел при фиксированных значениях трех параметров:

- $NC \in \{1,5; 1,8; 2,1\}$ — *фактор сетевой сложности* — среднее число непосредственных предшественников каждой работы.
- $RF \in \{0,25; 0,50; 0,75; 1,00\}$ — *фактор потребления* — отношение числа типов ресурсов, необходимых для выполнения каждой работы к общему числу типов ресурсов.
- $RS \in \{0,2; 0,5; 0,7; 1,0\}$ для $n = 30,\ 60$ и $RS \in \{0,1; 0,2; 0,3; 0,4; 0,5\}$ для $n = 120$ — *фактор выделения* — объем выделяемых ресурсов в каждый момент времени; значение $RS = 0,1$ соответствует примерам с минимальным объемом выделяемых ресурсов, достаточным для разрешимости задачи; значение $RS = 1$ соответствует случаю неограниченных ресурсов.

Известно [67, 68, 107], что значения параметров $RF = 4,\ RS = 0,2$ для $n = 30,\ 60$ и $RS = 0,1$ для $n = 120$ соответствуют наиболее трудным классам. В таких классах каждая работа требует ресурсы каждого типа, имеет в среднем одного–двух предшественников, а суммарно выделяемые ресурсы минимальны для существования допустимых решений [67]. Примерами таких классов являются:
j30 13, j30 29, j30 45 для $n = 30$,
j60 13, j60 29, j60 45 для $n = 60$,
j120 16, j120 36, j120 56 для $n = 120$.
Для этих примеров наблюдается наибольший разрыв между лучшей известной верхней и нижней оценкой. На этих классах проводилась основная часть численных экспериментов.

## 1.3 Экспериментальные исследования

### 1.3.1 Исследование зависимости числа локальных оптимумов от входных данных задачи

Для проведения эксперимента построим на множестве допустимых решений задачи 1000 расписаний случайным образом. Алгоритм построения случайного списка $L$ состоит из $n$ шагов. На каждом шаге $m = 1, \ldots, n$ рассматривается множество работ $D_m$, все предшественники которых уже включены в список $L$. Затем из указанного множества равновероятно выбирается одна работа, которая вносится в список $L$ на позицию $m$. При $m = 1$ множество $D_1$ состоит из работ, не имеющих предшественников. Понятно, что такой способ построения списка позволяет получить любой допустимый список и, соответственно, любое активное расписание.

К каждому из полученных случайных решений применяется процедура стандартного локального спуска по каждой из рассматриваемых окрестностей. В результате получается 1000 локальных оптимумов относительно каждой из окрестностей. Заметим, что некоторые из них могут совпадать. Целью первого эксперимента является исследование вопроса о числе разных локальных оптимумов в разных классах тестовых примеров.

Как отмечалось выше, тестовые примеры библиотеки PSPLib различаются числом работ, условиями предшествования и количественными характеристиками выделения и потребления ресурсов. Число найденных локальных оптимумов относительно окрестности $N_2(S)$, полученное в разных классах тестовых примеров размерности 30 работ показано на рисунке 1.4.

Результаты показывают, что в данной библиотеке наибольшим числом локальных оптимумов обладают классы с наиболее жесткими ограничениями по ресурсам. Примеры с менее жесткими ресурсными ограничениями обладают малым числом локальных оптимумов. В частности, при $RS = 1$, когда имеющихся ресурсов достаточно для выполнения всех работ, локальный оптимум всегда один. Он же является и глобальным. Это объясняется тем, что соседние решения относительно рассматриваемых окрестностей являются активными расписаниями. Нетрудно показать, что в примерах с неограниченными ресурсами всякое активное расписание является оптимальным. С увеличением размерности задачи число локальных оптимумов

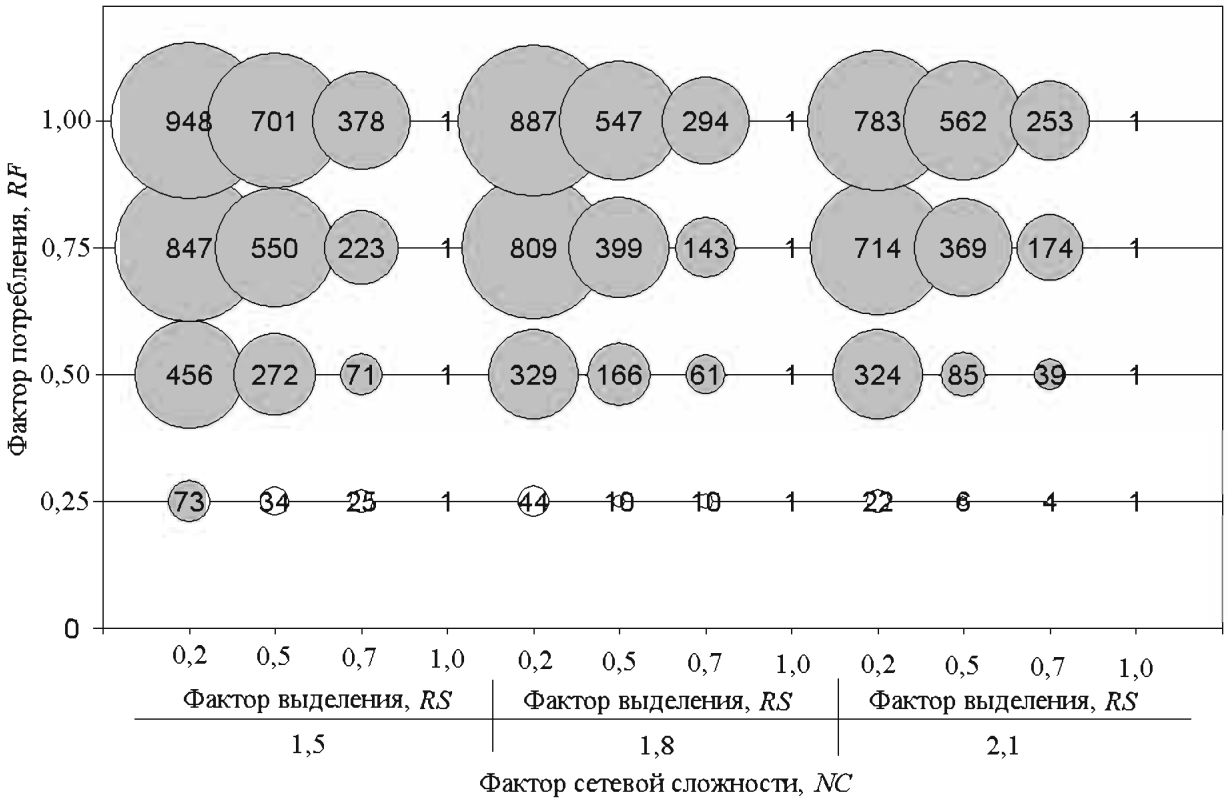


Рис. 1.4: Число локальных оптимумов

существенно возрастает. В таблице 1.1 приведены результаты для шести классов тестовых примеров размерности 30, 60 и 120 работ для трех рассматриваемых окрестностей. Классы J3013, J6013 и J12016 обладают более жесткими ресурсными ограничениями по сравнению с классами J305, J605 и J1206.

Данные таблицы показывают как растет число локальных оптимумов с увеличением размерности задачи. Это свойственно всем рассматриваемым окрестностям. Однако это число зависит от выбора окрестности. Было бы естественным предполагать, что с ростом мощности окрестности число локальных оптимумов должно сокращаться. Тем не менее результаты показывают, что это не всегда так. В рассматриваемой выборке из 1000 решений число локальных оптимумов для квадратичной окрестности $N_2(S)$ не меньше, а в некоторых классах даже больше, чем для окрестностей $N_1(S)$ и $N_3(S)$ с линейным числом соседей. Данное наблюдение говорит о том, что окрестность $N_2(S)$, несмотря на квадратичную мощность, по-видимому не обладает большей областью притяжения по сравнению с окрестностями

| Окрестность | Число найденных локальных оптимумов | | Средняя глубина спуска | | Максимальная глубина спуска | |
|---|---|---|---|---|---|---|
| | J305 | J3013 | J305 | J3013 | J305 | J3013 |
| $N_1$ | 329 | 343 | 1,3 | 2,1 | 8 | 7 |
| $N_2$ | 448 | 989 | 2,8 | 3,3 | 8 | 8 |
| $N_3$ | 371 | 896 | 1,6 | 2,5 | 5 | 7 |
| | J605 | J6013 | J605 | J6013 | J605 | J6013 |
| $N_1$ | 998 | 1000 | 2,3 | 3,2 | 10 | 10 |
| $N_2$ | 1000 | 1000 | 4,5 | 6,3 | 11 | 11 |
| $N_3$ | 998 | 1000 | 3,7 | 5 | 10 | 12 |
| | J1206 | J12016 | J1206 | J12016 | J1206 | J12016 |
| $N_1$ | 1000 | 1000 | 2,8 | 4,8 | 10 | 14 |
| $N_2$ | 1000 | 1000 | 5,7 | 9,7 | 14 | 15 |
| $N_3$ | 1000 | 1000 | 5,5 | 8,4 | 16 | 18 |

Таблица 1.1: Число локальных оптимумов и глубина спуска

$N_1(S)$ и $N_3(S)$. Последним также свойственна меньшая глубина спуска (см. таблицу 1.1).

### 1.3.2 Исследование концентрации локальных оптимумов

При определении окрестности $N_1(S)$ был введен оператор сдвига $Shift_{ij}(L)$ как элементарное преобразование списка $L$. Решения $L$ и $L'$ будем называть *близкими*, если второе может быть получено из первого за небольшое число элементарных преобразований. В численных экспериментах это число полагалось равным 4, 8 и 12 для задач размерности 30, 60 и 120 работ, соответственно. Для каждого полученного локального оптимума выделим множество близких локальных оптимумов. Назовем его *шаром*. Сам локальный оптимум назовём *центральным*. Для каждого шара определим понятие *веса* как среднее значение целевой функции принадлежащих ему локальных оптимумов.

На рисунках 1.5—1.7 приведены пузырьковые диаграммы, отражающие зависимость веса шара от значения целевой функции центрального локального оптимума для трех рассматриваемых окрестностей. На диаграмме ось $Y$ соответствует значениям центрального локального оптимума. Ось $X$ соответствует весам соответствующих шаров. Размер пузырьков соответствует мощности шаров. Результаты показывают, что для близких локаль-

ных оптимумов характерен небольшой разброс значения целевой функции. Другими словами рядом с "хорошими"локальными оптимумами преобладают "хорошие"и наоборот. Данное наблюдение является одним из аргументов в пользу разработки алгоритмов локального поиска, поскольку такие алгоритмы в большинстве своем концентрируют усилия на детальном исследовании областей, содержащих лучшие найденные локальные оптимумы.

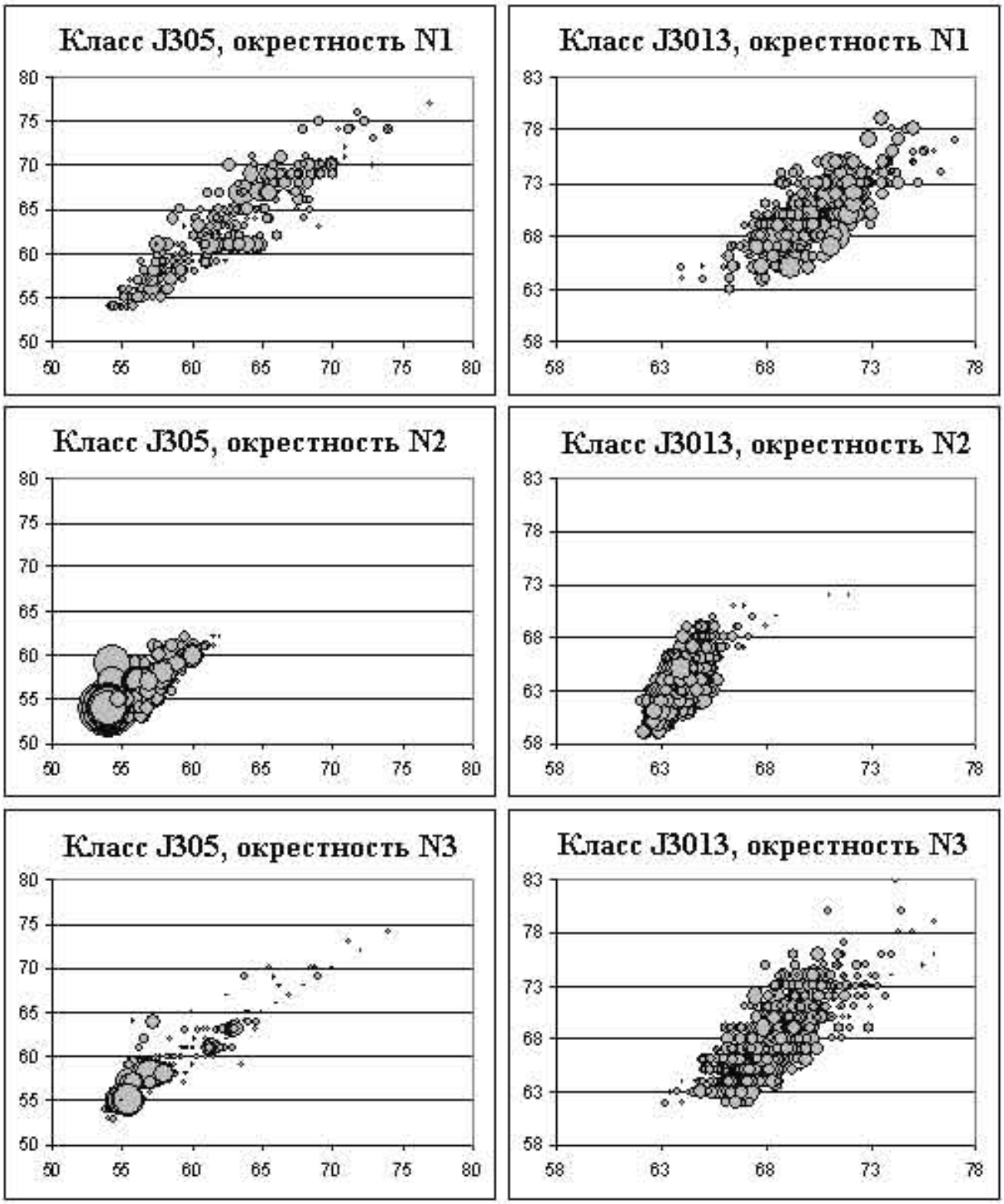


Рис. 1.5: Концентрация локальных оптимумов, n=30

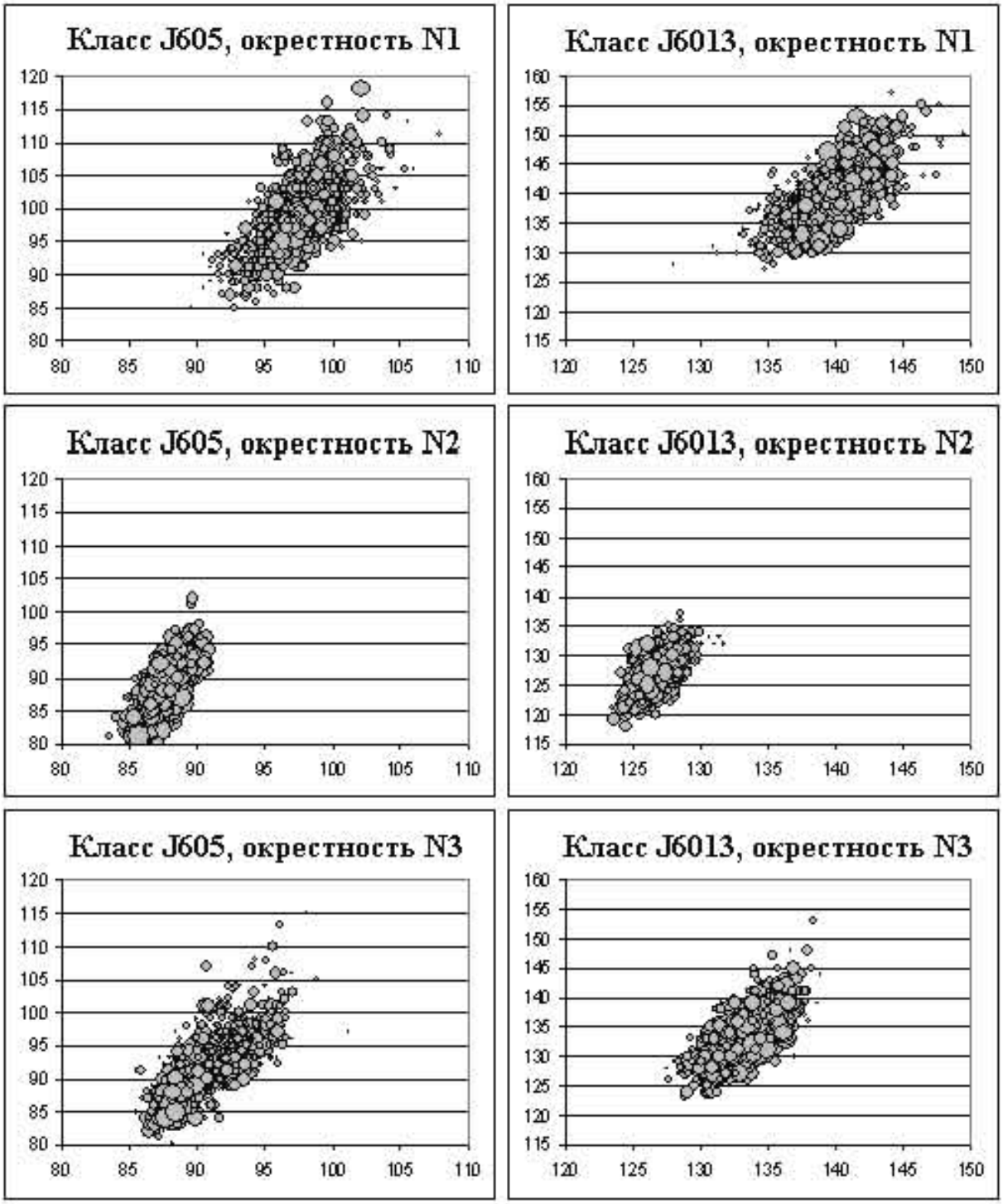


Рис. 1.6: Концентрация локальных оптимумов, n=60

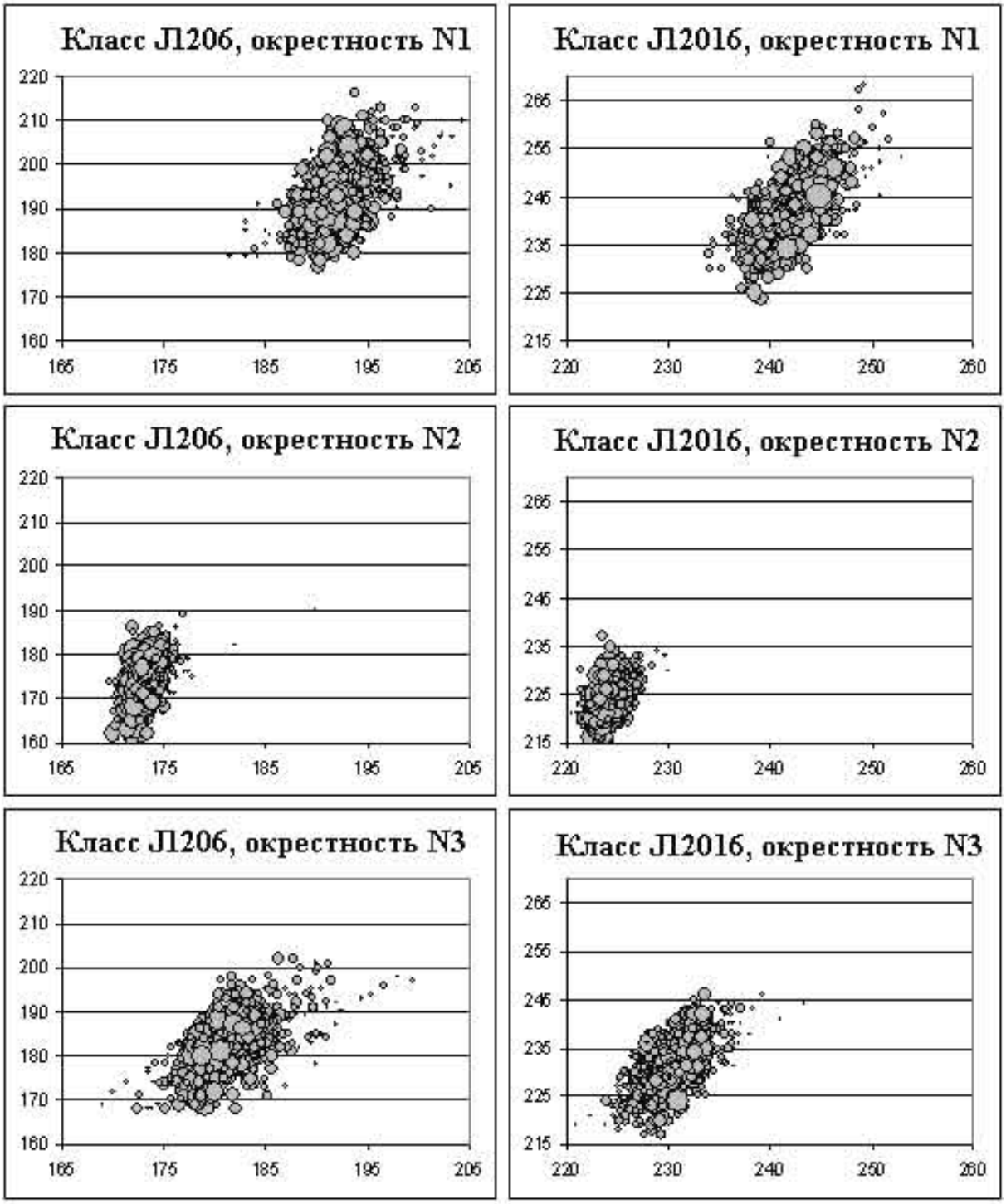


Рис. 1.7: Концентрация локальных оптимумов, n=120

# Глава 2

# Новые жадные алгоритмы

В данной главе рассматриваются три вероятностных жадных алгоритма, в которых по-разному трактуется понятие "жадный". Согласно первому алгоритму вычисляются временные задержки работ относительно наиболее поздних времен старта в задаче без ресурсных ограничений. Эти задержки используются для ранжирования работ и построения приближенного решения по схеме параллельного составления расписаний. Второй алгоритм использует оптимальное решение вспомогательной задачи на узкое место, в которой наряду с временными задержками явным образом учитываются ресурсные ограничения. Идея третьего алгоритма заключается в максимальном использовании выделяемых ресурсов. Для этих целей применяется вспомогательная задача о многомерном рюкзаке. Все алгоритмы являются рандомизированными и применяют методы локальной перестройки полученных жадных решений. Приводятся результаты численных экспериментов и сравнение с ранее разработанными алгоритмами.

Предлагаемый алгоритм решения задачи является серией независимых испытаний вероятностных жадных алгоритмов с последующим улучшением найденных решений методами локальной перестройки. Наилучшее решение, полученное в ходе таких испытаний, является результатом работы данного метода. В иностранной литературе класс таких методов называется GRASP (Greedy Randomized Adaptive Search Procedure) [41]. Аннотированную библиографию с обзором вариантов и приложений GRASP можно найти в [43]. Основная проблема, возникающая при разработке методов этого класса, состоит в разработке жадных алгоритмов и удачной их рандомизации. Введение элемента случайности дает возможность получать много

приближенных решений, которые согласно методам GRASP используются в качестве стартовых точек для алгоритмов локального спуска. Такой подход позволяет сконцентрировать внимание только на локальных оптимумах, но может выводить за рамки полиномиальных алгоритмов [109].

Рассматривается три новые вероятностные жадные стратегии, которые используются в схеме параллельного составления расписаний [60]. В этой схеме рассматриваются возрастающие моменты времени и для каждого из них определяется множество работ, которые будут выполняться параллельно. Выбор данного множества является важным аспектом этой схемы.

Первая эвристика определяет временные задержки работ относительно наиболее поздних моментов старта в задаче без ресурсных ограничений. Идея этой эвристики состоит в ранжировании работ по временным задержкам. Наибольший приоритет получают работы с большими задержками.

Вторая эвристика также использует наиболее поздние моменты старта работ в задаче без ресурсных ограничений, но в отличие от предшествующей эвристики здесь решается оптимизационная задача на узкое место, в которой ресурсные ограничения учитываются явным образом. Задача на узкое место является полиномиально разрешимой, но оптимальных решений может быть экспоненциально много, и они могут существенно отличаться по числу параллельно выполняемых работ и использованию ресурсов. Поэтому среди оптимальных решений ищется решение с минимальными остатками ресурсов.

Третья эвристика старается использовать имеющиеся ресурсы в максимально полном объеме. Эта идея формулируется в виде задачи о многомерном рюкзаке, которая является NP-трудной [72]. Однако точное решение в данном случае не требуется. Для метода GRASP важно иметь много разных приближенных решений. Для задачи о рюкзаке уже разработаны жадные эвристики. В новой эвристике используются их вероятностный аналог с обобщением на многомерный случай.

Все три эвристики строят решения, которые могут иметь достаточно большое отклонение от оптимума, так как являются полиномиальными алгоритмами. Для сокращения погрешности предусмотрена стадия улучшения, на которой используются процедуры локального спуска. Это весьма трудоемкая стадия, так как просмотр окрестности и выбор лучшего сосед-

него решения требует многократного вычисления целевой функции. Рассмотрены три известные окрестности разной мощности:
— окрестность линейной мощности, построенная на критических дугах [20];
— окрестность квадратичной мощности [10];
— окрестность экспоненциальной мощности, в которой просматривается только перспективное подмножество линейной мощности [8].

Применение локального спуска по выбранным окрестностям приводит к заметному сокращению погрешности. Априорная оценка числа шагов локального спуска не является полиномиальной и весьма далека от реально наблюдаемых величин. Во всех экспериментах это число не превосходило 5, что порождает гипотезу о полиномиальности данной процедуры в среднем. Тем не менее, она требует больших затрат машинного времени и в качестве альтернативы локальному спуску исследовалось поведение алгоритма Пинг-понг [8], который, стремясь сократить длину расписания, последовательно переходит от активных расписаний к $T$-поздним и обратно. Такая процедура не гарантирует получение локального оптимума. Однако с большой вероятностью результат работы метода GRASP с алгоритмом Пинг-понг на стадии улучшения оказывается локальным оптимумом для данных окрестностей. Для вероятности этого события построены доверительные интервалы, которые свидетельствуют о предпочтительности алгоритма Пинг-понг в виду его меньшей трудоемкости.

## 2.1 Метод параллельного составления расписаний

Метод параллельного составления расписаний (МПР) основан на идеях параллельного декодера, описанного во введении. Однако, он заключает в себе несколько иной содержательный смысл. В отличие от декодера, ставящего в соответствие некоторому коду допустимое расписание, МПР ничего не раскодирует. В данном случае он играет роль основы эвристического алгоритма для решения ЗКПОР. Еще одним существенным его отличием от параллельного декодера является то обстоятельство, что на каждом шаге построения расписания из допустимого множества выбирается не одна, а некоторое подмножество работ, совместное выполнение которых не нарушает ограничений по ресурсам.

Как отмечалось выше МПР выполняет не более $n$ шагов. На шаге $m$ рассматривается момент времени $t_m$ и три множества:

- $J(t_m) = \{j \in J \mid c_j \leq t_m\}$ — множество работ, завершенных к моменту времени $t_m$,
- $A(t_m) = \{j \in J \setminus J(t_m) \mid s_j \leq t_m < c_j\}$ — множество работ, находящихся в процессе выполнения в момент времени $t_m$,
- $D(t_m) = \{j \in J \setminus (J(t_m) \cup A(t_m)) \mid P_j \subseteq J(t_m),\ r_{jk} \leq R_k - \sum_{i \in A(t_m)} r_{ik},\ k \in K\}$ — множество работ, которые могут начаться в момент времени $t_m$.

Если множество $D(t_m)$ не пусто, то из него выбирается несколько работ, которые начнут выполняться в момент времени $t_m$. Если же это множество пусто, то определяется минимальный момент времени, когда закончится одна из выполняющихся работ, и процесс переходит к следующему шагу. Формально алгоритм выглядит следующим образом.

Алгоритм МПР:

1. Положить $m := 0,\ t_m := 0,\ J(t_m) := \emptyset,\ A(t_m) := \{1\},\ s_0 := c_0 := 0.$
2. Пока $|J(t_m) \bigcup A(t_m)| < n$ выполнять следующее:
   2.1 Положить $m := m + 1.$
   2.2 Найти $t_m := \min\{c_j \mid j \in A(t_{m-1})\}.$
   2.3 Вычислить $J(t_m),\ A(t_m),\ D(t_m).$
   2.4 Выбрать $D' \subseteq D(t_m)$ так, что $\sum_{j \in D'} r_{jk} \leq R_k - \sum_{j \in A(t_m)} r_{jk}, k \in K.$
   2.5 Положить $s_j := t_m,\ c_j := t_m + p_j,\ j \in D'.$

Стратегия выбора подмножества $D' \subseteq D(t_m)$ на шаге 2.4 является наиболее важным аспектом этого подхода. Свобода в выборе искомого множества открывает широкие перспективы для развития метода.

## 2.2 Вероятностные жадные стратегии

Рассмотрим три вероятностные жадные стратегии выбора подмножества $D'$: выбор на основе временных задержек работ, вычисленных относительно оптимального расписания без учета ресурсных ограничений, выбор с

помощью задачи на узкое место и с помощью задачи о многомерном рюкзаке.

### 2.2.1 Сортировка по временным задержкам

Предположим, что множество $D'$ выбрано и $c_i = t_m + p_i,\ i \in D'$. Рассмотрим работу $j \in D(t_m) \setminus D'$ и определим наиболее ранний момент ее начала, допустимый по ресурсным ограничениям. Искомый момент $t'_j$ является минимальным среди моментов окончания работ из множества $A(t_m) \cup D'$, в который имеется достаточный объем ресурсов для выполнения работы $j$:

$$t'_j = \min \gamma$$

при ограничениях

$$r_{jk} \le R_k - \sum_{i \in A(t_m) \cup D',\ c_i > \gamma} r_{ik},\ k \in K,$$

$$\gamma \in \{c_i \mid i \in A(t_m) \cup D'\}.$$

Величина $t'_j$ может быть найдена с трудоемкостью $O(|A(t_m) \cup D'||K|)$. Отметим, что $t'_j$ зависит от множества $D'$.

Пусть $\widetilde{t}_j$ — максимум среди всех величин $t'_j$, соответствующих всевозможным множествам $D'$. Вычислить эту величину за полиномиальное время не представляется возможным. Поэтому, рассмотрим оценку $\overline{t}_j \ge \widetilde{t}_j$, представляющую собой пессимистический прогноз наиболее раннего момента начала работы $j \in D(t_m) \setminus D'$ для любого $D'$. Определим $\overline{t}_j$ следующим образом. Предположим, что $c_i = t_m + p_i, i \in D' \setminus \{j\}$ и найдем наиболее ранний момент времени $\gamma$, когда работе $j$ хватает имеющихся ресурсов, т. е.

$$\overline{t}_j = \min \gamma$$

при ограничениях

$$r_{jk} \le R_k - \sum_{i \in A(t_m) \cup D(t_m) \setminus \{j\},\ c_i > \gamma} r_{ik},\ k \in K,$$

$$\gamma \in \{c_i \mid i \in A(t_m) \cup D(t_m) \setminus \{j\}\}.$$

Величина $\overline{t}_j$ может быть найдена за $O(|A(t_m) \cup D(t_m)|^2|K|)$ операций.

Пусть $LF_j$ — наиболее поздний момент завершения работы $j$ в задаче без учета ограничений по ресурсам. Тогда величина $\Delta_j = \overline{t}_j - LF_j + p_j$ характеризует задержку работы $j \in D(t_m)$. Соответственно, $\max\{\Delta_j \mid j \in D(t_m)\}$ характеризует задержку всего проекта. Таким образом, целесообразно в первую очередь включать в множество $D'$ работы с большими значениями $\Delta_j$, учитывая ресурсные ограничения. Одна из вероятностных версий такой жадной стратегии может быть представлена следующим образом. Выберем в множестве $D(t_m)$ случайное подмножество $D(q)$. Каждый элемент из $D(t_m)$ включается в $D(q)$ с вероятностью $q$ независимо от других элементов. Найдем в множестве $D(q)$ элемент с максимальным значением $\Delta_j$ и включим его в множество $D'$. Последовательно добавляя таким способом элементы и проверяя ресурсные ограничения, получим случайное множество $D'$. Более точно вероятностный алгоритм выбора множества $D'$, ориентированный на максимальные временные задержки (ВЗ), может быть представлен следующим образом.

Алгоритм ВЗ$(q, t_m, A(t_m), D(t_m))$

1. Положить $\overline{D} := D(t_m),\ D' := \emptyset, \widetilde{R}_k := R_k - \sum_{i \in A(t_m)} r_{ik},\ k \in K.$
2. Выбрать в $\overline{D}$ случайное подмножество $D(q)$.
   Если $D(q) = \emptyset$, то добавить в $D(q)$ любой элемент из $\overline{D}$.
3. Найти $i_0 \in D(q)$ с максимальной задержкой $\Delta_{i_0} = \max_{i \in D(q)} \Delta_i$.
4. Положить $D' := D' \bigcup \{i_0\};\ \overline{D} := \overline{D} \setminus \{i_0\},\ \widetilde{R}_k := \widetilde{R}_k - r_{i_0 k},\ k \in K.$
5. Для всех $i \in \overline{D}$: если $r_{ik} > \widetilde{R}_k$ для некоторого $k \in K$,
   то положить $\overline{D} := \overline{D} \setminus \{i\}$.
6. Если $\overline{D} \neq \emptyset$, то вернуться на шаг 2.

Результатом работы алгоритма является множество $D'$, которое зависит от параметра $q$. При $q = 1$ получаем детерминированный жадный алгоритм. Близкие по смыслу алгоритмы рассматривались в [58, 63]. Ниже будет показано, что параметр $q$ сильно влияет на результаты работы алгоритма. Даже для малой серии испытаний наилучшее найденное решение при $q < 1$ имеет меньшую погрешность, чем детерминированный аналог.

Заметим, что задержка работы $j$ могла быть оценена и другим способом,

например,

$$\Delta_j = t_m - LF_j + p_j,\ j \in D(t_m),$$

или

$$\Delta_j = t_m - s_j^0,\ j \in D(t_m),$$

где $s_j^0,\ j \in J$ — оптимальное решение задачи без учета ресурсных ограничений. Новые определения также вводят временные задержки, но не учитывают взаимовлияние элементов множества $D(t_m)$. Экспериментальные исследования показали, что определение задержек через величины $\overline{t}_j$ хоть и более трудоемко, но приводит к лучшим результатам.

### 2.2.2 Использование решения задачи на узкое место

Предположим, что известно оптимальное решение исходной задачи $s_j^*, j \in J$ и некоторое частичное расписание $s_j, j \in J(t_m) \cup A(t_m)$. Тогда для каждой работы $j \in D(t_m)$ можно вычислить задержку $a_j = t_m - s_j^*$, если работа будет включена в множество $D' \subseteq D(t_m)$ и задержку $b_j = \overline{t}_j - s_j^* \geq a_j$, если работа не будет включена в это множество. При известных задержках выбор множества $D'$ следует проводить так, чтобы максимальная задержка работ из множества $D(t_m)$ была бы минимальной, т. е., найти

$$\min_{x_j}\{\max_{j \in D(t_m)}(a_j x_j + b_j(1 - x_j))\}$$

при ограничениях

$$\sum_{j \in D(t_m)} r_{jk} x_j \leq R_k - \sum_{i \in A(t_m)} r_{ik},\ k \in K,$$

$$x_j \in \{0, 1\},\ j \in D(t_m).$$

Сформулированная задача является задачей на узкое место. Оптимальное значение целевой функции достигается на одном из значений $a_j$ или $b_j, j \in J$. Упорядочим множество $D(t_m)$ по невозрастанию значений $b_j$. Согласно этому порядку будем включать работы в множество $D'$. Как только очередная работа $j$ нарушает хотя бы одно ресурсное ограничение или для текущего множества $D'$ справедливо неравенство $b_j \geq \max_{i \in D'} a_i$, получаем оптимальное решение задачи. Однако на этом процесс пополнения

множества $D'$ не заканчивается. Несмотря на то, что оптимальное значение уже найдено, оставшиеся ресурсы могут допускать дальнейшее расширение множества $D'$. Поэтому просмотр работ продолжается и, если очередная работа удовлетворяет ресурсным ограничениям, то она также включается в множество $D'$. Проверка этого условия требует $O(|K|)$ операций, если последовательно пересчитывать остатки ресурсов. Таким образом, оптимальное решение задачи на узкое место может быть найдено за $O(n(|K| + \log_2 n))$ операций.

Положим $a_j = t_m - (LF_j - p_j), \quad b_j = \Delta_j = \overline{t}_j - (LF_j - p_j), j \in D(t_m)$, и пусть $x_j^*,\ j \in D(t_m)$ — оптимальное решение задачи на узкое место. Рассмотрим следующий вероятностный алгоритм решения задачи на узкое место (ВУМ), который сначала берет часть работ со значениями $x_j^* = 1$, а затем пополняет это множество случайным образом работами с $x_j^* = 0$.

Алгоритм ВУМ$(q, t_m, A(t_m), D(t_m))$

1. Решить задачу на узкое место.
2. Положить $D' = \emptyset, D_1 = \{j \in D(t_m) \mid x_j^* = 1\}, D_0 = \{j \in D(t_m) \mid x_j^* = 0\}$.
3. Каждый элемент из множества $D_1$ включить с вероятностью $q$ в множество $D'$.
4. Упорядочить работы из множества $D_0$ по невозрастанию величин $\sum_{k \in K} r_{jk}/R_k$.
5. Согласно данному порядку элементы из $D_0$ включать с вероятностью $q$ в множество $D'$ при соблюдении ресурсных ограничений.

Алгоритм ВУМ отдает предпочтение работам, вошедшим в оптимальное решение задачи на узкое место, и в этом смысле является жадным алгоритмом. Отметим, что в рассматриваемой задаче на узкое место как правило оптимальное решение не является единственным. Действительно, по определению $b_j \geq a_j, j \in D(t_m)$. Если минимум достигается на некотором значении $f^*$, то в оптимальном решении $x_j^* = 1$ при $b_j > f^*$ и $x_j^* = 0$ при $b_j = f^*$. Значения остальных переменных уже не влияют не оптимум. Другими словами, оптимальные решения могут сильно отличаться по числу выбранных работ. Из множества оптимальных решений целесообразно

выбрать такое решение, которое в максимальной степени использовало бы имеющиеся ресурсы. Для этих целей в алгоритме на шаге 5 используется множество $D_0$, из которого выбираются работы с большими требованиями по ресурсам. В следующем алгоритме идея максимальной утилизации ресурсов является доминирующей.

### 2.2.3 Использование решения задачи о многомерном рюкзаке

В первой главе при определении соседних решений для окрестности $N_3$ использовалось решение вспомогательной задачи о многомерном рюкзаке. Основная идея такого подхода заключалась в максимальном использовании имеющихся ресурсов в заданный момент времени. В данном разделе предлагается использовать ту же идею в качестве стратегии выбора в схеме жадного алгоритма для решения ЗКПОР.

В качестве множества $D'$ предлагается выбирать подмножество $\{j \in D(t_m) \mid x_j^* = 1\}$, где $x_j^*$, $j \in D(t_m)$ — решение задачи о многомерном рюкзаке, сформулированной в разделе 1.1.3. Для решения этой задачи используется алгоритм ВМР (см. раздел 1.1.4).

## 2.3 Локальные улучшения

Рассмотренные выше вероятностные жадные алгоритмы дают возможность порождать широкий спектр приближенных решений. Каждое из них может быть улучшено методами локальной перестройки. Для этой цели предлагается использовать стандартный алгоритм локального спуска по одной из трех окрестностей $N_1(S)$, $N_2(S)$, $N_3(S)$.

Как отмечалось выше, поиск соседнего решения с меньшим значением целевой функции требует значительных затрат машинного времени. Поэтому в качестве альтернативы локальному спуску рассмотрим так называемую *forward–backward* процедуру [63, 71, 82], которая последовательно переходит от активных расписаний к $T$-поздним и обратно до тех пор, пока это приводит к уменьшению целевой функции. В [8] эта процедура названа алгоритмом Пинг–Понг.

Пусть $S_0$ — активное расписание и $L_0$ соответствующий ему список. Построим список $L_1$, упорядочив работы по времени их окончания. К полу-

ченному списку применим процедуру построения $T$-позднего расписания, которая начиная с момента времени $T(S_0) = c_n(S_0)$ в обратном порядке вычисляет наиболее поздние времена окончания работ, не нарушая ограничений по ресурсам. Полученное расписание обозначим через $S_1$. Построим список $L_2$, упорядочив работы по началам их выполнения в $S_1$. По списку $L_2$ построим активное расписание $S_2$. Если расписание $S_2$ имеет меньшую длину, чем $S_0$, то положим $S_0 := S_2$ и повторим эту процедуру. Критерием остановки алгоритма является совпадение значений целевой функции для расписаний $S_0$ и $S_2$. Приведенный алгоритм не гарантирует получение локального оптимума по определенным выше окрестностям. Тем не менее, как мы увидим ниже, с большой вероятностью это событие имеет место для окрестностей $N_1, N_3$, а для окрестности $N_2$ число шагов до локального оптимума оказывается крайне малым.

## 2.4 Экспериментальные исследования

### 2.4.1 Сравнение жадных стратегий

Первый эксперимент посвящен сравнению стратегии выбора подмножества $D'$ на шаге 2.4 алгоритма МПР. Исследуются три детерминированные стратегии. Напомним, что первая стратегия основана на сортировке допустимого множества по временным задержкам $\Delta_j,\ j \in D(t_m)$. Вторая использует решение задачи на узкое место. Третья стратегия основана на решении задачи о многомерном рюкзаке. Результаты эксперимента приведены в таблице 2.1. В первой колонке указан идентификатор класса тестовых примеров. Во второй, третьей и четвертой колонках показаны средние относительные погрешности в процентах по отношению к наилучшему известному значению целевой функции для первой (ВЗ), второй (ВУМ) и третьей (ВМР) стратегии, соответственно. В этой и следующей таблицах приведены средние значения из 10 испытаний, по одному испытанию для каждого примера в каждом классе.

Первые две стратегии дают близкие результаты с небольшим перевесом в пользу решения задачи на узкое место. Последняя стратегия существенно им проигрывает. По-видимому, это объясняется тем, что решение задачи о многомерном рюкзаке учитывает только ограничения по ресурсам. Первые

же две стратегии используют отношения предшествования работ наряду с ресурсными ограничениями.

| Класс | МПР+ВЗ | МПР+ВУМ | МПР+ВМР |
|---|---|---|---|
| j30 13 | 10,27 | 9,51 | 15,92 |
| j30 29 | 12,10 | 9,60 | 16,46 |
| j30 45 | 9,82 | 8,84 | 14,06 |
| j60 13 | 13,67 | 12,85 | 19,57 |
| j60 29 | 12,68 | 13,02 | 19,12 |
| j60 45 | 13,37 | 12,02 | 18,74 |
| j120 16 | 10,97 | 11,26 | 16,30 |
| j120 36 | 13,43 | 10,94 | 17,45 |
| j120 56 | 14,72 | 12,79 | 16,14 |

Таблица 2.1: Средние относительные погрешности для разных стратегий

| Класс | МПР+ВЗ+ПП | МПР+ВУМ+ПП | МПР+ВМР+ПП |
|---|---|---|---|
| j30 13 | 9,18 / 1,5 (3) | 8,70 / 1,4 (2) | 11,41 / 2,1 (4) |
| j30 29 | 8,95 / 1,7 (2) | 7,89 / 1,6 (2) | 10,52 / 2 (3) |
| j30 45 | 7,19 / 1,5 (2) | 7,10 / 1,6 (2) | 10,10 / 1,8 (2) |
| j60 13 | 11,88 / 1,8 (2) | 9,52 / 2,4 (4) | 11,96 / 2,2 (3) |
| j60 29 | 10,94 / 1,9 (3) | 11,64 / 1,7 (2) | 13,96 / 2 (2) |
| j60 45 | 11,86 / 2,1 (3) | 9,99 / 2,1 (4) | 12,59 / 2,6 (4) |
| j120 16 | 9,71 / 2,2 (3) | 9,46 / 2,3 (3) | 11,85 / 2,5 (4) |
| j120 36 | 11,65 / 2,3 (3) | 9,76 / 2,2 (3) | 12,47 / 2,5 (4) |
| j120 56 | 12,55 / 2,7 (4) | 11,23 / 2,3 (5) | 13,25 / 2,3 (4) |

Таблица 2.2: Влияние алгоритма ПП

Второй эксперимент связан с исследованием влияния процедуры Пинг–Понг (ПП). Структура таблицы 2.2 аналогична структуре таблице 2.1. В ней приведены средние значения относительной погрешности, среднее и максимальное число шагов процедуры ПП. Напомним, что шаг состоит в построении двух расписаний, $T$-позднего и активного. Использование $T$-поздних расписаний позволяет заметно сократить относительную погрешность. Интересно отметить, что число шагов алгоритма ПП достаточно мало. В среднем оно составляет 1–2 шага, максимальное число не превосходит 5. Для всех классов время работы на PC Pentium 1200 Mhz трех вариантов алгоритма примерно одинаковое и в среднем составляет поряд-

ка 0,01 секунды для примеров с 30 работами и порядка 0,03 и 0,12 секунд для задач размерности 60 и 120 работ, соответственно.

### 2.4.2 Внесение рандомизации

Следующий эксперимент посвящен рандомизированным вариантам данных стратегий. Если в алгоритме МПР на шаге 2.4 осуществлять вероятностный выбор множества $D'$, то это позволяет получать несколько расписаний, а затем выбирать из них наилучшее.

На рисунках 2.1—2.3 показано изменение средней относительной погрешности в зависимости от параметра рандомизации $q$ и числа построенных расписаний $Z$. Величина $q$ менялась в пределах от 0,1 до 0,9. Объем выборки составлял 100 испытаний: 10 испытаний для каждого из 10 примеров в каждом классе. Рисунки показывают, что качественное поведение алгоритма мало меняется при смене стратегии. Однако, как и в детерминированном случае использование стратегии ВУМ оказывается предпочтительнее.

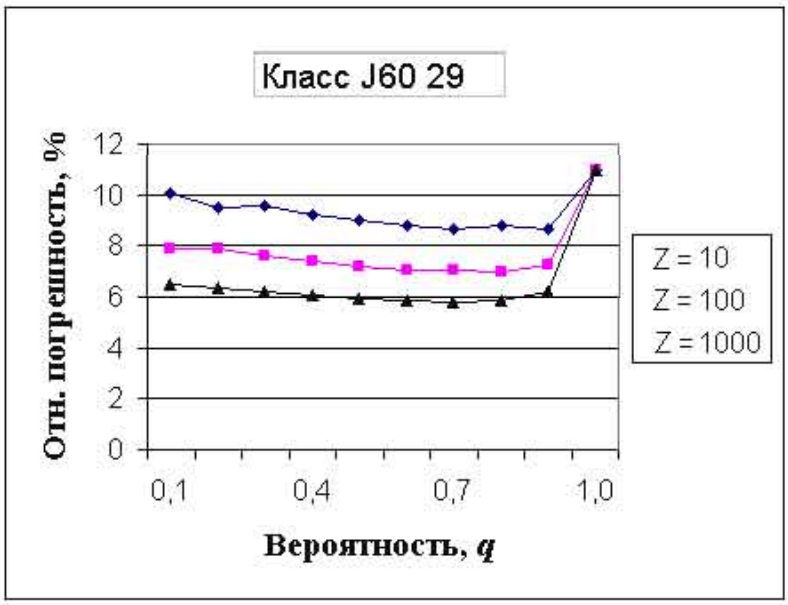


Рис. 2.1: Влияние рандомизации на погрешность МПР+ВЗ+ПП

На рисунках представлены результаты для класса j60 29. В остальных классах получаются аналогичные результаты. Они отличаются лишь абсолютными значениями погрешностей, но общая структура графиков сохраняется.

На рисунке 2.4 показано сравнение рандомизированных стратегий с лучшими значениями параметра $q$ для $Z = 10, 100, 1000$. Оптимальное значение $q$ составляет 0,6–0,9. Оно зависит от размерности задачи и падает с ростом $Z$. При фиксированном $Z$ увеличение размерности приводит к росту $q$. Рисунок 2.4 наглядно свидетельствует о преимуществе второй стратегии. Именно она будет использоваться в дальнейших экспериментах.

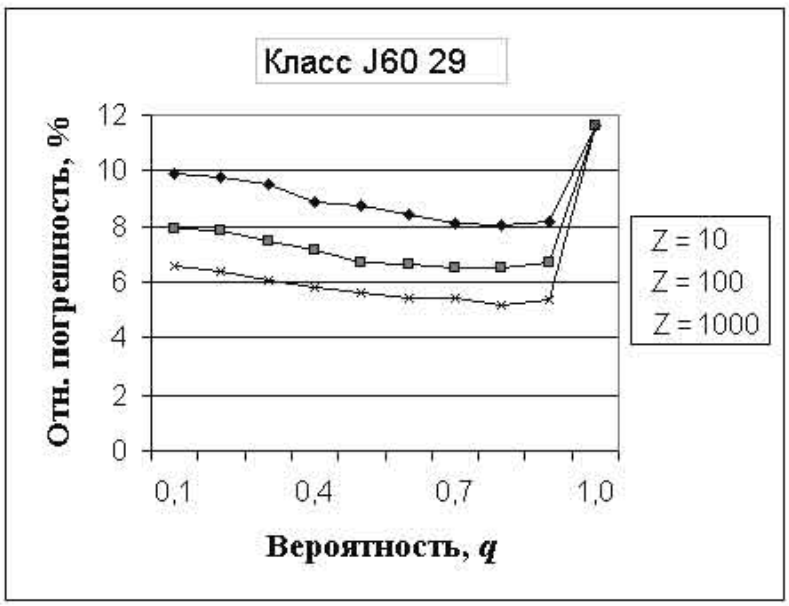


Рис. 2.2: Влияние рандомизации на погрешность МПР+ВУМ+ПП

### 2.4.3 Локальный спуск

Следующий эксперимент связан с исследованием влияния локального спуска на качество получаемых решений. После работы алгоритма ПП применяется стандартная процедура поиска локального оптимума по одной из трех окрестностей, описанных в первой главе диссертации. На рис. 2.5 показана средняя относительная погрешность локальных оптимумов в зависимости от $Z$ для каждой из окрестностей $N_1$, $N_2$, $N_3$. Эксперимент проводился на классе j60 29 при $q = 0,7$.

С ростом $Z$ относительная погрешность падает. Вторая окрестность приводит к лучшим локальным оптимумам, что, по-видимому, обусловлено ее квадратичной мощностью. Однако, ее просмотр требует больших временных затрат. Интересно отметить, что относительная погрешность для второй окрестности почти совпадает с погрешностью для третьей окрестности

при 10-ти кратном увеличении $Z$. Поскольку просмотр окрестности $N_2$ требует почти в $n$ раз больше времени, чем просмотр окрестности $N_3$, несмотря на многократное решение задачи о многомерном рюкзаке, то использование третьей окрестности более эффективно. Преимущество третьей окрестности по сравнению с первой представляется очевидным.

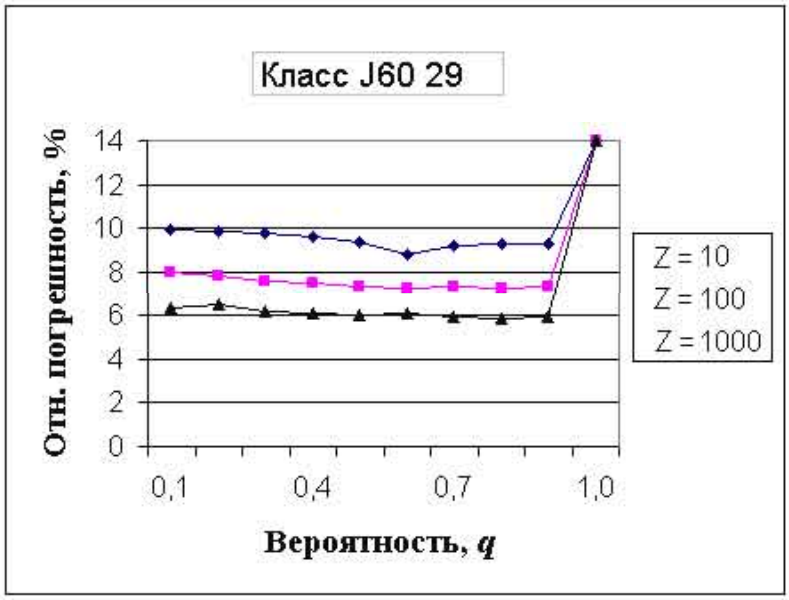


Рис. 2.3: Влияние рандомизации на погрешность МПР+ВМР+ПП

В следующем эксперименте проводилось сравнение окрестностей по средней относительной погрешности получаемых локальных оптимумов и числу шагов для завершения локального спуска. Результаты эксперимента приведены в таблице 2.3. Значение $Z$ выбрано равным 100 и соответствует ста расписаниям, к каждому из которых последовательно применялся алгоритм ПП и стандартная процедура локального спуска. В качестве жадного алгоритма использовался алгоритм ВУМ при $0,7 \leq q \leq 0,9$.

Заметим, что число шагов, которое в таблице 2.3 обозначено через *Depth* в среднем не превосходит единицы для линейных окрестностей и составляет 2–3 шага для квадратичной окрестности. Другими словами, в большинстве случаев результат работы алгоритма ПП либо является локальным оптимумом, либо лежит в одном шаге от него для окрестностей $N_1, N_3$. Для окрестности $N_2$ требуется чуть больше шагов, но это число также очень мало. Картина меняется, если в качестве начальной точки выбрано случайное расписание. В этом случае средняя погрешность получаемых

локальных оптимумов и число шагов оказываются почти в два раза выше.

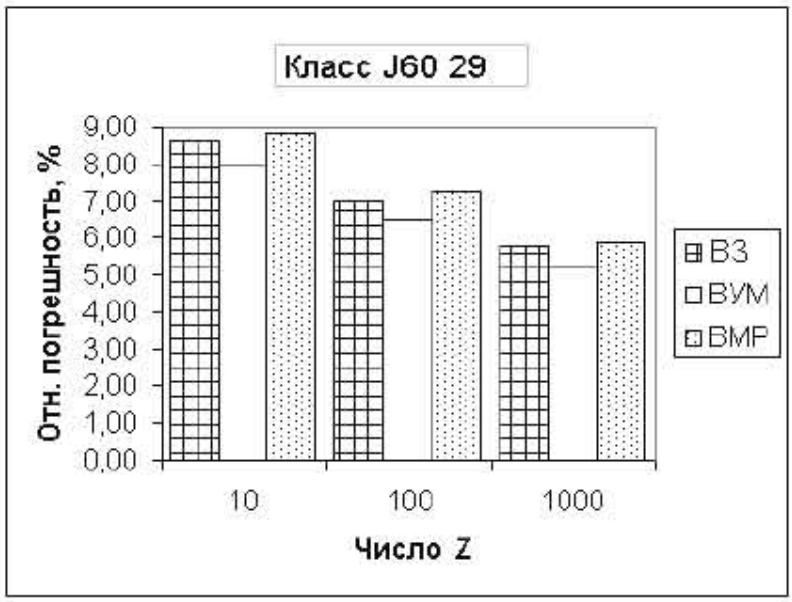


Рис. 2.4: Сравнение вероятностных стратегий

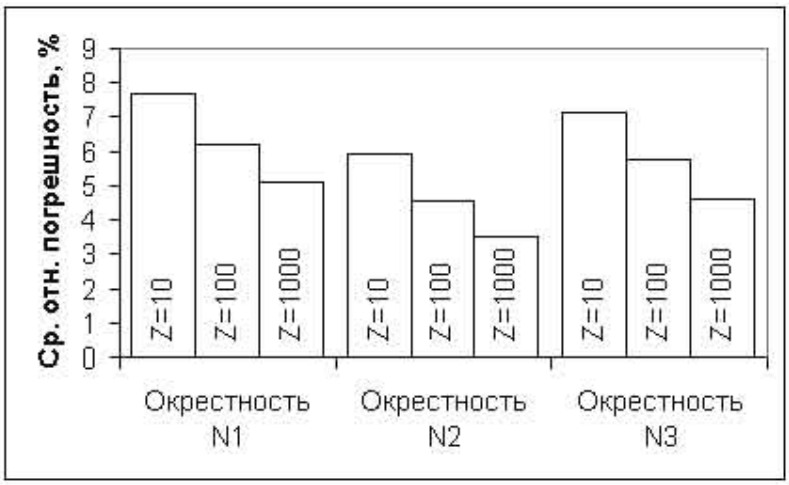


Рис. 2.5: Относительная погрешность локальных оптимумов

В таблице 2.4 приведены доверительные интервалы для вероятности $P_0$ получения алгоритмом ПП локального оптимума по каждой из рассматриваемых окрестностей. Результаты расчетов показывают, что для первой и третьей окрестностей получаемые решения довольно часто оказываются локально оптимальными. Для второй окрестности это событие маловероятно. Она имеет бóльшую мощность и дает больше возможностей найти лучшее соседнее решение.

| Класс | ПП $\varepsilon$, % | Окр. $\varepsilon$, % | $N_1$ *Depth* | Окр. $\varepsilon$, % | $N_2$ *Depth* | Окр. $\varepsilon$, % | $N_3$ *Depth* |
|---|---|---|---|---|---|---|---|
| j30 13 | 3,69 | 3,49 | 0,44 | 1,23 | 2,29 | 2,81 | 0,56 |
| j30 29 | 3,31 | 2,66 | 0,55 | 0,88 | 2,38 | 2,49 | 0,82 |
| j30 45 | 1,95 | 1,42 | 0,92 | 0,18 | 2,38 | 1,33 | 0,70 |
| j60 13 | 6,25 | 6,13 | 0,28 | 4,81 | 1,8 | 5,64 | 0,88 |
| j60 29 | 6,55 | 6,22 | 0,9 | 4,56 | 2,2 | 5,78 | 0,90 |
| j60 45 | 6,33 | 6,02 | 0,69 | 3,84 | 2,54 | 5,24 | 1,07 |
| j120 16 | 6,57 | 6,50 | 0,44 | 5,47 | 2,09 | 6,16 | 0,99 |
| j120 36 | 7,31 | 7,10 | 0,42 | 5,62 | 2,98 | 6,71 | 1,38 |
| j120 56 | 8,40 | 7,86 | 0,95 | 6,43 | 2,48 | 7,55 | 1,31 |

Таблица 2.3: Погрешность локальных оптимумов и глубина спуска

В таблице 2.5 приведены доверительные интервалы для вероятности $P_1$ получения решения, содержащего локальный оптимум в своей окрестности. Уровень доверия в обоих случаях составлял 0,95. Величина выборки — сто испытаний.

| Класс | $N_1$ | $N_2$ | $N_3$ |
|---|---|---|---|
| j30 13 | ( 0,72 ; 0,77 ) | ( 0,12 ; 0,16 ) | ( 0,55 ; 0,61 ) |
| j60 29 | ( 0,63 ; 0,69 ) | ( 0,10 ; 0,14 ) | ( 0,40 ; 0,47 ) |
| j120 36 | ( 0,60 ; 0,66 ) | ( 0,05 ; 0,08 ) | ( 0,32 ; 0,38 ) |

Таблица 2.4: Доверительные интервалы для величины $P_0$

| Класс | $N_1$ | $N_2$ | $N_3$ |
|---|---|---|---|
| j30 13 | ( 0,93 ; 0,96 ) | ( 0,38 ; 0,44 ) | ( 0,84 ; 0,88 ) |
| j60 29 | ( 0,88 ; 0,92 ) | ( 0,32 ; 0,40 ) | ( 0,72 ; 0,78 ) |
| j120 36 | ( 0,86 ; 0,90 ) | ( 0,19 ; 0,24 ) | ( 0,62 ; 0,68 ) |

Таблица 2.5: Доверительные интервалы для величины $P_1$

Представленные результаты касались локальной оптимальности получаемых решений в среднем. Однако вероятностные жадные алгоритмы строят несколько расписаний (Z=10, 100, 1000) и лучшее из них предъявляют в качестве ответа. Интересно знать, с какой вероятностью такое решение оказывается локальным оптимумом. Результаты расчетов представлены в таблице 2.6.

| Класс | $N_1$ | $N_2$ | $N_3$ |
|---|---|---|---|
| j30 13 | (0,91; 0,99) | (0,59; 0,77) | (0,88; 0,98) |
| j60 29 | (0,88; 0,98) | (0,47; 0,67) | (0,78; 0,92) |
| j120 36 | (0,78; 0,92) | (0,26; 0,44) | (0,58; 0,76) |

Таблица 2.6: Доверительные интервалы для величины $P_0$ при $Z = 1000$

Для линейных окрестностей $N_1, N_3$ данная вероятность достаточно высока и медленно падает с ростом размерности. Для окрестности $N_2$ она составляет в среднем около 0,5. Локальный спуск по такой окрестности скорее всего позволит сократить погрешность, но потребует большого числа подсчетов целевой функции. Для того, чтобы *уравнять в правах* рассматриваемые окрестности, зафиксируем число подсчетов целевой функции, разрешив каждому из алгоритмов обращаться к этой процедуре одинаковое число раз, например, 1000. Тогда число $Z$ для каждой окрестности станет разным, линейные окрестности получат больше локальных оптимумов, сравнение алгоритмов станет более адекватным с точки зрения трудоемкости.

| Класс | ПП | ПП+ЛС(1) | | | ПП+ЛС | | |
|---|---|---|---|---|---|---|---|
| | | $N_1$ | $N_2$ | $N_3$ | $N_1$ | $N_2$ | $N_3$ |
| j30 13 | 3,13 | 3,92 | 5,98 | 3,87 | 3,92 | 4,84 | 3,90 |
| j60 29 | 5,84 | 6,67 | 9,51 | 7,01 | 6,64 | 9,75 | 7,25 |
| j120 36 | 6,80 | 7,75 | 9,72 | 8,25 | 7,73 | 9,79 | 8,51 |

Таблица 2.7: Средняя относительная погрешность для трех вариантов алгоритма

Предшествующие расчеты подсказывают, что с высокой вероятностью получаемые решения могут оказаться локальными оптимумами, и усилия по просмотру окрестности будут напрасными. В связи с этим интересно сравнить три варианта алгоритма:

— без локального спуска после работы алгоритма ПП;

— с одним шагом локального спуска;

— с локальным спуском до получения локального оптимума.

Результаты расчетов приведены в таблице 2.7. Они свидетельствуют, что при ограничении числа подсчетов целевой функции в одну тысячу применение методов локального спуска нецелесообразно. Просмотр окрестностей

$N_1, N_3$ и даже $N_2$ слишком часто оказывается безрезультатным. Лучшее решение легче получить, начав всю процедуру заново, а не пытаться улучшить уже имеющееся решение. Заметим, что ситуация может измениться, если число вычислений целевой функции значительно возрастет. В этом случае дальнейшее сокращение погрешности может потребовать слишком больших значений $Z$, и любые приемы, в том числе и локальный спуск, могут стать оправданными.

### 2.4.4 Сравнение с другими алгоритмами

Последний эксперимент посвящен сравнению с известными приближенными алгоритмами для решения ЗКПОР. Результаты сравнения приведены в таблицах 2.9—2.10. Следуя [63], число вычислений целевой функции ограничивалось величинами 1000, 5000 и 50000. Средняя относительная погрешность приводится для алгоритма МПР, в котором используется процедура ВУМ с последующим улучшением алгоритмом ПП.

Результаты сравнения показывают, что в задачах малой размерности разработанный алгоритм хотя и уступает некоторым известным алгоритмам аналогичного класса, но с увеличением размерности задачи его выигрыш растет. В задачах с числом работ 60 он проигрывает только одному из жадных алгоритмов. В таблицах 8—10 этот алгоритм помечен знаком (*) и приводится без указания ссылки, поскольку на данный момент он не опубликован. Его авторы P. Tormos и A. Lova. В задачах с числом работ 120 разработанный алгоритм превосходит другие алгоритмы, если критерий остановки ограничивает общее количество построенных расписаний числом 1000 или 5000.

Следует отметить, что метаэвристики, такие как генетические алгоритмы, поиск с запретами, имитация отжига и т.п. не всегда выигрывают у жадных алгоритмов. В основном это происходит в тех случаях, когда критерий остановки подразумевает большое число вычислений целевой функции. Если же это число предполагается небольшим, то преимущество, как правило, оказывается у жадных алгоритмов. В этом смысле удачный выбор алгоритма для решения конкретной задачи обусловлен временным фактором.

| Алгоритм | 1000 | 5000 | 50000 |
|---|---|---|---|
| Жадный алгоритм* | 0,25 | 0,13 | 0,05 |
| Жадный алгоритм [100] | 0,30 | 0,16 | 0,07 |
| Жадный алгоритм [101] | 0,30 | 0,17 | 0,09 |
| МПР+ВУМ+ПП | 0,45 | 0,35 | 0,25 |
| Жадный алгоритм [105] | 0,46 | 0,28 | 0,11 |
| Жадный адаптивный алгоритм [94] | 0,65 | 0,44 | — |
| Жадный адаптивный алгоритм [61] | 0,74 | 0,52 | — |
| Жадный алгоритм [60] | 1,40 | 1,29 | 1,13 |
| Жадный алгоритм [59] | 1,77 | 1,48 | 1,22 |
| Генетический алгоритм [16] | 0,33 | 0,12 | — |
| Генетический алгоритм [51] | 0,38 | 0,22 | 0,08 |
| Имитация отжига [28] | 0,38 | 0,23 | — |
| Поиск с запретами [77] | 0,46 | 0,16 | 0,05 |
| Генетический алгоритм [32] | 0,74 | 0,33 | 0,16 |
| Поиск с запретами [20] | 0,86 | 0,44 | — |

Таблица 2.8: Сравнение алгоритмов, $n = 30$

| Алгоритм | 1000 | 5000 | 50000 |
|---|---|---|---|
| Жадный алгоритм* | 11,88 | 11,62 | 11,36 |
| МПР+ВУМ+ПП | 12,10 | 11,98 | 11,84 |
| Жадный алгоритм [101] | 12,14 | 11,82 | 11,47 |
| Жадный алгоритм [100] | 12,18 | 11,87 | 11,54 |
| Жадный алгоритм [105] | 12,73 | 12,35 | 11,94 |
| Жадный адаптивный алгоритм [94] | 12,94 | 12,58 | — |
| Жадный адаптивный алгоритм [61] | 13,51 | 13,06 | — |
| Жадный алгоритм [60] | 13,59 | 13,23 | 12,85 |
| Жадный алгоритм [59] | 14,89 | 14,30 | 13,66 |
| Генетический алгоритм [51] | 12,21 | 11,70 | 11,21 |
| Генетический алгоритм [16] | 12,57 | 11,86 | — |
| Имитация отжига [28] | 12,75 | 11,90 | — |
| Поиск с запретами [77] | 12,97 | 12,18 | 11,58 |
| Генетический алгоритм [32] | 13,28 | 12,63 | 11,94 |
| Поиск с запретами [20] | 13,80 | 13,48 | — |

Таблица 2.9: Сравнение алгоритмов, $n = 60$

| Алгоритм | 1000 | 5000 | 50000 |
|---|---|---|---|
| МПР+ВУМ+ПП | 34,92 | 34,40 | 33,85 |
| Жадный алгоритм* | 35,01 | 34,41 | 33,71 |
| Жадный алгоритм [101] | 36,24 | 35,56 | 34,77 |
| Жадный алгоритм [100] | 36,49 | 35,81 | 35,01 |
| Жадный алгоритм [105] | 38,21 | 37,47 | 36,46 |
| Жадный алгоритм [60] | 39,60 | 38,75 | 37,74 |
| Жадный адаптивный алгоритм [94] | 39,85 | 38,70 | — |
| Жадный адаптивный алгоритм [61] | 41,37 | 40,45 | — |
| Жадный алгоритм [59] | 44,46 | 43,05 | 41,44 |
| Генетический алгоритм [51] | 37,19 | 35,39 | 33,21 |
| Муравьиные колонии [74] | — | 35,43 | — |
| Генетический алгоритм [16] | 39,36 | 36,57 | — |
| Генетический алгоритм [32] | 39,97 | 38,41 | 36,44 |
| Поиск с запретами [77] | 40,86 | 37,88 | 35,85 |

Таблица 2.10: Сравнение алгоритмов, $n = 120$

# Глава 3

# Поиск с запретами и чередованием окрестностей

В этой главе предлагается новый алгоритм локального поиска, основанный на идее чередующихся окрестностей [48]. В алгоритме используется два типа окрестностей. Первая строится по активным расписаниям, то есть таким расписаниям, когда ни одна работа не может быть начата раньше указанного ей срока без нарушения либо условий предшествования, либо ограничений по ресурсам. Вторая окрестность строится по $T$-поздним расписаниям. Такие расписания рассматривались в работе [1] для задач со складируемыми ресурсами и использовались для получения точных и ассимптотически точных решений. В некотором смысле $T$-поздние расписания являются аналогом активных расписаний с той лишь разницей, что теперь каждая работа не может быть завершена позднее указанного ей срока без нарушения условий предшестования, ограничений по ресурсам или увеличения времени завершения проекта. Активные и $T$-поздние расписания удачно дополняют друг друга и переход от одной окрестности к другой привносит определенное разнообразие в локальный поиск, что благотворно сказывается на результатах работы алгоритма.

## 3.1 Окрестности

Рассмотрим две окрестности, $N_A(S)$ и $N_T(S)$. Первая из них строится для активных расписаний, вторая — для $T$-поздних. Каждая окрестность подразумевает просмотр линейного числа соседних решений, при построении которых используется идея параллельного декодера.

В качестве окрестности $N_A(S)$ выступает окрестность $N_3$, описанная в главе 1. Окрестность $N_T(S)$ является в некотором смысле ее симметричным аналогом и использует $T$-поздний декодер для определения соседних решений.

Пусть $S$ является $T$-поздним расписанием и блок работы $j$ не содержит ее последователей, то есть $B_j \cap P_i = \emptyset, (j,i) \in C$. По аналогии с определением окрестности $N_A(S)$, выделим сегмент $L(j)$ в $L$. Началом сегмента является минимальная позиция в списке $L$ для работ блока $B_j$ и входящей сети. Концом сегмента является максимальная позиция в $L$ работ блока $B_j$.

Пусть, как и прежде, список $L$ состоит из трех частей: $L', L(j), L''$. Сохраним старые сроки выполнения для работ списка $L''$. Применим к нему модификацию параллельного декодера, которая в обратном порядке вычисляет времена выполнения работ из сегмента $L_j$, используя задачу о многомерном рюкзаке. Для оставшихся работ из $L'$ моменты выполнения определяются $T$-поздним декодером. Полученное расписание назовем соседним для $S$ и обозначим его через $S_T(j)$. Множество всех таких расписаний назовем окрестностью $N_T(S)$.

## 3.2 Общая схема

Разработанный алгоритм сочетает в себе идеи двух методов: локальный поиск с чередующимися окрестностями [48] и поиск с запретами [46]. За основу принята схема поиска с запретами, в которой систематически осуществляется переход от окрестности $N_A(S)$ к $N_T(S)$ и наоборот.

### 3.2.1 Построение начального решения

Хорошее начальное приближение позволяет выбрать перспективную область и сконцентрировать усилия на улучшении уже достаточно хорошего решения. Выбор начального приближения не является критическим для методов локального поиска, но неудачный выбор может потребовать неоправданных затрат на исправление ситуации.

Среди допустимых списков выберем список $L_0$, в котором для любой

соседней пары работ $(j_m, j_{m+1})$ если $j_m \notin P_{j_{m+1}}$, то

$$\sum_{k \in K} r_{j_m k}/R_k \geq \sum_{k \in K} r_{j_{m+1} k}/R_k.$$

Таких списков много. Один из них можно построить, например, упорядочив работы по рангам в сети, порожденной условиями предшествования, а работы с равными рангами — по весу.

Применим к списку $L_0$ рандомизированный вариант параллельного декодера. На шаге 1.3 вместо множества $D(t_m)$ будем использовать его непустое подмножество, выбранное случайным образом. В результате получим некоторое расписание $S_0$. Далее будут применяться $T$-поздний и последовательный декодеры до тех пор, пока это приводит к убыванию целевой функции.

Положим $T = T(S_0)$ и по расписанию $S_0$ построим список $L_1$, упорядочив работы по времени их окончания, $c_{j_m} \leq c_{j_{m+1}}, m = 0, \ldots, n$. К полученному списку применим $T$-поздний декодер. Получим расписание $S_1$. Построим список $L_2$, упорядочив работы по началу их выполнения, $s_{j_m} \leq s_{j_{m+1}}, m = 0, \ldots, n$ и применим последовательный декодер. Получим расписание $S_2$. Если $T > T(S_2)$, то повторим процедуру с $T$-поздним и последовательным декодерами. Неформально, эта процедура напоминает игру в пинг-понг. Случайным образом формируется начальное расписание. Затем идет обмен между активными и $T$-поздними расписаниями до тех пор, пока удается сокращать время выполнения всех работ. Формально этот алгоритм может быть представлен следующим образом.

Алгоритм Пинг-понг

1. Построить начальный список и применить к нему рандомизированный вариант параллельного декодера.
2. Положить $T := s_{n+1}$.
3. Упорядочить работы так, чтобы $c_{j_m} \leq c_{j_{m+1}}, m = 0, \ldots, n$ и построить $T$-позднее расписание.
4. Упорядочить работы так, чтобы $s_{j_m} \leq s_{j_{m+1}}, m = 0, \ldots, n$ и построить активное расписание.
5. Если $T > s_{n+1}$, то положить $T := s_{n+1}$ и вернуться на шаг 3.

Каждый шаг этой процедуры, очевидно, является полиномиальным по числу выполняемых операций. Однако число возвращений на шаг 3 может оказаться большим. Оценить его сверху через полином от длины записи исходных данных не представляется возможным. По-видимому, этот вопрос тесно связан с другим, более широким вопросом о сложности поиска локальных оптимумов для трудных комбинаторных задач [109]. Тем не менее, численные эксперименты показывают малое число возвращений на шаг 3. Как правило, это число значительно меньше $n$.

### 3.2.2 Вероятностный поиск с запретами

Приведем общую схему вероятностного алгоритма поиска с запретами. На каждом шаге этой процедуры имеется некоторое расписание $S$ и значение функции $f(S) = \sum_{j \in J} s_j$ от расписаний, полученных на последних $h$ шагах алгоритма. Набор таких значений будем называть списком запретов. Шаг алгоритма состоит в переходе от расписания $S$ к соседнему расписанию $S'$ по окрестности $N_A(S)$ или $N_T(S)$. Для окрестности формируется случайным образом ее непустое подмножество $N'_A(S)$ или $N'_T(S)$, из которого выбирается расписание с минимальным значением целевой функции. Подмножество $N'_A(S)$ $(N'_T(S))$ формируется следующим образом. Из окрестности $N_A(S)$ $(N_T(S))$ удаляются все расписания, для которых значение функции $f(S)$ совпадает с одним из значений в списке запретов. Затем каждое из оставшихся расписаний включается в множество $N'_A(S)$ $(N'_T(S))$ с некоторой вероятностью $q$. Если длина списка запретов велика, то в результате удаления из окрестности "запрещенных"элементов, она может оказаться пустой. В этом случае значение длина списка запретов уменьшается. Если подмножество $N'_A(S)$ $(N'_T(S))$ оказалось пустым, то в него добавляется произвольный незапрещенный элемент из окрестности $N_A(S)$ $(N_T(S))$.

Алгоритм поиска с запретами

1. Построить начальное расписание $S$,
   положить $T^* := T(S)$, $S^* := S$.
2. Пока не выполнен критерий остановки выполнять следующее:

2.1 Выбрать окрестность.

2.2 Найти соседнее расписание $S'$.

2.3 Если $T(S') < T^*$, то положить $T^* := T(S'),\ S^* := S'$.

2.4 Обновить список запретов и положить $S := S'$.

Значение $T^*$ является результатом работы алгоритма. Оно соответствует наилучшему расписанию $S^*$, найденному в ходе работы алгоритма. В качестве критерия остановки используется либо максимальное число итераций, либо требуемое отклонение от заданной верхней или нижней оценки целевой функции. Смена окрестности производится через заданное число итераций. Коэффициент рандомизации $q$ в ходе поиска не меняется.

Приведенная схема алгоритма является одной из наиболее простых и может быть дополнена правилами интенсификации и диверсификации поиска [46]. Если значение $T^*$ не меняется на протяжении большого числа итераций, то целесообразно вернуться к расписанию $S^*$ и более тщательно исследовать эту часть допустимой области, либо выбрать новое начальное расписание.

Отметим, что список запретов кроме расписаний, полученных на последних итерациях, запрещает и многие другие расписания. В работах [20, 77, 98] рассматривались другие списки запретов. Они соответствовали окрестностям, полученным сдвигом одной или нескольких работ в списке $L$. Список запретов препятствовал возвращению работ на старое место и тем самым предотвращал зацикливание алгоритма. Однако, как отмечалось выше, одному расписанию может соответствовать несколько списков и, как следствие, расписание может остаться прежним в результате сдвига нескольких работ. Функция $f(S)$ устраняет этот недостаток, хотя и возникает опасность запрета всех соседних расписаний особенно на задачах малой размерности. В связи с этим величина $h$ не должна быть слишком большой и ее изменение должно тщательно контролироваться (см., например, [4, 22]).

## 3.3 Экспериментальные исследования

### 3.3.1 Эффект чередующихся окрестностей

Для методов локального поиска выбор окрестности играет решающую роль. Может показаться, что чем шире окрестность, тем эффективнее поиск: меньше локальных оптимумов, шире область притяжения, лучше значения локальных оптимумов. Конечно, для более широкой окрестности тратится больше времени на ее просмотр, но если на это обстоятельство не обращать внимания, то результаты поиска должны быть лучше. Однако, это не всегда так. В частности, экспоненциально большие окрестности [14] с их бесспорным достоинством быстро находить оптимальное решение в экспоненциальном множестве соседних решений не имеют значительного превосходства. Широкая область притяжения препятствует переходу от одного локального оптимума к другому. Эффективность методов падает, так как поиск концентрируется в малой части допустимой области.

Для преодоления этой трудности в работе [48] была предложена простая, но эффективная стратегия. Она использует не одну большую окрестность, а несколько окрестностей разной структуры и мощности. Переход от одной окрестности к другой меняет ландшафт, области притяжения локальных оптимумов, привносит элемент диверсификации без кардинальной смены области поиска.

Первый численный эксперимент связан с влиянием чередования окрестностей на эффективность предложенного метода. Таблица 3.1 содержит среднюю относительную погрешность $\varepsilon$ на наиболее трудных классах тестовых задач с числом работ 30, 60 и 120 для трех вариантов поиска:

- только по окрестности $N_A(S)$,
- только по окрестности $N_T(S)$,
- с чередованием окрестностей $N_A(S)$ и $N_T(S)$.

При $n$=30 погрешность считалась относительно точного решения. При $n$>30 для подсчета погрешности вместо точного решения использовалось наилучшее известное решение, опубликованное в библиотеке PSPLib.

Чередование окрестностей положительно сказывается на результатах поиска. Интересно отметить, что расширение окрестности до объединения

$N_A(S) \cup N_T(S)$ не лучше их чередования. Последняя колонка в таблице 3.1 свидетельствует, что объединение окрестностей не всегда приводит к сокращению погрешности. Диаграмма 3.1 показывает влияние интервала чередования окрестностей. При больших интервалах ($\tau > 1000$) по сути получаем последовательный поиск по окрестностям $N_A(S)$ и $N_T(S)$. Оптимальный интервал чередования соответствует 5-10 шагам алгоритма.

| Класс | $N_A$ | $N_T$ | $N_A,\ N_T$ | $N_A \cup N_T$ |
|---|---|---|---|---|
| j3013 | 0,65 | 0,67 | 0,07 | 0,00 |
| j3029 | 0,87 | 0,45 | 0,11 | 0,13 |
| j3045 | 0,35 | 0,53 | 0,08 | 0,00 |
| j6013 | 1,60 | 1,30 | 0,83 | 0,41 |
| j6029 | 1,69 | 1,43 | 0,81 | 0,85 |
| j6045 | 0,85 | 0,82 | 0,27 | 0,66 |
| j12016 | 2,15 | 1,99 | 1,42 | 0,71 |
| j12036 | 1,97 | 1,74 | 1,13 | 0,91 |
| j12056 | 1,91 | 1,85 | 1,03 | 1,16 |

Таблица 3.1: Изменение $\varepsilon(\%)$ при различном выборе окрестности


Рис. 3.1: Влияние интервала чередования окрестностей, $n$=60

Для каждого из трех вариантов алгоритма помимо относительной погрешности была подсчитана частота нахождения лучшего известного решения.

Частота представляет собой статистическую оценку вероятности получения лучшего решения. Для данной оценки построены доверительные интервалы [2]. Рассмотрим схему Бернулли с вероятностью успеха $p$. Под

| Класс | $N_A$ | $N_T$ | $N_A,\ N_T$ |
|---|---|---|---|
| j3013 | [ 0,63 ; 0,81 ] | [ 0,56 ; 0,74 ] | [ 0,87 ; 0,97 ] |
| j3029 | [ 0,39 ; 0,59 ] | [ 0,60 ; 0,78 ] | [ 0,82 ; 0,94 ] |
| j3045 | [ 0,68 ; 0,84 ] | [ 0,64 ; 0,72 ] | [ 0,87 ; 0,97 ] |
| j6013 | [ 0,00 ; 0,08 ] | [ 0,05 ; 0,17 ] | [ 0,19 ; 0,37 ] |
| j6029 | [ 0,01 ; 0,09 ] | [ 0,06 ; 0,18 ] | [ 0,15 ; 0,31 ] |
| j6045 | [ 0,20 ; 0,38 ] | [ 0,26 ; 0,44 ] | [ 0,41 ; 0,61 ] |
| j12016 | [ 0,00 ; 0,00 ] | [ 0,00 ; 0,00 ] | [ 0,00 ; 0,05 ] |
| j12036 | [ 0,00 ; 0,03 ] | [ 0,00 ; 0,00 ] | [ 0,01 ; 0,11 ] |
| j12056 | [ 0,00 ; 0,00 ] | [ 0,00 ; 0,03 ] | [ 0,00 ; 0,08 ] |

Таблица 3.2: Доверительные интервалы для частоты получения лучшего решения, $\alpha = 0,05,\ N = 100$

успехом будем понимать случайное событие, состоящее в том, что алгоритм нашел наилучшее известное решение. Рассмотрим серию из $N$ испытаний и набор булевых переменных $x_i,\ i = 1, \ldots, N$. Полагаем $x_i = 1$, если в $i$-м испытании алгоритм нашел лучшее решение и $x_i = 0$ в противном случае. Тогда случайная величина $p^* = \overline{X} = \sum x_i/N$ есть частота получения алгоритмом лучшего решения и в то же время оценка для параметра $p$ в рассмотренной схеме Бернулли. Заметим, что величина $\frac{\sqrt{N}(\overline{X}-p)}{\sqrt{\overline{X}(1-\overline{X})}}$ слабо сходится к стандартному нормальному закону. Отсюда доверительный интервал для параметра $p^* = \overline{X}$ имеет вид

$$\overline{X} - \frac{\tau_{1-\alpha/2}\sqrt{\overline{X}(1-\overline{X})}}{\sqrt{N}} \le p \le \overline{X} + \frac{\tau_{1-\alpha/2}\sqrt{\overline{X}(1-\overline{X})}}{\sqrt{N}},$$

где $\tau_{1-\alpha/2}$ – квантиль стандартного нормального распределения уровня $1 - \alpha/2$. При $\alpha = 0,05$ значение квантили составляет 1,96. Значения доверительных интервалов приведены в таблице 3.2. Из таблицы видно, что в большинстве рассмотренных классов доверительные интервалы, соответствующие алгоритму с одной и чередующимися окрестностями не пересекаются.

### 3.3.2 Размер окрестности и ее рандомизация

Следующий численный эксперимент посвящен выбору коэффициента $q$ рандомизации окрестности. При малых $q$ время просмотра окрестности сокращается, но появляется опасность пропустить оптимальное решение даже

находясь в его окрестности. При $q$=1 просматриваются все соседние решения, но так как отбор лучшего решения в окрестности ведется по целевой функции, а она меняется незначительно, то, как правило, всегда находятся соседние решения с нехудшим значением функционала. По сути эти решения мало отличаются от текущего, идет перестановка некритических работ и эффективность поиска падает. "Золотая середина"в данном случае соответствует $q \approx 0,2$ (см. рис.3.2). Близкие значения рандомизации наблюдались и для задач размещения [4]. По-видимому, положительный эффект рандомизации при локальном поиске имеет место и для других трудных комбинаторных задач.

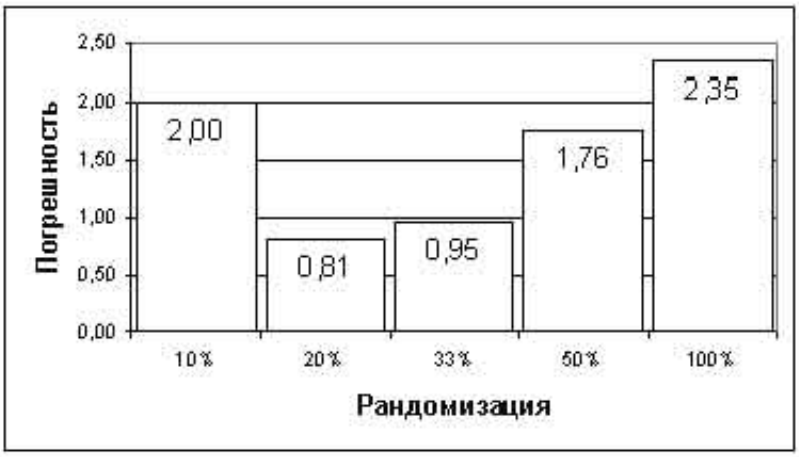


Рис. 3.2: Влияние рандомизации, класс j6029

Возвращаясь к вопросу об оптимальной мощности окрестности, был проведен еще один эксперимент. При определении соседних решений рассматривалась задача о многомерном рюкзаке и для нее вместо точного алгоритма применялся вероятностный жадный алгоритм. Его достоинство состоит не только в том, что он позволяет быстро получать приближенные решения задачи. Не менее важным является возможность получать различные приближенные решения. Ее можно использовать по-разному. Например, многократно применять алгоритм и запоминать лучшее из полученных решений. Таким образом можно повысить точность за счет увеличения трудоемкости. Правда, ниоткуда не следует, что чем точнее решение, тем меньше окажется длина расписания. Поэтому логично рассматривать не только наилучшее решение для задачи о многомерном рюкзаке, но и другие, близкие к нему решения. В ходе численного эксперимента варьировалось число

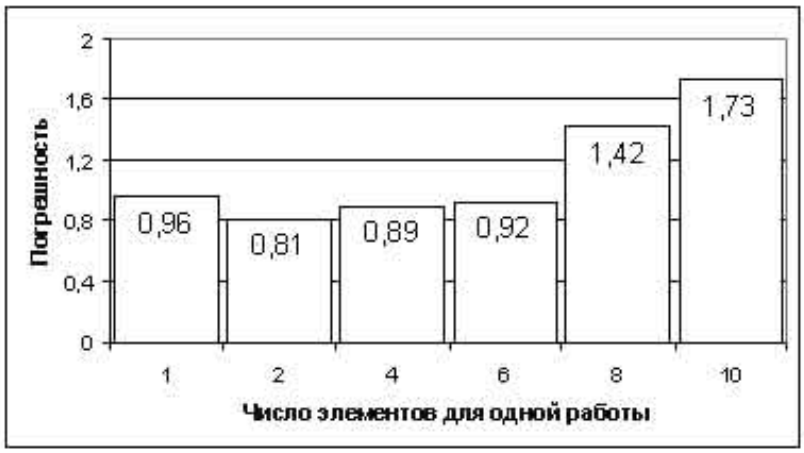


Рис. 3.3: Влияние мощности окрестности, класс j6029, $q$=0,2

лучших приближенных решений, отбираемых при решении задачи о рюкзаке и для каждого их них строилось соседнее расписание. Таким образом, мощность окрестности возрастала в несколько раз. Рисунок 3.3 показывает влияние этого процесса на относительную погрешность алгоритма. Добавление "плохих" приближенных решений задачи о рюкзаке не дает положительного эффекта, засоряет окрестность некачественными расписаниями и ухудшает результаты поиска. Тем не менее, наилучший выбор соответствует не одному, а двум приближенным решениям задачи о многомерном рюкзаке.

### 3.3.3 Влияние списка запретов

Основным назначением списка запретов является предотвращение зацикливания. Как уже отмечалось выше, функция $f(S) = \sum_{j \in J} s_j$ запрещает не только просмотренные решения, но и многие другие. Целью следующего эксперимента было определение оптимальной длины списка запретов и его влияния на относительную погрешность. На рисунке 3.4 показано типичное поведение алгоритма при длине списка запретов $TL$=5, 50 и 500. Наличие горизонтальных участков (см. рис. 3.4 б) и в)) свидетельствует о недостаточной длине списка. Алгоритм попадает на своеобразное "плато" и длительное время не может с него уйти. При $TL = 500$ такого эффекта не наблюдается и поиск становится более продуктивным.

Внесение рандомизации окрестности меняет поведение алгоритма. При $q = 0,2$ (см. рис. 3.5) уже при $TL$=5 "плато" не является препятствием для

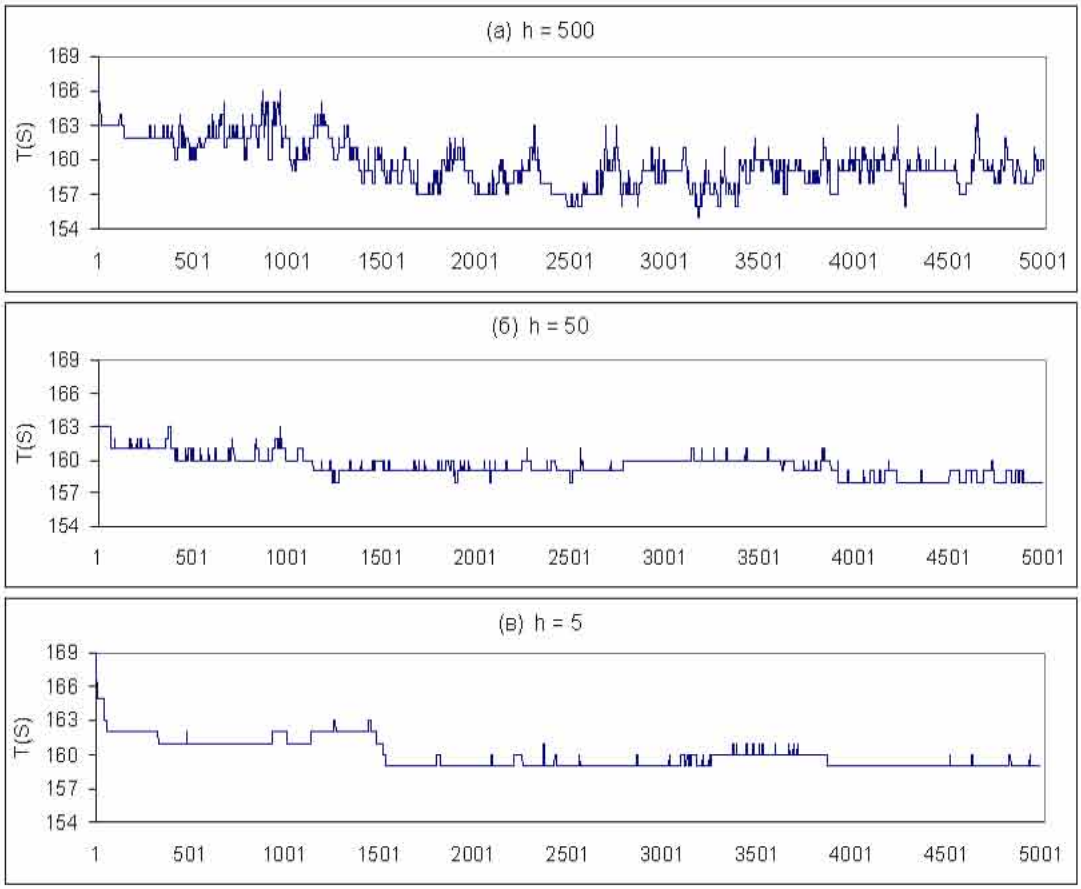


Рис. 3.4: Влияние списка запретов, j6029, $q = 1$

алгоритма. Влияние списка запретов ослабевает. Таким образом, рандомизация окрестности не только сокращает время ее просмотра, но и позволяет использовать списки запретов меньшей длины.

### 3.3.4 Выбор начального решения

Следующий эксперимент связан с выбором начального решения. В таблице 3.3 приведены значения относительной погрешности в случае, когда начальные решения выбираются случайно и с помощью алгоритма "Пинг-понг".

Как видно из таблицы, на задачах малой размерности выбор начального решения почти не влияет на итоговый результат. Преимущество в выборе хорошего начального решения сказывается с ростом размерности задачи. Это объясняется тем, что на небольших задачах локальному поиску требуется незначительное число итераций чтобы существенным образом улучшить неудачно выбранное начальное решение. В задачах большой раз-

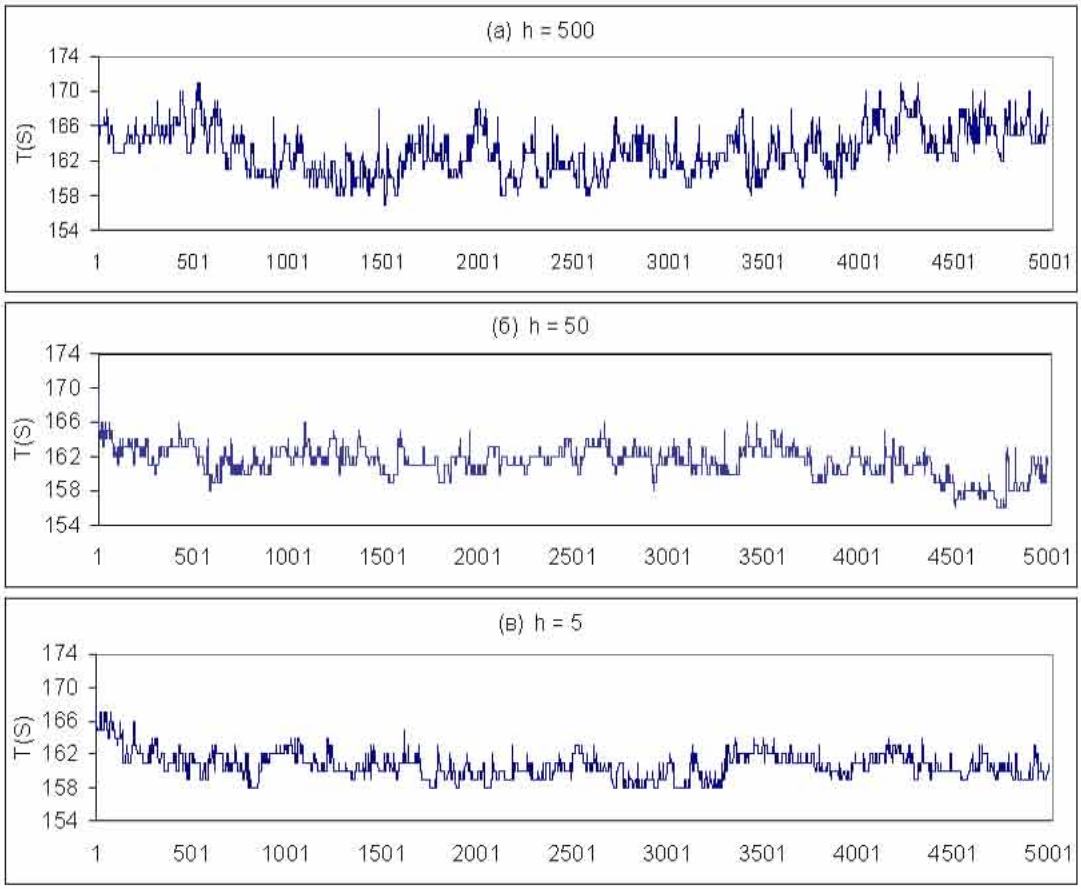


Рис. 3.5: Влияние списка запретов, j6029, $q = 0, 2$

мерности требуется большее время для исправления ситуации. Несмотря на то, что при $n$=120 алгоритм выполнял 20000 шагов, случайный выбор начального решения привел к худшему результату на всех трех классах j12016, j12036 и j12056. Таким образом, на задачах большой размерности целесообразно использовать эвристические процедуры для генерации хорошего начального приближения.

### 3.3.5 Интенсификация поиска

Последний эксперимент связан с влиянием интенсификации поиска на поведение алгоритма. В этом эксперименте сравниваются три варианта алгоритма, различающихся частотой возвращения к наилучшему найденному решению:

- поиск без возвращения с общим числом итераций $I(n)$,
- поиск с возвращением через каждые $I(n)/5$ итераций,

| Класс | Случайное | Пинг-понг |
|---|---|---|
| j3013 | 0,00 | 0,07 |
| j3029 | 0,12 | 0,11 |
| j3045 | 0,10 | 0,08 |
| j6013 | 0,81 | 0,83 |
| j6029 | 0,99 | 0,81 |
| j6045 | 0,50 | 0,27 |
| j12016 | 1,47 | 1,42 |
| j12036 | 1,22 | 1,13 |
| j12056 | 1,22 | 1,03 |

Таблица 3.3: Влияние начального решения на $\varepsilon(\%)$

| Класс | $\varepsilon_1$ | $\varepsilon_5$ | $\varepsilon_{10}$ |
|---|---|---|---|
| j3013 | 0,07 | 0,07 | 0,10 |
| j3029 | 0,22 | 0,11 | 0,20 |
| j3045 | 0,08 | 0,08 | 0,12 |
| j6013 | 0,92 | 0,83 | 0,84 |
| j6029 | 0,96 | 0,81 | 0,86 |
| j6045 | 0,26 | 0,27 | 0,27 |
| j12016 | 1,47 | 1,42 | 1,48 |
| j12036 | 1,20 | 1,13 | 1,11 |
| j12056 | 0,94 | 1,03 | 1,16 |

Таблица 3.4: Влияние интенсификации поиска, $q$=0,2

- поиск с возвращением через каждые $I(n)/10$ итераций.

Общее число итераций $I(n)$ составляло 5000, 10000 и 20000 при $n$ равном 30, 60 и 120, соответственно. В таблице 3.4 приведены средние значения $\varepsilon_1$, $\varepsilon_5$ и $\varepsilon_{10}$ относительной погрешности для каждого из трех вариантов алгоритма.

Как видно из таблицы, второй вариант алгоритма имеет небольшое преимущество в большинстве рассмотренных классов. Это позволяет сделать вывод о том, что интенсификация поиска не оказывает существенного влияния на результат работы алгоритма.

### 3.3.6 Сравнение с другими алгоритмами

Результаты сравнения разработанного алгоритма поиска с запретами и чередованием окрестностей (ПЗЧО) с другими известными алгоритмами приведены в таблице 3.5. Алгоритмы сравнивались по двум показателям: средняя относительная погрешность и время счета. Для сравнения рассмотрены следующие алгоритмы:

- Гибридный алгоритм (ГИА) [86], сочетает в себе идею локального поиска с элементами полного перебора. Такой подход, по-видимому, должен обеспечивать высокое качество получаемых решений, но требовать достаточно много времени.
- Эволюционный алгоритм (ЭА) [104], основанный на стратегии связывающих путей. В этом алгоритме по сути отсутствует локальный поиск, что приводит к малому времени счёта.
- Генетический алгоритм (ГА) [51], использующий кодировку списком и двуточечный оператор скрещивания. Особенностью этого алгоритма является то, что в нем на каждой итерации имеет место случайный выбор между последовательным и параллельным декодером.
- Алгоритм имитации отжига (ИО) [28], использующий списочную кодировку и последовательный декодер.
- Алгоритм поиска с запретами (ПЗ1) [20], использующий кодировку в виде логической схемы со специально разработанным для нее декодером.
- Алгоритм поиска с запретами (ПЗ2) [77], использующий списочную кодировку и последовательный декодер.
- Поиск по переменной окрестности (ППО) [44], использующий списочную кодировку и последовательный декодер.
- Алгоритм муравьиной колонии (МК) [74], использующий последовательный декодер, ориентированный на приоритетное правило LFT.
- Алгоритм лагранжевой релаксации (ЛР) [76], использующий задачу о минимальном разрезе при построении нижней оценки.

- Алгоритм 'CARA' [103], представляющий собой локальный поиск, основанный на перемещении критических работ.

В [98] предложен оригинальный вариант алгоритма поиска с запретами. Он не использует кодировок, а работает прямо с расписаниями. К сожалению, алгоритм тестировался на другой библиотеке тестовых задач. Однако известно, что он уступает генетическому алгоритму, предложенному в работе [51].

Таблица имеет следующую структуру. В первой колонке стоит название алгоритма. Во второй, третьей и червертой – числовые характерискии алгоритмов для задач размерности 30, 60 и 120, соответственно. Каждая из трех последних колонок разбита на две подколонки, в которых, соответственно, представлены средние значения относительной погрешности и времени счета по всем примерам данной размерности библиотеки PSPLib. Для задач с числом работ, равным 30, погрешность вычислялась по отношению к оптимуму. Для задач большей размерности данная величина вычислялась по отношению к нижней оценке. Алгоритмы тестировались на следующих вычислительных машинах:

ПЗЧО — PENTIUM III 1800MHz 256Mb RAM,
ЭА, CARA — AMD 400MHz,
ППО — PENTIUM III 1GHz,
ГИА — 4 proc. HP 9000 440MHz 2Gb RAM,
ГА — PENTIUM 133MHz 32Mb RAM,
ПЗ2 — Sun Ultra 2 300MHz 1Gb RAM,
ЛР — Sun Ultra 2 200MHz 512Mb RAM,
МК — PENTIUM III 500MHz,
ПЗ1 — SUN/Sparc 20/801 80MHz + 1Mb SC.
Для алгоритма ИО тип вычислительной машины неизвестен.

Результаты численных экспериментов показывают, что разработанный алгоритм не уступает по качеству получаемых решений лучшим известным алгоритмам. Кроме того, для некоторых примеров из библиотеки PSPLib с помощью данного алгоритма были получены приближенные решения с рекордным значением целевой функции: в классах j609, j6025, j12016 и j12036 по одному примеру, в классах j6013 и j6029 по два примера, в классах j6045, j12056 по три примера. Полученные результаты позволяют сделать вывод

| Алгоритм | $\varepsilon$, (%) $n$ = | $t$ (с) 30 | $\varepsilon$, (%) $n$ = | $t$ (с) 60 | $\varepsilon$, (%) $n$ = | $t$ (с) 120 |
|---|---|---|---|---|---|---|
| ПЗЧО | 0,01 | 0,110 | 10,69 | 6,465 | 31,93 | 44,673 |
| ЭА | 0,13 | 0,38 | 10,98 | 1,14 | 32,18 | 14,52 |
| ППО | 0,01 | 0,64 | 10,94 | 8,89 | 33,10 | 219,86 |
| ГИА | 0,02 | 22,23 | 10,93 | 58,03 | 33,16 | 318,92 |
| CARA | 0,06 | 1,61 | 11,46 | 2,76 | 34,53 | 17,00 |
| ГА | 0,17 | – | 11,89 | – | 35,60 | 14,05 |
| ПЗ2 | – | – | – | – | 35,86 | 109,4 |
| ЛР | – | – | – | – | 36,00 | 72,9 |
| МК | – | – | – | – | 36,65 | – |
| ИО | 0,23 | – | 11,90 | – | 37,68 | – |
| ПЗ1 | 0,44 | – | 37,68 | – | – | – |

Таблица 3.5: Сравнительная характеристика алгоритмов, 5000 итераций

об эффективности данного подхода для решения задачи календарного планирования с ограниченными ресурсами.

# Глава 4

# Эволюционные алгоритмы

Идея эволюционных алгоритмов заимствована у живой природы и состоит в моделировании биологического процесса, конечной целью которого является получение оптимального решения в сложной комбинаторной задаче. Эволюционный алгоритм представляет собой итерационный случайный процесс, оперирующий с набором особей – *популяцией*. Каждая особь является допустимым решением рассматриваемой задачи. Существует много разновидностей эволюционных алгоритмов. Одним из наиболее простых и естественных методов является генетический алгоритм.

Стандартный генетический алгоритм начинает свою работу с формирования начальной популяции $POP = \{S_1, S_2, \ldots, S_m\}$ – конечного набора допустимых решений задачи. Эти решения могут быть выбраны случайным образом или получены с помощью эвристических методов, например вероятностного жадного алгоритма.

Каждый шаг эволюции состоит из трех основных этапов: отбора кандидатов, операции скрещивания и обновления популяции. В ходе отбора кандидатов выделяется часть популяции, содержащая наиболее "перспективных" родителей для достижения желаемого результата. Данная "элита" разбивается на пары, к каждой из которых применяется операция скрещивания, в результате чего порождаются новые особи. Полученные решения подвергается небольшим случайным модификациям, которые принято называть *мутациями*. Затем эти решения добавляются в популяцию, тем самым ее увеличивая. Чтобы вернуть ей исходный размер применяется процедура обновления, которая отсеивает наименее подходящие решения. Затем описанные три этапа повторяются снова, формируя эволюционный

процесс, продолжающийся пока не выполнен критерий остановки.

Ниже приводится схема работы стандартного генетического алгоритма.

Генетический алгоритм

0. Построить начальную популяцию $POP = \{S_1, \ldots, S_m\}$.
1. Выбрать множество кандидатов $POP_{ELT} \subseteq POP$.
2. Разбить множество $POP_{ELT}$ на пары.
3. Применить к каждой паре оператор скрещивания.
4. Применить к каждому из полученных решений оператор мутации.
5. Добавить полученные решения в популяцию.
6. Обновить популяцию.
7. Если не выполнен критерий остановки, вернуться на шаг 1.

Существует много способов модифицировать стандартный алгоритм. Например, на шаге 4 вместо оператора мутации можно применять более сложные алгоритмы локальной перестройки, такие как локальный поиск.

Для ЗКПОР опубликовано достаточно много эволюционных алгоритмов [63], большинство которых — генетические алгоритмы. Характерной особенностью генетических алгоритмов является то, что в качестве множества кандидатов, как правило, выступает вся популяция или значительная ее часть. В данной работе предлагается принципиально иная схема эволюционного алгоритма, т.н. *стратегия связывающих путей*, известная в иностранной литературе как *Path Relinking* [46, 47]. Согласно этой стратегии, из популяции выбирается одна пара "родителей", для которой строится некоторый набор решений — *связывающий путь*. На этом пути выбирается новое решение, которое затем добавляется в популяцию, которая в дальнейшем подлежит операции отбора. В отличие от стандартного генетического алгоритма, число потомков, генерируемое на текущем этапе эволюции, относительно мало. Это обстоятельство позволяет успешно использовать эволюционный алгоритм в сочетании с локальным поиском на большую глубину, таким как поиск с запретами, имитация отжига, поиск по переменной окрестности и т.п. В разделе 4.3 обсуждаются преимущества такой комбинированной стратеги. Алгоритмы, основанные на идеях

стратегии связывающих путей, хорошо зарекомендовали себя в решении многих задач комбинаторной оптимизации, таких как задача Штейнера на графах [21], трехиндексная, обобщенная, двукритериальная и квадратичная задача о назначениях [15, 17, 45, 79], задача MAX-CUT [42] и многие другие [46]. В данной работе предлагается реализация стратегии связывающих путей для решения ЗКПОР.

## 4.1 Стратегия связывающих путей

Как отмечалось выше, стратегия связывающих путей (ССП) имеет следующие принципиальные отличия от генетических алгоритмов (ГА):

— размер популяции,
— мощность множества кандидатов,
— особый оператор скрещивания,
— число потомков.

В отличие от генетического алгоритма, ССП оперирует с относительно небольшой популяцией, порядка 5-20 особей, в то время как в ГА это число составляет 100-200 и более решений. Несколько иначе работает оператор скрещивания. В ГА пара родителей порождает одно-два решения. ССП строит целый набор решений, составляющих связывающий путь. Множество кандидатов, как правило, представляет собой два решения, которые называются *порождающей парой*. Однако это не жесткое требование и существуют модификации с тремя и более кандидатами [46]. Ниже приводится схема работы эволюционного алгоритма на основе стратегии связывающих путей.

Алгоритм ССП

1. Построить начальную популяцию $POP = \{S_1, \ldots, S_m\}$.
2. Определить порождающую пару $S_i,\ S_j$.
3. Построить связывающий путь $P = \{S^0, \ldots, S^k \mid S^0 = S_i,\ S^k = S_j\}$.
4. Выбрать случайную часть $PR \subset P$.
5. Найти в $PR$ элемент $S^*$ с наименьшим значением целевой функции.
6. Положить $POP = POP \cup \{S^*\}$.

7. Найти в $POP$ элемент $S^W$ с наибольшим значением целевой функции.
8. Положить $POP = POP \setminus \{S^W\}$.
9. Если не выполнен критерий остановки, вернуться на шаг 2.

На четвертом шаге алгоритма ССП используется рандомизация, что вносит разнообразие в эволюционный процесс. Начальная популяция может быть выбрана случайно или построена с помощью вероятностных жадных алгоритмов. Последний вариант позволяет строить хорошие начальные популяции, но вместе с тем требует дополнительного времени. Выбор порождающей пары имеет важное значение. Она может быть выбрана как случайным образом, так и с помощью некоторых правил. Эти и другие особенности алгоритма обсуждаются в разделе 4.3.

## 4.2 Построение связывающего пути

Основу ССП составляет оператор скрещивания, связывающий элементы порождающей пары набором допустимых решений. Рассмотрим текущую популяцию $POP = \{S_1, \ldots, S_{POPSIZE}\}$, где величина $POPSIZE$ обозначает размер популяции. Предположим, что порождающая пара выбрана и ее элементы закодированы в виде списков $L$ и $L'$.

Связывающий путь представляет собой набор списков $L_0,\ L_1, \ldots,\ L_k$, где $L_0 = L,\ L_k = L'$ и каждый последующий элемент набора является элементарным преобразованием предыдущего. В качестве такого преобразования предлагается использовать оператор сдвига, т.е. $L_i = Shift_{pq}(L_{i-1}),\ i = 1, \ldots, k$. Последовательный переход от списка $L$ к списку $L'$ достигается выбором пары работ $p$ и $q$ на каждом шаге $i$ построения связывающего пути. Для того, чтобы понять как выбираются эти работы нам потребуются следующие обозначения.

Через $B(j)$ будем обозначать множество работ, расположенных левее работы $j$ в списке $L$. Аналогичное обозначение $B'(j)$ введем и для списка $L'$.

**Определение 10** *Будем говорить, что для работы $j$ выполнено условие порядка, если $B'(j) \subseteq B(j)$.*

Это определение позволяет оценить степень различия двух списков в смысле числа элементарных преобразований, необходимых для превращения одного списка в другой.

Обозначим через $Order$ множество работ, для которых не выполнено условие порядка. На каждом этапе построения связывающего пути работа $p$ выбирается из множества $Order$. Работа $q$ однозначно определятся по выбранной работе $p$ как работа, занимающая максимальную позицию в списке $L$ среди всех работ множества $B'(p)$. Таким образом, в результате действия оператора $Shift_{pq}$ множество $Order$ сокращается по крайней мере на работу $p$. Такой способ построения связывающего пути гарантирует конечное число его элементов, не превосходящее мощности указанного множества.

**Теорема 6** *Длина связывающего пути не превосходит $n$.*

**Доказательство.** Справедливость утверждения теоремы следует из того, что на каждом шаге построения связывающего пути мощность множества $Order$ сокращается по крайней мере на единицу. Очевидно, что в результате действия оператора $Shift_{pq}$ работа $p$ исключается из множества $Order$. Поэтому осталось убедиться в том, что никакие работы из $J \setminus Order$ туда не попадут. Допустим, что это не так и найдется такая работа $j$. Понятно, что в этом случае она может находиться лишь между работами $p$ и $q$ в текущем списке $L$ и левее работы $q$ в списке $L'$, поскольку для всех остальных работ такое преобразование не нарушает условия порядка. Итак, $p \in B'(j)$ и $j \in J \setminus Order$. Следовательно, $B'(p) \subseteq B'(j)$, а поскольку по определению $q \in B'(p)$, значит, $q \in B'(j)$. С другой стороны, поскольку работа $j$ расположена левее работы $q$ в списке $L'$, то $q \notin B'(j)$, что является противоречием. Очевидно, что изначально $|Order| \leq n$, следовательно длина связывающего пути не превосходит $n$. Теорема доказана.

Заметим, что справедливость утверждения теоремы не зависит от выбора работы $p \in Order$. Это обстоятельство позволяет использовать жадный алгоритм построения связывающего пути, где на каждом шаге $i$ выбор работы $p$ осуществляется из условия минимизации длины расписания, соответствующего списку $L_i$, получаемого действием оператора $Shift_{pq}(L_{i-1})$. Другими словами, выбор работы $p$ подразумевает решение следующей оптимизационной задачи:

Найти

$$\min T(S)$$

при ограничениях

$$p \in Order,\ q \in B'(p),$$

$$B'(p) \setminus \{q\} \subseteq B(q),$$

где $T(S)$ — длина расписания, соответствующего списку $L_i$. Последнее ограничение означает, что работа $q$ занимает максимальную позицию в списке $L$ среди всех работ множества $B'(p)$.

Итак, оператор скрещивания порождает связывающий путь. На этом пути выбирается решение с наименьшим значением целевой функции. Однако выбор указанного решения может быть осуществлен не из всего пути, а только из некоторой случайно выделенной его части. Это привносит разнообразие в эволюционный процесс и благотворно сказывается на результатах.

## 4.3 Экспериментальные исследования

### 4.3.1 Свойства оператора скрещивания

Выбор оператора скрещивания играет ключевую роль в работе эволюционного алгоритма. Первый эксперимент посвящен сравнению результатов работы алгоритма с использованием нового оператора скрещивания, основанного на ССП и двух известных ранее опреаторов: одноточечного и двуточечного [50]. На рисунках 4.1—4.3 приводятся графики зависимости относительной погрешности получаемых решений в процессе эволюции. На графиках использованы обозначения: "OP" для одноточечного оператора скрещивания, "TP" для двуточечного и "PR" для связывающего пути. Эксперимент проводился на трёх классах тестовых примеров размерности 30, 60 и 120 работ.

На графиках хорошо видно кардинальное различие в относительной погрешности между новым оператором и известными ранее. Более того, с ростом числа просмотренных расписаний наблюдается более резкое ее сокращение. По-видимому, данный факт объясняется тем, что новый оператор строит существенно большее число потомков, что позволяет охватить

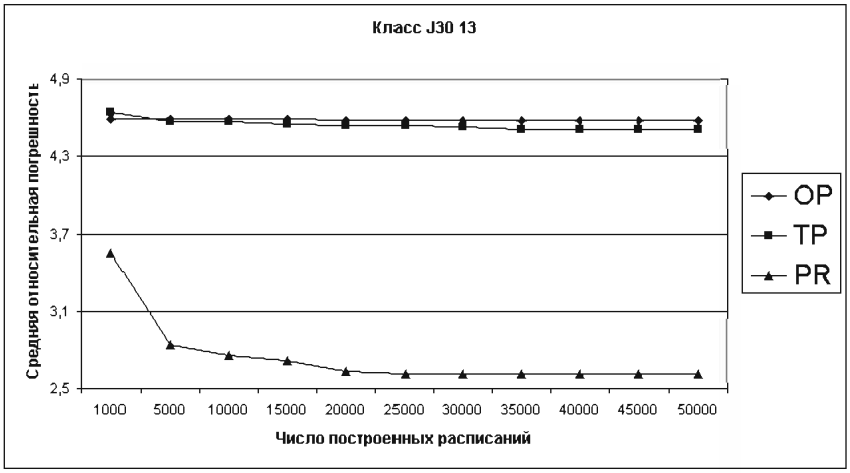


Рис. 4.1: Влияние оператора скрещивания, $n = 30$

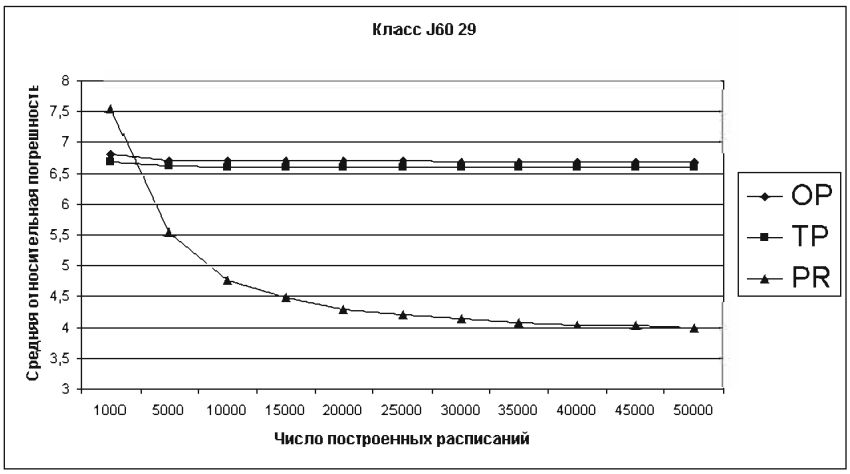


Рис. 4.2: Влияние оператора скрещивания, $n = 60$

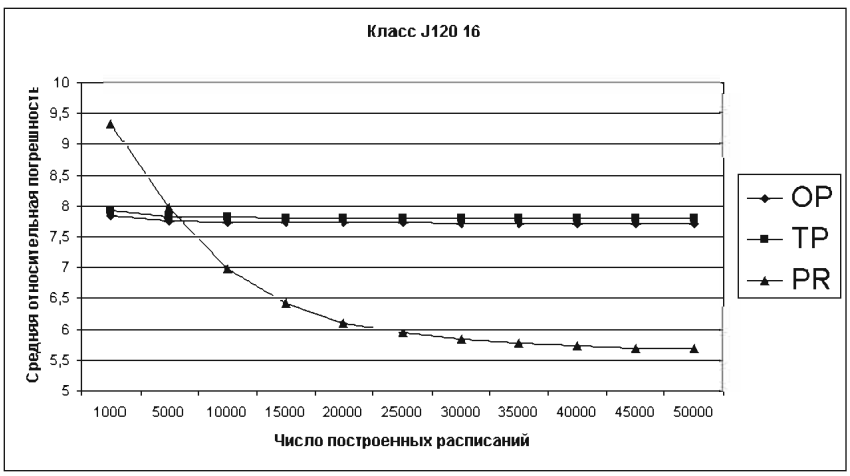


Рис. 4.3: Влияние оператора скрещивания, $n = 120$

большее пространство исследуемой области.

### 4.3.2 Выбор порождающей пары

В алгоритме исследовалось 4 способа выбора порождающей пары:

— оба решения выбираются случайно,

— лучшее и случайное,

— случайное и наиболее удаленное (в смысле мощности множества $Order$),

— случайное и ближайшее.

Все четыре способа по-своему оправданы. Первый способ вносит наиболь-

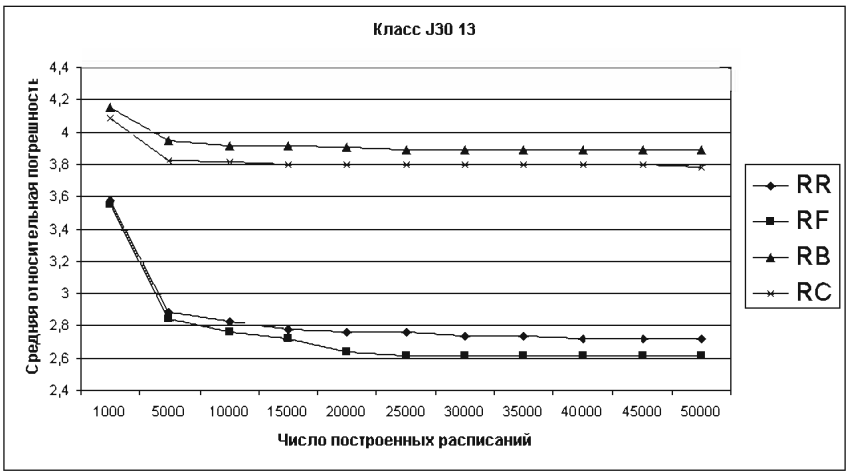


Рис. 4.4: Влияние порождающей пары, $n = 30$

шее разнообразие и одинаково хорош для задач разной размерности. Второй способ сочетает разнообразие с качеством, но, как ни странно, уступает предыдущему, особенно в примерах малой размерности. Причиной тому может служить чрезвычайно малый разброс значений целевой функции среди элементов популяции, наблюдаемый в наибольшей степени в конце эволюции, когда популяция "сходится". Идея третьего способа заключается в попытке охватить большую область поиска. При использовании четвертого способа, напротив, усилия алгоритма концентрируются "внутри" популяции. Последние два способа обладают переменным успехом в зависимости от размерности задачи. Преимущество третьего способа в случае малой размерности скорее всего объясняется относительно малым размером допустимой области и малой длиной связывающих путей. Поэтому при построении связывающего пути имеет смысл просматривать более широкие

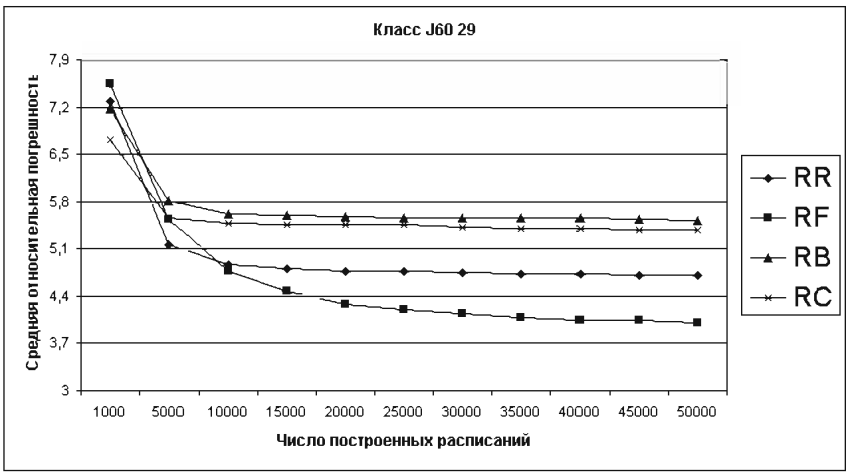


Рис. 4.5: Влияние порождающей пары, $n = 60$

области. В задачах большой размерности, напротив, исследование больших областей, как правило, не дает результатов, поскольку элементы популяции могут быть достаточно сильно удалены друг от друга, поэтому детальное исследование таких областей не представляется возможным. Однако, если в качестве порождающей пары рассматривать "близкие" элементы популяции, то с течением эволюции последняя начинает сходиться, обеспечивая тем самым детальное исследование вышеупомянутых областей.

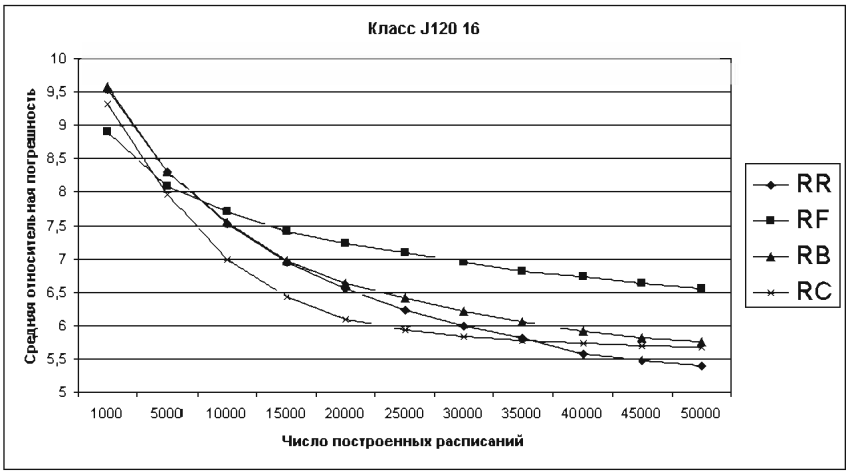


Рис. 4.6: Влияние порождающей пары, $n = 120$

### 4.3.3 Выбор начальной популяции

Как отмечалось ранее, начальная популяция может быть выбрана как случайно, так и с помощью вероятностных жадных алгоритмов. Случайный

выбор обеспечивает большее разнообразие и не требует существенных временных затрат. Основное преимущество жадных алгоритмов состоит в том, что они позволяют получить достаточно хорошие решения в относительно короткое время, а их вероятностные варианты еще и обеспечить разнообразие получаемых решений. На рисунках 4.7—4.9 показаны графики

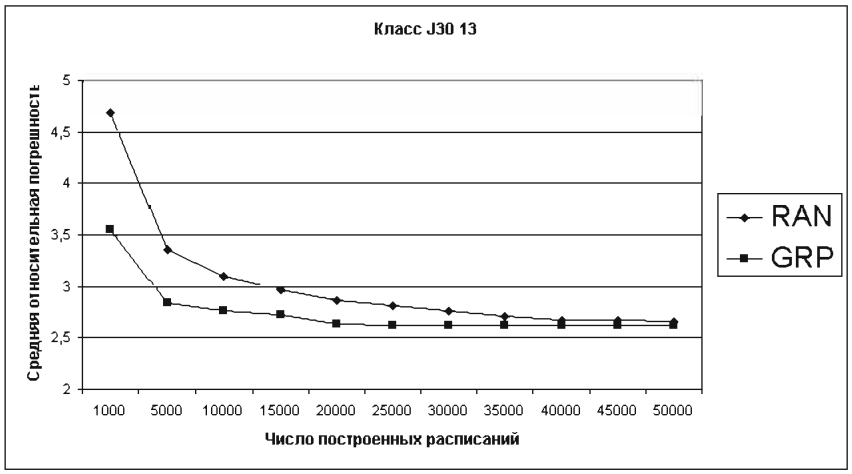


Рис. 4.7: Влияние начальной популяции, $n = 30$

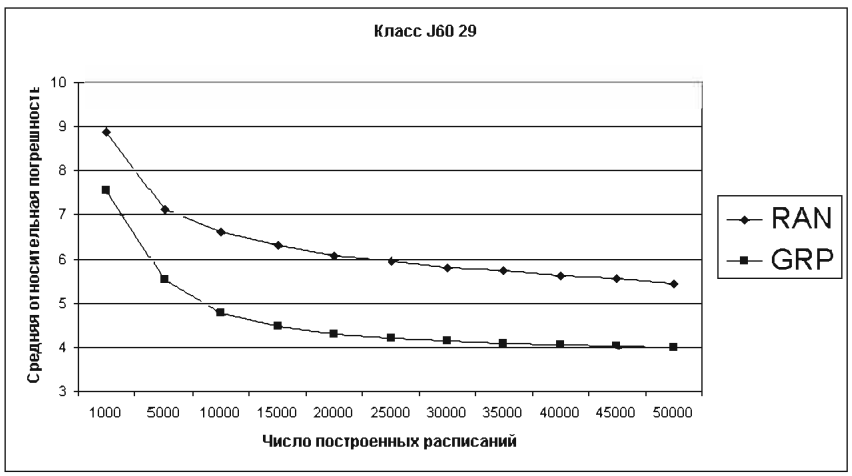


Рис. 4.8: Влияние начальной популяции, $n = 60$

изменения относительной погрешности алгоритма с течением эволюции в зависимости от выбора начальной популяции. Обозначение "RAN" соответствует случайному выбору начальной популяции. Через "GRP" обозначен вероятностный адаптивный метод МПР+ВУМ+ПП, описанный в главе 2.

Как видно из графиков, использование жадной эвристики приводит к сокращению относительной погрешности. Интересно отметить, что с ро-

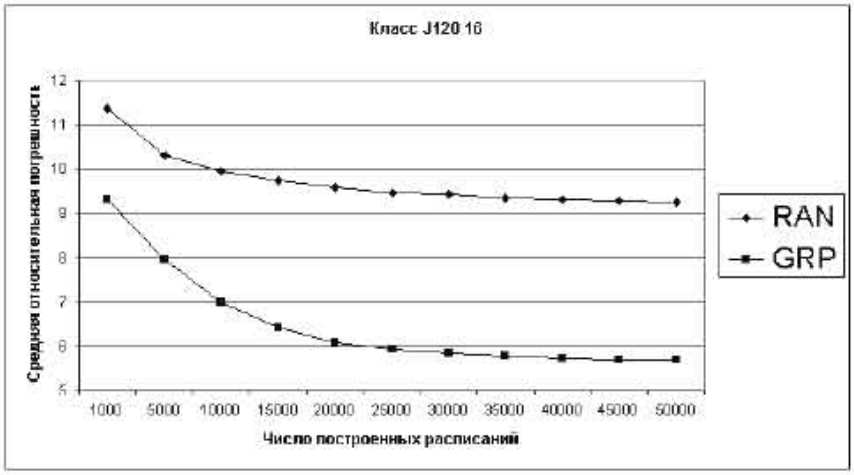


Рис. 4.9: Влияние начальной популяции, $n = 120$

стом числа просмотренных расписаний для задач малой размерности разрыв между графиками сокращается, а для больших задач наоборот, растет. По-видимому, это объясняется более сложным характером допустимой области в задачах большой размерности, обусловленным чрезвычайно большим числом локальных оптимумов.


Рис. 4.10: Влияние локального поиска, $n = 30$

### 4.3.4 Комбинация с локальным поиском

Особенностью разработанного эволюционного алгоритма является то, что на каждом шаге эволюции в популяцию добавляется только одно решение, выбираемое из связывающего пути. Это позволяет эффективно использо-

вать процедуру локального поиска в схеме эволюционного алгоритма, подвергая указанное решение локальному улучшению до занесения в популяцию. Для этой цели предлагается использовать алгоритм поиска с за-


Рис. 4.11: Влияние локального поиска, $n = 60$


Рис. 4.12: Влияние локального поиска, $n = 120$

претами по чередующимся окрестностям (ПЗЧО), описанный в главе 3. Он играет роль интенсификации поиска. Одним из ключевых вопросов использования локального поиска является его глубина, или, другими словами, число расписаний, отведенное на итерационный поиск по окрестности. Исследованию этого вопроса посвящен следующий эксперимент. На рисунках 4.10—4.12 приведена зависимость относительной погрешности алгоритма

от глубины локального поиска. Общее число расписаний, просматриваемое эволюционным алгоритмом ограничено числом 50000.

Результаты показывают, что оптимальное значение глубины ПЗЧО растет с увеличением размерности задачи. Важно, что при этом оно не достигает ни одного из крайних значений. Это означает, что эволюционный процесс и локальный поиск удачно дополняют друг друга. С одной стороны, локальный поиск играет роль интенсификации, глубже исследуя территории, прилегающие к выбранному решению. С другой стороны, ССП играет роль генератора начальной точки для локального поиска, выполняя функцию диверсификации. В таблице 4.1 приведено сравнение такого комбинированного алгоритма с алгоритмом ПЗЧО, состоящим из нескольких запусков из произвольной точки.

| Класс | ПЗЧО с мультистартом | ССП+ПЗЧО |
|---|---|---|
| j30 13 | 0,17 | 0,00 |
| j60 29 | 2,07 | 1,64 |
| j120 16 | 4,09 | 2,55 |

Таблица 4.1: Сравнение эволюционной стратегии с мультистартом

В таблицах В.2—В.7 приложения приведены средние значения погрешности, относительно нижней оценки в различных классах тестовых примеров библиотеки PSPLib. Знаком (★) отмечены те классы, в которых алгоритмом ССП+ПЗЧО были получены решения с рекордным значением целевой функции. Всего было получено 28 новых рекордов. Последнее обстоятельство еще раз подтверждает целесообразность использования такого сочетания метаэвристик.

### 4.3.5 Сравнение алгоритмов локального поиска

В этом разделе приводятся результаты сравнения трех алгоритмов, описанных в главах 2-4 диссертационной работы:
— вероятностного жадного алгоритма, использующего решение вспомогательной задачи на узкое место в качестве стратегии выбора и алгоритм пинг-понг в качестве фазы локального улучшения (МПР+ВУМ+ПП),
— поиск с запретами и чередованием окрестностей (ПЗЧО),
— эволюционный алгоритм, основанный на стратегии связывающих путей

и использующий алгоритм ПЗЧО в качестве фазы локального улучшения (ССП+ПЗЧО).

В таблице 4.2 приведены средние значения относительной погрешности по всем примерам библиотеки PSPLib размерности 30, 60 и 120 работ. Сравнение алгоритмов осуществляется в соответствии с общим критерием остановки по числу построенных расписаний. Максимальное число построенных расписаний составляло 1000, 5000 и 50000.

| Алгоритм | Критерий остановки | | |
|---|---|---|---|
| | 1000 | 5000 | 50000 |
| $n = 30$ | | | |
| МПР+ВУМ+ПП | 0,45 | 0,35 | 0,25 |
| ПЗЧО | 0,23 | 0,07 | 0,01 |
| ССП+ПЗЧО | 0,10 | 0,04 | 0,00 |
| $n = 60$ | | | |
| МПР+ВУМ+ПП | 12,10 | 11,98 | 11,84 |
| ПЗЧО | 12,01 | 11,30 | 10,77 |
| ССП+ПЗЧО | 11,71 | 11,17 | 10,74 |
| $n = 120$ | | | |
| МПР+ВУМ+ПП | 34,92 | 34,40 | 33,85 |
| ПЗЧО | 37,32 | 34,77 | 32,61 |
| ССП+ПЗЧО | 34,74 | 33,36 | 32,06 |

Таблица 4.2: Сравнение алгоритмов локального поиска

Результаты показывают, что в большинстве случаев метаэвристики выигрывают у жадного алгоритма. В большей степени это наблюдается в случае когда критерий остановки подразумевает построение большого числа расписаний. И наоборот, с уменьшением этого числа разница в относительной погрешности алгоритмов сокращается, а в случае когда $n = 120$ жадный алгоритм опережает ПЗЧО при построении 1000 и 5000 расписаний. Скорее всего это объясняется тем, что в задачах такой размерности локальный поиск проделывает чрезвычайно малое число итераций, поскольку мощность окрестности в этом случае достаточно велика.

Эволюционный алгоритм выигрывает у обоих методов независимо от критерия остановки. По-видимому, при построении большого числа расписаний выигрыш обеспечивается алгоритмом ПЗЧО, выполняющим функцию локального улучшения. В случае если предполагается построение ма-

лого числа расписаний выигрыш скорее всего обуславливается особенностями процедуры построения связывающего пути, по существу являющейся жадным алгоритмом. Тем не менее, следует отметить, что разница в относительной погрешности между жадным и эволюционным алгоритмом сокращается с ростом размерности задачи, особенно если речь идет о построении 1000 расписаний. Данное обстоятельство порождает мысль о том, что в задачах с числом работ более 120 преимущество может оказаться на стороне жадных алгоритмов.

Подводя итог, можно заключить, что в случае большой размерности и малого числа генерируемых расписаний целесообразно использование вероятностных жадных алгоритмов. В случае малой размерности и большого числа генерируемых расписаний, по-видимому, наиболее выгодно использовать метаэвристики.

### 4.3.6 Сравнение с мировым уровнем

В данном разделе приводится сравнение разработанного эволюционного алгоритма с другими известными алгоритмами для решения ЗКПОР. В сравнении участвуют свыше двадцати различных алгоритмов, краткое описание которых можно найти в обзоре [62]. Предлагаемый в диссертационной работе эволюционный алгоритм также присутствует в этом обзоре. Результаты сравнения приведены в таблицах 4.3—4.5. Следуя общепринятому критерию сравнения алгоритмов, число вычислений целевой функции ограничивалось величинами 1000, 5000 и 50000. В последующих трех таблицах приводятся средние значения относительной погрешности для каждого из трех указанных критериев остановки алгоритма. Для задач размерности 30 работ погрешность вычислялась относительно оптимума. Для задач большей размерности погрешность вычислялась по отношению к общепринятой нижней оценке [62].

Результаты сравнения показывают, что разработанный алгоритм превосходит все рассмотренные зарубежные аналоги в задачах размерности 30 работ. Он же оказался единственным алгоритмом, который получил оптимальное решение во всех примерах указанной размерности. В задачах большей размерности алгоритм несколько уступает генетическому алгоритму, разработанному группой ученых из Испании. К сожалению на дан-

ный момент он не опубликован.

| Algorithm | max. #schedules | | |
|---|---|---|---|
| | 1000 | 5000 | 50000 |
| GA, TS — path relinking (Kochetov, Stolyar, 2003) | 0,10 | 0,04 | 0,00 |
| GA — hybrid, FBI (Valls et al., 2003) | 0,27 | 0,06 | 0,02 |
| GA — forw.-backward (Alcaraz, Maroto, 2001) | 0,33 | 0,12 | — |
| GA — FBI (Valls et al., 2004) | 0,34 | 0,20 | 0,02 |
| sampling — LFT, FBI (Tormos, Lova, 2003) | 0,25 | 0,13 | 0,05 |
| TS — activity list (Nonobe, Ibaraki, 2002) | 0,46 | 0,16 | 0,05 |
| sampling — LFT, FBI (Tormos, Lova, 2001) | 0,30 | 0,16 | 0,07 |
| GA — self adapting (Hartmann, 2002) | 0,38 | 0,22 | 0,08 |
| GA — activity list (Hartmann, 1998) | 0,54 | 0,25 | 0,08 |
| sampling — LFT, FBI (Tormos, Lova, 2003) | 0,30 | 0,17 | 0,09 |
| sampling — random, FBI (Valls et al., 2004) | 0,46 | 0,28 | 0,11 |
| SA — activity list (Bouleimen, Lecocq, 2003) | 0,38 | 0,23 | — |
| GA — late join (Coelho, Tavares, 2003) | 0,74 | 0,33 | 0,16 |
| sampling — adaptive (Schirmer, 2000) | 0,65 | 0,44 | — |
| TS — schedule scheme (Baar et al., 1998) | 0,86 | 0,44 | — |
| sampling — adaptive (Kolisch, Drexl, 1996) | 0,74 | 0,52 | — |
| GA — random key (Hartmann, 1998) | 1,03 | 0,56 | 0,23 |
| sampling — LFT, $\alpha = 1$, serial (Kolisch, 1996) | 0,83 | 0,53 | 0,27 |
| sampling — global (Coelho, Tavares, 2003) | 0,81 | 0,54 | 0,28 |
| sampling — random, serial (Kolisch, 1995) | 1,44 | 1,00 | 0,51 |
| sampling — LFT, $\alpha = 3$ (Kolisch, 1996) | 1,05 | 0,78 | 0,56 |
| GA — priority rule (Hartmann, 1998) | 1,38 | 1,12 | 0,88 |
| sampling — WCS (Kolisch, 1996) | 1,40 | 1,28 | — |
| sampling — LFT, $\alpha = 1$, parallel (Kolisch, 1996) | 1,40 | 1,29 | 1,13 |
| sampling — random, parallel (Kolisch, 1995) | 1,77 | 1,48 | 1,22 |
| GA — problem space (Leon, Ramamoorthy, 1995) | 2,08 | 1,59 | — |

Таблица 4.3: Сравнение алгоритмов, PSPLib, $n = 30$

Для каждого из сравниваемых алгоритмов в таблице 4.6 приводится среднее и дисперсия номера занимаемой позиции в таблицах 4.3—4.5. Эти же данные продублированы на рисунке 4.13, что по мнению автора является более наглядным. Наилучшие результаты по-прежнему соответствуют предлагаемому в диссертационной работе алгоритму и вышеупомянутому генетическому алгоритму. Относительно низкая дисперсия свидетельствует об их устойчивости в смысле размерности задачи и критерия остановки.

| Algorithm | max. #schedules | | |
|---|---|---|---|
| | 1000 | 5000 | 50000 |
| GA — hybrid, FBI (Valls et al., 2003) | 11,56 | 11,10 | 10,73 |
| GA, TS — path relinking (Kochetov, Stolyar, 2003) | 11,71 | 11,17 | 10,74 |
| GA — FBI (Valls et al., 2004) | 12,21 | 11,27 | 10,74 |
| GA — self adapting (Hartmann, 2002) | 12,21 | 11,70 | 11,21 |
| GA — activity list (Hartmann, 1998) | 12,68 | 11,89 | 11,23 |
| sampling — LFT, FBI (Tormos, Lova, 2003) | 11,88 | 11,62 | 11,36 |
| sampling — LFT, FBI (Tormos, Lova, 2003) | 12,14 | 11,82 | 11,47 |
| GA — forw.-backward (Alcaraz, Maroto, 2001) | 12,57 | 11,86 | — |
| sampling — LFT, FBI (Tormos, Lova, 2001) | 12,18 | 11,87 | 11,54 |
| SA — activity list (Bouleimen, Lecocq, 2003) | 12,75 | 11,90 | — |
| TS — activity list (Nonobe, Ibaraki, 2002) | 12,97 | 12,18 | 11,58 |
| sampling — random, FBI (Valls et al., 2004) | 12,73 | 12,35 | 11,94 |
| sampling — adaptive (Schirmer, 2000) | 12,94 | 12,58 | — |
| GA — late join (Coelho, Tavares, 2003) | 13,28 | 12,63 | 11,94 |
| GA — random key (Hartmann, 1998) | 14,68 | 13,32 | 12,25 |
| GA — priority rule (Hartmann, 1998) | 13,30 | 12,74 | 12,26 |
| sampling — adaptive (Kolisch, Drexl, 1996) | 13,51 | 13,06 | — |
| sampling — WCS (Kolisch, 1996) | 13,66 | 13,21 | — |
| sampling — global (Coelho, Tavares, 2003) | 13,80 | 13,31 | 12,83 |
| sampling — LFT, $\alpha = 3$ (Kolisch, 1996) | 13,75 | 13,34 | 12,84 |
| sampling — LFT, $\alpha = 1$, parallel (Kolisch, 1996) | 13,59 | 13,23 | 12,85 |
| TS — schedule scheme (Baar et al., 1998) | 13,80 | 13,48 | — |
| GA — problem space (Leon, Ramamoorthy, 1995) | 14,33 | 13,49 | — |
| sampling — LFT, $\alpha = 1$, serial (Kolisch, 1996) | 13,96 | 13,53 | 12,97 |
| sampling — random, parallel (Kolisch, 1995) | 14,89 | 14,30 | 13,66 |
| sampling — random, serial (Kolisch, 1995) | 15,94 | 15,17 | 14,22 |

Таблица 4.4: Сравнение алгоритмов, PSPLib, $n = 60$

| Algorithm | max. #schedules | | |
|---|---|---|---|
| | 1000 | 5000 | 50000 |
| GA — hybrid, FBI (Valls et al., 2003) | 34,07 | 32,54 | 31,24 |
| GA — FBI (Valls et al., 2004) | 35,39 | 33,24 | 31,58 |
| GA, TS — path relinking (Kochetov, Stolyar, 2003) | 34,74 | 33,36 | 32,06 |
| population-based — FBI (Valls et al., 2004) | 35,18 | 34,02 | 32,81 |
| GA — self adapting (Hartmann, 2002) | 37,19 | 35,39 | 33,21 |
| sampling — LFT, FBI (Tormos, Lova, 2003) | 35,01 | 34,41 | 33,71 |
| ant system (Merkle et al., 2002) | — | 35,43 | — |
| GA — activity list (Hartmann, 1998) | 39,37 | 36,74 | 34,03 |
| sampling — LFT, FBI (Tormos, Lova, 2003) | 36,24 | 35,56 | 34,77 |
| sampling — LFT, FBI (Tormos, Lova, 2001) | 36,49 | 35,81 | 35,01 |
| GA — forw.-backward (Alcaraz, Maroto, 2001) | 39,36 | 36,57 | — |
| TS — activity list (Nonobe, Ibaraki, 2002) | 40,86 | 37,88 | 35,85 |
| GA — late join (Coelho, Tavares, 2003) | 39,97 | 38,41 | 36,44 |
| sampling — random, FBI (Valls et al., 2004) | 38,21 | 37,47 | 36,46 |
| SA — activity list (Bouleimen, Lecocq, 2003) | 42,81 | 37,68 | — |
| GA — priority rule (Hartmann, 1998) | 39,93 | 38,49 | 36,51 |
| sampling — adaptive (Schirmer, 2000) | 39,85 | 38,70 | — |
| sampling — LFT, $\alpha = 1$, parallel (Kolisch, 1996) | 39,60 | 38,75 | 37,74 |
| sampling — WCS (Kolisch, 1996) | 39,65 | 38,77 | — |
| GA — random key (Hartmann, 1998) | 45,82 | 42,25 | 38,83 |
| sampling — LFT, $\alpha = 3$ (Kolisch, 1996) | 41,27 | 40,38 | 39,34 |
| sampling — adaptive (Kolisch, Drexl, 1996) | 41,37 | 40,45 | — |
| sampling — global (Coelho, Tavares, 2003) | 41,36 | 40,46 | 39,41 |
| GA — problem space (Leon, Ramamoorthy, 1995) | 42,91 | 40,69 | — |
| sampling — LFT, $\alpha = 1$, serial (Kolisch, 1996) | 42,84 | 41,84 | 40,63 |
| sampling — random, parallel (Kolisch, 1995) | 44,46 | 43,05 | 41,44 |
| sampling — random, serial (Kolisch, 1995) | 49,25 | 47,61 | 45,60 |

Таблица 4.5: Сравнение алгоритмов, PSPLib, $n = 120$

| | Algorithm | Mean Rank | Rank Variance |
|---|---|---|---|
| 1. | GA — late join (Coelho, Tavares, 2003) | 10,89 | 0,32 |
| 2. | sampling — random, parallel (Kolisch, 1995) | 18,22 | 0,40 |
| 3. | GA — hybrid, FBI (Valls et al., 2003) | 1,44 | 0,47 |
| 4. | GA, TS — path relinking (Kochetov, Stolyar, 2003) | 1,89 | 0,54 |
| 5. | sampling — LFT, $\alpha = 3$ (Kolisch, 1996) | 14,89 | 0,54 |
| 6. | sampling — random, FBI (Valls et al., 2004) | 9,44 | 0,91 |
| 7. | sampling — LFT, FBI (Tormos, Lova, 2003) | 3,78 | 1,28 |
| 8. | sampling — global (Coelho, Tavares, 2003) | 14,22 | 1,28 |
| 9. | GA — self adapting (Hartmann, 2002) | 5,89 | 1,88 |
| 10. | sampling — LFT, FBI (Tormos, Lova, 2001) | 6,11 | 2,10 |
| 11. | sampling — random, serial (Kolisch, 1995) | 18,11 | 2,10 |
| 12. | GA — activity list (Hartmann, 1998) | 7,78 | 2,17 |
| 13. | sampling — LFT, FBI (Tormos, Lova, 2003) | 6,00 | 2,22 |
| 14. | sampling — LFT, $\alpha = 1$, serial (Kolisch, 1996) | 15,22 | 3,51 |
| 15. | GA — FBI (Valls et al., 2004) | 3,78 | 3,73 |
| 16. | GA — random key (Hartmann, 1998) | 14,78 | 4,17 |
| 17. | GA — priority rule (Hartmann, 1998) | 13,56 | 5,14 |
| 18. | sampling — LFT, $\alpha = 1$, parallel (Kolisch, 1996) | 14,56 | 6,91 |
| 19. | TS — activity list (Nonobe, Ibaraki, 2002) | 8,44 | 7,36 |

Таблица 4.6: Среднее и дисперсия положения алгоритмов в сравнительных таблицах

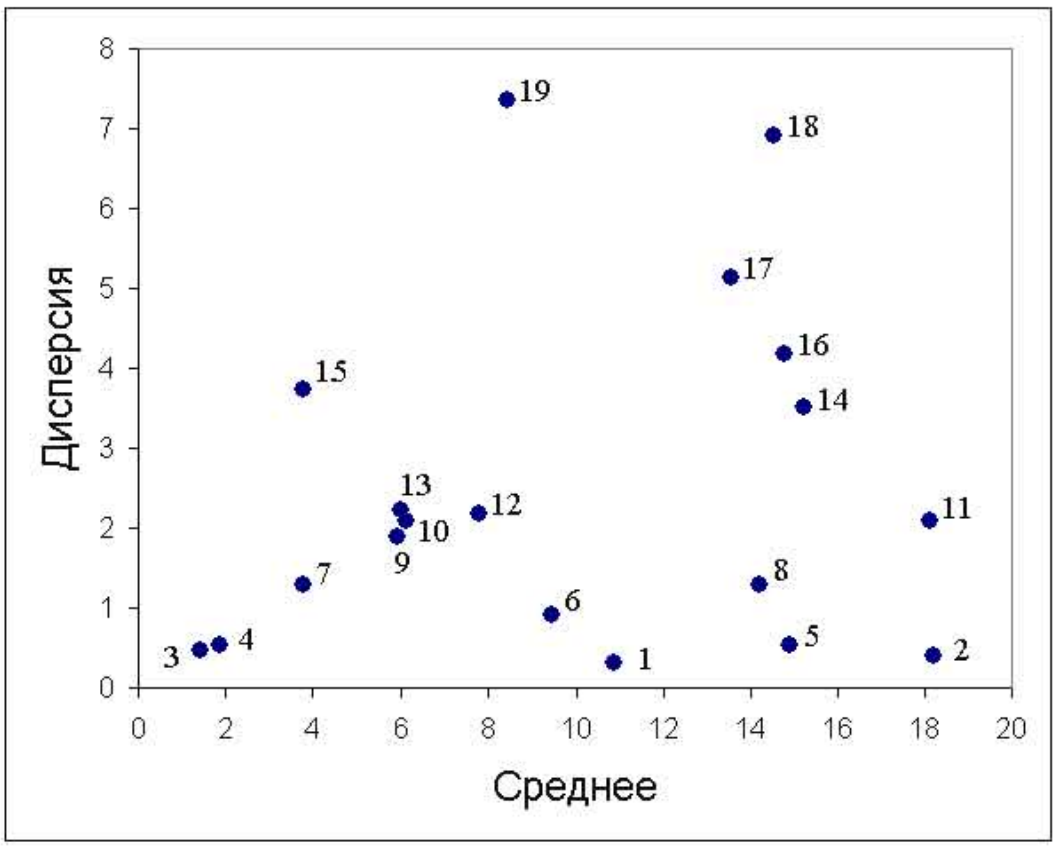


Рис. 4.13: Среднее и дисперсия положения алгоритмов в сравнительных таблицах

# Заключение

Основные результаты диссертационной работы заключаются в следующем.

Разработан новый вероятностный жадный алгоритм, использующий вспомогательную минимаксную задачу. В алгоритме используется фаза локального улучшения, основанная на итеративном переходе от активного расписания к T-позднему и обратно. Разработанный алгоритм является одним из лучших в классе жадных алгоритмов решения задачи календарного планирования с ограниченными ресурсами.

Разработан алгоритм поиска с запретами и чередованием окрестностей. В алгоритме использованы две удачно дополняющих друг друга окрестности, определенные, соответственно, для активных и T-поздних расписаний. Окрестности обладают экспоненциальной мощностью, но в алгоритме просматривается только полиномиальное число их элементов, которые выбираются с помощью решения задачи о многомерном рюкзаке.

Разработан новый эволюционный алгоритм, основанный на стратегии связывающих путей. Алгоритм содержит фазу локального улучшения, в ходе которой получаемые решения подлежат перестройке методом поиска с запретами. При формировании начального набора решений используется разработанный автором вероятностный жадный алгоритм.

Проведены численные эксперименты на примерах из международной библиотеки тестовых задач PSPLib [66]. Результаты показали, что разработанный эволюционный алгоритм, сочетающий в себе поиск с запретами и вероятностные жадные стратегии, является лучшим среди опубликованных эвристических методов. Для 28 тестовых примеров этой библиотеки алгоритмом были получены новые рекордные значения целевой функции.

# Литература


[1] Гимади Э. Х., Залюбовский В. В., Севастьянов С. В. Полиномиальная разрешимость задач календарного планирования с ограниченными ресурсами и директивными сроками. // Дискрет. анализ и исслед. операций. Сер. 2. 2000. Т. 7, № 1. С. 9–34.

[2] Гимади Э. Х., Глебов Н. И. Дискретные экстремальные задачи принятия решений: Учебное пособие / Новосибирский ун-т. 1991.

[3] Гончаров Е. Н., Кочетов Ю. А. Поведение вероятностных жадных алгоритмов для многостадийной задачи размещения // Дискрет. анализ и исслед. операций. Сер. 2. 1999. Т. 6, № 1. С. 12–32.

[4] Гончаров Е. Н., Кочетов Ю. А. Вероятностный поиск с запретами для дискретных задач безусловной оптимизации // Дискрет. анализ и исслед. операций. Сер. 2. 2002. Т. 9, № 2. С. 13–20.

[5] Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. Москва: Мир, 1982.

[6] Кочетов Ю. А., Столяр А. А. Алгоритм поиска с запретами для задачи календарного планирования с ограниченными ресурсами // Сборник тезисов конференции "Дискретный анализ и исследование операций", 2000. С.187.

[7] Кочетов Ю. А., Столяр А. А. Вероятностный адаптивный поиск для задачи календарного планирования с ограниченными ресурсами // Сборник тезисов конференции "Дискретный анализ и исследование операций", 2004. С.188.

[8] Кочетов Ю. А., Столяр А. А. Использование чередующихся окрестностей для приближенного решения задачи календарного планиро-

вания с ограниченными ресурсами // Дискрет. анализ и исслед. операций. Сер. 2. 2003. Т. 10, № 2. С. 29–55.

[9] Кочетов Ю. А., Столяр А. А. Локальный поиск с экспоненциальной окрестностью для задачи календарного планирования с ограниченными ресурсами // Сборник тезисов конференции "Проблемы оптимизации и экономические приложения", 2003. С.98.

[10] Столяр А. А. Задача календарного планирования с ограниченными ресурсами: исследование окрестностей для локального поиска // Труды XII Байкальской международной конференции 2001. Т. 6. С. 46–50.

[11] Столяр А. А. Стратегия связывающих путей для задачи календарного планирования с ограниченными ресурсами // Сборник тезисов конференции "Дискретный анализ и исследование операций", 2002. С. 238.

[12] Столяр А. А. Эволюционный поиск с запретами для задачи календарного планирования с ограниченными ресурсами // Сборник тезисов конференции "Математическое программирование и приложения", 2003. С. 218.

[13] Aarts E. H. L., Korst J. H. M., van Laarhoven P. J. M. Simulated annealing // Local search in combinatorial optimization. Chichester: John Wiley & Sons, 1997. P. 91–120.

[14] Ahuja R. K., James O. E., Orlin B., Punnen A. P. A survey of very large-scale neighborhood search techniques // Discrete Appl. Math. 2002. V. 123, P. 75–102.

[15] Aiex R. M., Resende M. G. C., Pardalos P.M., Toraldo G. GRASP with path relinking for the three-index assignment problem // Techn. rep. AT&T Labs Research, Florham Park, NJ, 07733. 2000.

[16] Alcaraz J., Maroto C. A robust genetic algorithm for resource allocation in project scheduling // Annalls of Operaions Research, V. 102. 2001. P. 83–109.

[17] Alfandari L., Plateau A., Tolla P. A two-phase path-relinking algorithmfor the generalized assignment problem // Proc. of 4th Metaheuristics International Conference, 2001. P. 175–180.

[18] Alvarez-Valdés R., Tamarit J. M. Heuristic algorithms for resource-constrained project scheduling: A review and empirical analysis // Advances in project scheduling. Amsterdam: Elsevier, 1989. P. 113–134.

[19] Alvarez-Valdés R., Tamarit J. M. Algoritmos heurísticos deterministas y aleatorios en secuenciacón de proyectos con recursos limitados // Qüestiió. 1989. V. 13. P. 173–191.

[20] Baar T., Brucker P., Knust S. Tabu-search algorithms for the resource-constrained project scheduling problem. Working Paper, Universität Osnabrück, 1997.

[21] Bastos M. P., Ribeiro C. C. Reactive tabu search with path-relinking for the steiner problem in graphs // Essays and Surveys of Metaheuristics. Boston: Kluwer Acad. Publ., 2002. P. 39–58.

[22] Battiti R., Protasi M. Reactive local search for maximum clique // Proc. of Workshop on Algorithm Engineering, 1997. P. 74–82.

[23] Bell C., Han J. A new heuristic solution method in resource-constrained project scheduling // Nav. Res. Logist. 1991. V. 38. P. 315–331.

[24] Bixby N., Boyed E. Using of CPLEX callable library // CPLEX Optimization Inc.. Houston. 1996.

[25] Blażewich J., Lenstra J. K., Rinooy Kan A. H. G. Scheduling subject to resource constraints: Classification and complexity // Discr. Appl. Math. 1983. V. 5. P. 11–24.

[26] Boctor F. Some efficient multi-heuristic procedures for resource-constrained project scheduling // European J. Oper. Res. 1990. V. 49. P. 3–13.

[27] Boese K. D., Kahng A. B., Muddu S., A new adaptive multi–start technique for combinatorial global optimizations, Oper. Res. Lett. 16 (2), 1994, 101–114.

[28] Bouleimen K., Lecocq H. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple modes version // European J. Oper. Res. 2003. V. 149. P. 268–281.

[29] Brucker P., Knust S., Schoo A., Thiele O. A branch and bound algorithm for the resource-constrained project scheduling problem // European J. Oper. Res. 1998. V. 107. P. 272–288.

[30] Cho J.-H., Kim Y.-D. A simulated annealing algorithm for resource constrained project scheduling problems // J. Oper. Res. Society. 1997. V. 48. P. 736–744.

[31] Chrisrofides N., Alvarez-Valdés R., Tamarit J. M. Project scheduling with resource constraints: A branch and bound approach // European J. Oper. Res. 1987. V. 29. P. 262–273.

[32] Coelho J., Tavares L. Comparative analysis of meta-heuristics for the the resource constrained project scheduling problem // Techn. Rep. Department of Civil Engineering, Instituto Superior Tecnico, Portugal. 2003.

[33] Cooper D. A note on serial and parallel heuristics for resource-constrained project scheduling // Foundations of Control Engineering. 1977. V. 2, № 4. P. 131–133.

[34] Cooper D. Heuristics for scheduling resource-constrained projects: An expiremental investigation // Manag. Sci. 1976. V. 22, № 11. P. 1186–1194.

[35] Davis E. An experimental investigation of resource allocation in multi-activity projects // Operational Research Quarterly. 1973. V. 24. P. 587–591.

[36] Davis E., Patterson J. A comparison of heuristic and optimum solutions in resource-constrained project scheduling // Manag. Sci. 1975. V. 21. P. 944–955.

[37] Dorigo M., Stützle T. Ant Colony Optimization. Boston: MIT Press, 2004.

[38] Drexl A. Scheduling of project networks by job assignment // Manag. Sci. 1991. V. 37, №12. P. 1590–1602.

[39] Elmaghraby S. Activity networks: Project planning and control by network models. New York: Wiley, 1977.

[40] Elsayed E. Algorithms for project scheduling with resource constraints // International Journal of Production Research. 1982. V. 20, № 1. P. 95–103.

[41] Feo T. A., Resende M. G. C. Greedy randomized adaptive search procedures // J. Glob. Optim. 1995. V. 6. P. 109–133.

[42] Festa P., Resende M. G. C. CIRCUT+PR: A rank-2 heuristic with path relinking for MAX-CUT // Proc. of 5th Metaheuristics International Conference, 2003. P. 19-1—19-7.

[43] Festa P., Resende M. G. C. GRASP: An annotated bibliography // Essays and Surveys in Metaheuristics. Boston: Kluwer Acad. Publ., 2002. P. 325–368.

[44] Fleszar K., Hindi K. Solving the resource-constrained project scheduling problem by a variable neighborhood search // European J. Oper. Res.(to appear).

[45] Gandibleux X., Morita H., Katoh N. Impact of clusters, path-relinking and mutation operators on the heuristic using a genetic heritage for solving assignment problems with two objectives // Proc. of 5th Metaheuristics International Conference, 2003. P. 23-1—23-6.

[46] Glover F., Laguna M. Tabu search. Boston: Kluwer Acad. Publ., 1997.

[47] Glover F., Laguna M., Martí R. Fundamentals of scatter search path relinking // Control and Cybernetics. 2000. V. 39. P. 653–684.

[48] Hansen P., Mladenović N. Developments of variable neighborhood search // Essays and Surveys of Metaheuristics. Boston: Kluwer Acad. Publ., 2002. P. 415–440.

[49] Hansen P., Ribeiro C. C. Essays and Surveys of Metaheuristics. Boston: Kluwer Acad. Publ. 2002.

[50] Hartmann S. A competitive genetic algorithm for resource-constrained project scheduling // Nav. Res. Logist. 1998. V. 45, N 7. P. 733–750.

[51] Hartmann S. Self-adapting genetic algorithm with an application to project scheduling // Techn. Rep. University of Kiel, 1999.

[52] Kirkpatrick, S., Gelatt jr., C. D., Vecchi, M. P. Optimization by simulated annealing. *Science* 220, p.671-680, 1983.

[53] Kochetov Yu., Stolyar A. A GRASP approach to the resource constrained project scheduling problem // Proc. of 2-th Intern. Workshop on Discrete Optimization Methods in Production and Logistics. Extended Abstracts (Omsk-Irkutsk, Russia, July 20-27, 2004). P. 131–136.

[54] Kochetov Yu., Stolyar A. A probabilistic multi-pass heuristic for the resource constrained project scheduling problem // Proc. of 9-th Intern. Workshop on Project Management and Scheduling. Extended Abstracts (France, April 26-28, 2004). P. 319–322.

[55] Kochetov Yu, Stolyar A. Distribution of Local Optima for the Resource Constrained Project Scheduling Problem // Book of abstracts of International Conference on Operations Research, Duisburg, Germany, 2001, p.80.

[56] Kochetov Yu., Stolyar A. Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem // Proc. of 3-th Intern. Workshop of Computer Science and Information Technologies. Extended Abstracts (Ufa, Russia, September 16-18, 2003). P. 96–99.

[57] Kochetov Yu., Stolyar A. Evolutionary Tabu Search for the Resource Constrained Project Scheduling Problem // Book of abstracts of Joint International Meeting EURO/INFORMS, Istanbul, Turkey, 2003. p. 208.

[58] Kolisch R. Efficient priority rules for the resource-constrained project scheduling problem // Journal of Operations Management. 1996. V. 14, N 3. P. 179–192.

[59] Kolisch, R. Project scheduling under resource constraints – Efficient heuristics for several problem classes. Phisica, Heidelberg, 1995.

[60] Kolisch R. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation // European J. Oper. Res. 1996. V. 90. P. 320–333.

[61] Kolisch R., Drexl A. Adaptive search for solving hard project scheduling problems // Nav. Res. Logist. 1996. V. 43. P. 23–40

[62] Kolisch R., Hartmann S. Experimental investigation of heuristics for resource-constrained project scheduling: An update // Working Paper, Technical University of Munich. 2004.

[63] Kolisch R., Hartmann S. Heuristic algorithms for the resource-constrained project scheduling problem: classification and computational analysis // Project Scheduling: Recent Models, Algorithms and Applications. Boston: Kluwer Acad. Publ., 1999. P. 147–178.

[64] Kolisch R., Padman R. An integrated survey of project scheduling // Techn. Rep. 463, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel. 1997.

[65] Kolisch R., Schwindt C., Sprecher A. Benchmark instances for project scheduling problems // Project Scheduling. Recent Models, Algorithms and Applications. Boston: Kluwer Acad. Publ., 1999. P. 197–212.

[66] Kolisch R., Sprecher A. PSPLib — A project scheduling problem library // European J. Oper. Res. 1996. V. 96. P. 205–216.

[67] Kolisch R., Sprecher A., Drexl A. Characterization and generation of a general class of resource-constrained project scheduling problems — Easy and hard instances // Research report 301, Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel, 1992.

[68] Kolisch R., Sprecher A., Drexl A. Characterization and generation of a general class of resource-constrained project scheduling problems // Manag. Sci. 1995. V. 41. P. 1693–1703.

[69] Krämer A. Scheduling multiprocessor tasks on dedicated processors // Dissertation. Universität Osnabrück, 1995.

[70] Lawrence S. Resource constrained project scheduling — A computational comparison of heuristic scheduling techniques // Techn. Rep. Graduate School of Industrial Administration, Carnegie—Mellon University, Pittsburg. 1985.

[71] Li R.-Y., Willis J. An iterative scheduling technique for resource-constrained project scheduling // European J. Oper. Res. 1992. V. 56. P. 370–379.

[72] Martello S., Toth P. Knapsack problems. Algorithms and computer implementations. Chichester: John Wiley & Sons, 1990.

[73] Mausser H., Lawrence S. Exploiting block structure to improve resource-constrained project schedules // Metaheuristics. 1998. Forthcoming.

[74] Merkle D., Middendorf M., Schmeck H. Ant colony optimization for resource-constrained project scheduling // Proc. of the Genetic and Evolutionary Computation Conference, 2000. P. 893–900.

[75] Mingozzi A., Maniezzo V., Ricciardelli S., Bianco L. An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation // Manag. Sci. 1998. V. 44. P. 714–729.

[76] Möhring R. H., Schulz A. S., Stork F., Uetz M. Solving project scheduling problems by minimum cut computations // Manag. Sci. 2003. V. 49, N 3. P. 330–350.

[77] Nonobe K., Ibaraki T. Formulation and tabu search algorithm for the resource constrained project scheduling problem (RCPSP) // Techn. Rep. 99010. University of Kioto, 1999.

[78] Oğuz O., Bala H. A comparative study of computational procedures for the resource-constrained project scheduling problem // European J. Oper. Res. 1994. V. 72. P. 406–416.

[79] Oliveira C. A. S., Pardalos P.M., Resende M. G. C. GRASP with path relinking for the QAP // Proc. of 5th Metaheuristics International Conference, 2003. P. 57-1—57-6.

[80] Özdamar L., Ulusoy G. A local constrained based analysis approach to project scheduling under general resource constraints // European J. Oper. Res. 1994. V. 79. P. 287–298.

[81] Özdamar L., Ulusoy G. A note on an iterative forward/backward scheduling technique with reference to a procedure by Li and Willis // European J. Oper. Res. 1996. V. 89. P. 400–407.

[82] Özdamar L., Ulusoy G. An iterative local constraint based analysis for solving the resource-constrained project scheduling problem // Journal of Operations Management. 1996. V. 14, N 3. P. 193–208.

[83] Patterson J. H. Alternate methods of project scheduling with limited resources // // Nav. Res. Logist. Quarterly. 1973. V. 23. P. 767–784.

[84] Patterson J. H. Project scheduling: The effects of problem structure on heuristic performance // Nav. Res. Logist. Quarterly. 1976. V. 20. P. 95–123.

[85] Patterson J. H., Sowiński R., Talbot F. B., Weglarz J. An algorithm for a general class of precedence and resource constrained scheduling problems // Advances in Project Scheduling. Amsterdam: Elsevier, 1989. P. 3–28.

[86] Palpant M., Artigues Ch., Michelon Ph. Conception d'une métaheuristique et application au problème d'ordonnancement de projet à moyens limités // LIA Techn. Rep. 252. Université d'Avignon, 2001.

[87] Pollack-Johnson B. Hybrid structures and improving forecasting and scheduling in project managment // Journal of Operations Management. 1995. V. 12. P. 101–117.

[88] Pritsker A., Watters L., Wolfe P. Multiproject scheduling with limited resources: A zero-one programming approach // Manag. Sci. 1969. V. 16. P. 93–107.

[89] Ribeiro C., Hansen P. Essays and Surveys of Metaheuristics. Boston: Kluwer Acad. Publ., 2002.

[90] Sampson S., Weiss E. Local search techniques for the generalized resource constrained project scheduling problem // Nav. Res. Logist. 1993. V. 40. P. 665–675.

[91] Shaffer, L., Ritter, J., Meyer, W. The critical-path method. McGraw Hill, New York, 1965.

[92] Sprecher A., Solving the RCPSP efficiently at modest memory requirements // Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, N. 425, 1996.

[93] Sprecher A., Kolisch R., Drexl A., Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem // European J. Oper. Res. 1995. V. 80. P. 94–102.

[94] Schirmer, A. Case-based reasoning and improved adaptive search for project scheduling // Techn. Rep. 472, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel. 1998.

[95] Schirmer, A., Riesenberg S. Parameterized heuristics for project scheduling — Biased random sampling methods // Techn. Rep. 456, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel. 1997.

[96] Stinson J. P., Davis E. W., Khumawala B. M. Multiple resource-constrained scheduling using branch and bound // AIIE Transactions. 1978. V. 10. P. 252–259.

[97] Thesen A. Heuristic scheduling of activities under resource and precedence restrictions // Manag. Sci. 1976. V. 23, № 4. P. 412–422.

[98] Thomas P., Salhi S. A tabu search approach for the resource constrained project scheduling problem // J. Heurist. 1998. V. 4. P. 123–139.

[99] Thomas P., Salhi S. An investigation into the relationship of heuristic performance with network-resource characteristics // J. Oper. Res. Society. 1997. V. 48. P. 34–43.

[100] Tormos P., Lova A. A competitive heuristic solution techniques for resource-consrtained project scheduling // Annals of Operations Research. 2001. V. 102. P. 65–81.

[101] Tormos P., Lova A. An efficient multi-pass heuristic for project scheduling with consrtained resources // International Journal of Production Research. 2003. V. 41, № 5. P. 1071–1086.

[102] Ulusoy G., Özdamar L. Heuristic perfomance and network/resource characteristics in resource-constrained project scheduling // J. Oper. Res. Society. 1989. V. 40. P. 1145–1152.

[103] Valls V., Ballestín F., Quintanilla S. Resource-constrained project scheduling: a critical activity reordering heuristic // Proc. of 7-th Intern. Workshop on Project Management and Scheduling. 2000. P. 282–283.

[104] Valls V., Ballestín F., Quintanilla S. A population-based approach to the resource-constrained project scheduling problem // Techn. Rep. 10-2001. University of Valencia, 2001.

[105] Valls V., Ballestín F., Quintanilla S. Justification and RCPSP: A technique that pays // European J. Oper. Res., 2004.

[106] Valls V., Pérez M., Quintanilla M. Heuristic performance in large resource-constrained projects // Techn. Rep. 92-2 Departament D'Estadistica I Invecigacio Operativa, Universitat de Valencia. 1992.

[107] Węglarz J. Project Scheduling. Recent Models, Algorithms and Applications. Boston: Kluwer Acad. Publ., 1999.

[108] Whitehouse G., Brown J. GENRES: An extention of Brooks algorithm for project scheduling with resource constraints // Computers & Industrial Engineering. V. 3. P. 261–268.

[109] Yannakakis M. Computational Complexity // Local Search in Combinatorial Optimization. Chichester: John Wiley & Sons, 1997. P. 19–56.

# Приложение А

# Основные приоритетные правила

| | |
|---|---|
| Наибольший позиционный вес (GRPW) | $p_j + \sum_{i \in S_j} p_i$ |
| Наиболее поздний момент завершения (LFT) | $LF_j$ |
| Наиболее поздний момент начала (LST) | $LF_j - p_j$ |
| Минимальный резерв (MSLK) | $LF_j - EF_j$ |
| Максимум всех последователей (MTS) | $\lvert\overline{S}_j\rvert$ |
| Метод распределения ресурсов (RSM) | $\max_{(i,j) \in AP} \{0, t_g + p_j - (LF_i - p_i)\}$ |
| Минимальное время выполнения (SPT) | $p_j$ |
| Резерв в худшем случае (WCS) | $LF_j - p_j - \max_{(i,j) \in AP} \{E(i,j)\}$ |

Таблица А.1: Основные приоритетные правила

Обозначения:

- $EF_j$ — наиболее ранний момент завершения работы $j$
- $LF_j$ — наиболее поздний момент завершения работы $j$
- $\overline{S}_j$ — транзитивное замыкание множества $S_j$
- $AP = \{(i,j) \in D_m \times D_m \mid i \neq j\}$
- $E(i,j)$ — наиболее ранний возможный момент начала работы $j$ при условии, что работа $i$ начата в момент времени $t_m$

# Приложение В

# Относительная погрешность эволюционного алгоритма

В эволюционном алгоритме использовались следующие значения параметров:

| | |
|---|---|
| Размер популяции, $POPSIZE$ | 5 |
| Длина эволюции, $I$ | |
| (число построенных расписаний) | 1000, 5000 и 50000 |
| Выбор порождающей пары | $RF$ для $n = 30,\ 60$ |
| | $RC$ для $n = 120$ |
| Построение начальной популяции | Алгоритм МПР+ВУМ+ПП |
| Глубина локального поиска | $I/100$ для $n = 30$ |
| | $I/10$ для $n = 60$ |
| | $I/5$ для $n = 120$ |
| Коэффициент рандомизации | |
| окрестности, $q$ | 0,2 |
| Интервал чередования окрестности | 5 итераций |
| Длина списка запретов, $TL$ | 5 |

Таблица В.1: Параметры эволюционного алгоритма

| Класс | Значения параметров | | | Критерий остановки | | |
|---|---|---|---|---|---|---|
| | $NC$ | $RF$ | $RS$ | 1000 | 5000 | 50000 |
| J30 1 | 1,5 | 0,25 | 0,2 | 0,00 | 0,00 | 0,00 |
| J30 2 | 1,5 | 0,25 | 0,5 | 0,00 | 0,00 | 0,00 |
| J30 3 | 1,5 | 0,25 | 0,7 | 0,00 | 0,00 | 0,00 |
| J30 4 | 1,5 | 0,25 | 1,0 | 0,00 | 0,00 | 0,00 |
| J30 5 | 1,5 | 0,50 | 0,2 | 0,00 | 0,00 | 0,00 |
| J30 6 | 1,5 | 0,50 | 0,5 | 0,00 | 0,00 | 0,00 |
| J30 7 | 1,5 | 0,50 | 0,7 | 0,00 | 0,00 | 0,00 |
| J30 8 | 1,5 | 0,50 | 1,0 | 0,00 | 0,00 | 0,00 |
| J30 9 | 1,5 | 0,75 | 0,2 | 0,22 | 0,00 | 0,00 |
| J30 10 | 1,5 | 0,75 | 0,5 | 0,00 | 0,00 | 0,00 |
| J30 11 | 1,5 | 0,75 | 0,7 | 0,00 | 0,00 | 0,00 |
| J30 12 | 1,5 | 0,75 | 1,0 | 0,00 | 0,00 | 0,00 |
| J30 13 | 1,5 | 1,00 | 0,2 | 1,39 | 0,36 | 0,00 |
| J30 14 | 1,5 | 1,00 | 0,5 | 0,58 | 0,22 | 0,00 |
| J30 15 | 1,5 | 1,00 | 0,7 | 0,00 | 0,00 | 0,00 |
| J30 16 | 1,5 | 1,00 | 1,0 | 0,00 | 0,00 | 0,00 |
| J30 17 | 1,8 | 0,25 | 0,2 | 0,00 | 0,00 | 0,00 |
| J30 18 | 1,8 | 0,25 | 0,5 | 0,00 | 0,00 | 0,00 |
| J30 19 | 1,8 | 0,25 | 0,7 | 0,00 | 0,00 | 0,00 |
| J30 20 | 1,8 | 0,25 | 1,0 | 0,00 | 0,00 | 0,00 |
| J30 21 | 1,8 | 0,50 | 0,2 | 0,00 | 0,00 | 0,00 |
| J30 22 | 1,8 | 0,50 | 0,5 | 0,00 | 0,00 | 0,00 |
| J30 23 | 1,8 | 0,50 | 0,7 | 0,00 | 0,00 | 0,00 |
| J30 24 | 1,8 | 0,50 | 1,0 | 0,00 | 0,00 | 0,00 |
| J30 25 | 1,8 | 0,75 | 0,2 | 0,81 | 0,25 | 0,00 |
| J30 26 | 1,8 | 0,75 | 0,5 | 0,00 | 0,00 | 0,00 |
| J30 27 | 1,8 | 0,75 | 0,7 | 0,00 | 0,00 | 0,00 |
| J30 28 | 1,8 | 0,75 | 1,0 | 0,00 | 0,00 | 0,00 |
| J30 29 | 1,8 | 1,00 | 0,2 | 1,01 | 0,68 | 0,00 |
| J30 30 | 1,8 | 1,00 | 0,5 | 0,15 | 0,00 | 0,00 |
| J30 31 | 1,8 | 1,00 | 0,7 | 0,00 | 0,00 | 0,00 |
| J30 32 | 1,8 | 1,00 | 1,0 | 0,00 | 0,00 | 0,00 |

Таблица B.2: Относительная погрешность в различных классах, $n = 30$, часть 1

| Класс | Значения параметров | | | Критерий остановки | | |
|---|---|---|---|---|---|---|
| | $NC$ | $RF$ | $RS$ | 1000 | 5000 | 50000 |
| J30 33 | 2,1 | 0,25 | 0,2 | 0,00 | 0,00 | 0,00 |
| J30 34 | 2,1 | 0,25 | 0,5 | 0,00 | 0,00 | 0,00 |
| J30 35 | 2,1 | 0,25 | 0,7 | 0,00 | 0,00 | 0,00 |
| J30 36 | 2,1 | 0,25 | 1,0 | 0,00 | 0,00 | 0,00 |
| J30 37 | 2,1 | 0,50 | 0,2 | 0,00 | 0,00 | 0,00 |
| J30 38 | 2,1 | 0,50 | 0,5 | 0,00 | 0,00 | 0,00 |
| J30 39 | 2,1 | 0,50 | 0,7 | 0,00 | 0,00 | 0,00 |
| J30 40 | 2,1 | 0,50 | 1,0 | 0,00 | 0,00 | 0,00 |
| J30 41 | 2,1 | 0,75 | 0,2 | 0,21 | 0,00 | 0,00 |
| J30 42 | 2,1 | 0,75 | 0,5 | 0,15 | 0,15 | 0,00 |
| J30 43 | 2,1 | 0,75 | 0,7 | 0,00 | 0,00 | 0,00 |
| J30 44 | 2,1 | 0,75 | 1,0 | 0,00 | 0,00 | 0,00 |
| J30 45 | 2,1 | 1,00 | 0,2 | 0,34 | 0,22 | 0,00 |
| J30 46 | 2,1 | 1,00 | 0,5 | 0,00 | 0,00 | 0,00 |
| J30 47 | 2,1 | 1,00 | 0,7 | 0,00 | 0,00 | 0,00 |
| J30 48 | 2,1 | 1,00 | 1,0 | 0,00 | 0,00 | 0,00 |

Таблица В.3: Относительная погрешность в различных классах, $n = 30$, часть 2

| Класс | Значения параметров | | | Критерий остановки | | |
|---|---|---|---|---|---|---|
| | $NC$ | $RF$ | $RS$ | 1000 | 5000 | 50000 |
| J60 1 | 1,5 | 0,25 | 0,2 | 10,30 | 10,15 | 9,98 |
| J60 2 | 1,5 | 0,25 | 0,5 | 2,41 | 2,41 | 2,41 |
| J60 3 | 1,5 | 0,25 | 0,7 | 1,98 | 1,98 | 1,98 |
| J60 4 | 1,5 | 0,25 | 1,0 | 0,00 | 0,00 | 0,00 |
| J60 5 | 1,5 | 0,50 | 0,2 | 32,07 | 30,01 | 28,56 |
| J60 6 | 1,5 | 0,50 | 0,5 | 1,03 | 1,03 | 0,70 |
| J60 7 | 1,5 | 0,50 | 0,7 | 0,00 | 0,00 | 0,00 |
| J60 8 | 1,5 | 0,50 | 1,0 | 0,00 | 0,00 | 0,00 |
| J60 9 | 1,5 | 0,75 | 0,2 | 45,25 | 43,87 | 42,13 |
| J60 10 | 1,5 | 0,75 | 0,5 | 0,16 | 0,16 | 0,16 |
| J60 11 | 1,5 | 0,75 | 0,7 | 0,00 | 0,00 | 0,00 |
| J60 12 | 1,5 | 0,75 | 1,0 | 0,00 | 0,00 | 0,00 |
| J60 13★ | 1,5 | 1,00 | 0,2 | 73,83 | 71,29 | 68,68 |
| J60 14 | 1,5 | 1,00 | 0,5 | 1,70 | 1,55 | 1,40 |
| J60 15 | 1,5 | 1,00 | 0,7 | 0,00 | 0,00 | 0,00 |
| J60 16 | 1,5 | 1,00 | 1,0 | 0,00 | 0,00 | 0,00 |

Таблица В.4: Относительная погрешность в различных классах, $n = 60$, часть 1

| Класс | Значения параметров | | | Критерий остановки | | |
|---|---|---|---|---|---|---|
| | $NC$ | $RF$ | $RS$ | 1000 | 5000 | 50000 |
| J60 17 | 1,8 | 0,25 | 0,2 | 10,32 | 10,32 | 9,87 |
| J60 18 | 1,8 | 0,25 | 0,5 | 0,81 | 0,81 | 0,81 |
| J60 19 | 1,8 | 0,25 | 0,7 | 1,30 | 1,30 | 1,30 |
| J60 20 | 1,8 | 0,25 | 1,0 | 0,00 | 0,00 | 0,00 |
| J60 21 | 1,8 | 0,50 | 0,2 | 36,70 | 35,39 | 33,84 |
| J60 22 | 1,8 | 0,50 | 0,5 | 2,03 | 2,03 | 1,87 |
| J60 23 | 1,8 | 0,50 | 0,7 | 0,40 | 0,40 | 0,40 |
| J60 24 | 1,8 | 0,50 | 1,0 | 0,00 | 0,00 | 0,00 |
| J60 25 | 1,8 | 0,75 | 0,2 | 53,12 | 51,41 | 48,58 |
| J60 26 | 1,8 | 0,75 | 0,5 | 2,86 | 2,40 | 2,27 |
| J60 27 | 1,8 | 0,75 | 0,7 | 0,00 | 0,00 | 0,00 |
| J60 28 | 1,8 | 0,75 | 1,0 | 0,00 | 0,00 | 0,00 |
| J60 29★ | 1,8 | 1,00 | 0,2 | 73,74 | 71,84 | 69,26 |
| J60 30 | 1,8 | 1,00 | 0,5 | 4,03 | 3,89 | 3,49 |
| J60 31 | 1,8 | 1,00 | 0,7 | 0,00 | 0,00 | 0,00 |
| J60 32 | 1,8 | 1,00 | 1,0 | 0,00 | 0,00 | 0,00 |
| J60 33 | 2,1 | 0,25 | 0,2 | 10,79 | 10,65 | 10,65 |
| J60 34 | 2,1 | 0,25 | 0,5 | 2,30 | 2,30 | 2,30 |
| J60 35 | 2,1 | 0,25 | 0,7 | 1,32 | 1,32 | 1,32 |
| J60 36 | 2,1 | 0,25 | 1,0 | 0,00 | 0,00 | 0,00 |
| J60 37 | 2,1 | 0,50 | 0,2 | 36,99 | 36,33 | 35,32 |
| J60 38 | 2,1 | 0,50 | 0,5 | 2,03 | 1,88 | 1,75 |
| J60 39 | 2,1 | 0,50 | 0,7 | 0,12 | 0,12 | 0,12 |
| J60 40 | 2,1 | 0,50 | 1,0 | 0,00 | 0,00 | 0,00 |
| J60 41 | 2,1 | 0,75 | 0,2 | 58,82 | 57,06 | 55,23 |
| J60 42 | 2,1 | 0,75 | 0,5 | 3,30 | 3,15 | 3,03 |
| J60 43 | 2,1 | 0,75 | 0,7 | 0,14 | 0,14 | 0,14 |
| J60 44 | 2,1 | 0,75 | 1,0 | 0,00 | 0,00 | 0,00 |
| J60 45★ | 2,1 | 1,00 | 0,2 | 75,76 | 74,79 | 71,88 |
| J60 46 | 2,1 | 1,00 | 0,5 | 7,30 | 6,63 | 5,56 |
| J60 47 | 2,1 | 1,00 | 0,7 | 0,00 | 0,00 | 0,00 |
| J60 48 | 2,1 | 1,00 | 1,0 | 0,00 | 0,00 | 0,00 |

Таблица B.5: Относительная погрешность в различных классах, $n = 60$, часть 2

| Класс | Значения параметров | | | Критерий остановки | | |
|---|---|---|---|---|---|---|
| | $NC$ | $RF$ | $RS$ | 1000 | 5000 | 50000 |
| J120 1 | 1,5 | 0,25 | 0,1 | 28,42 | 27,64 | 27,54 |
| J120 2 | 1,5 | 0,25 | 0,2 | 15,25 | 14,65 | 13,62 |
| J120 3 | 1,5 | 0,25 | 0,3 | 0,75 | 0,62 | 0,50 |
| J120 4 | 1,5 | 0,25 | 0,4 | 1,60 | 1,60 | 1,60 |
| J120 5 | 1,5 | 0,25 | 0,5 | 0,00 | 0,00 | 0,00 |
| J120 6 | 1,5 | 0,50 | 0,1 | 88,90 | 86,91 | 84,16 |
| J120 7 | 1,5 | 0,50 | 0,2 | 37,80 | 36,46 | 34,47 |
| J120 8 | 1,5 | 0,50 | 0,3 | 11,26 | 10,33 | 9,31 |
| J120 9 | 1,5 | 0,50 | 0,4 | 1,13 | 1,00 | 1,00 |
| J120 10 | 1,5 | 0,50 | 0,5 | 0,00 | 0,00 | 0,00 |
| J120 11 | 1,5 | 0,75 | 0,1 | 121,01 | 119,97 | 118,22 |
| J120 12 | 1,5 | 0,75 | 0,2 | 60,99 | 60,58 | 59,54 |
| J120 13 | 1,5 | 0,75 | 0,3 | 18,68 | 18,02 | 17,49 |
| J120 14 | 1,5 | 0,75 | 0,4 | 6,62 | 6,18 | 5,85 |
| J120 15 | 1,5 | 0,75 | 0,5 | 0,00 | 0,00 | 0,00 |
| J120 16★ | 1,5 | 1,00 | 0,1 | 159,22 | 156,41 | 154,50 |
| J120 17 | 1,5 | 1,00 | 0,2 | 67,00 | 66,02 | 64,93 |
| J120 18 | 1,5 | 1,00 | 0,3 | 30,21 | 29,76 | 29,33 |
| J120 19 | 1,5 | 1,00 | 0,4 | 10,83 | 10,60 | 10,60 |
| J120 20 | 1,5 | 1,00 | 0,5 | 0,88 | 0,77 | 0,77 |
| J120 21 | 1,8 | 0,25 | 0,1 | 25,02 | 24,54 | 23,44 |
| J120 22 | 1,8 | 0,25 | 0,2 | 12,51 | 11,84 | 11,84 |
| J120 23 | 1,8 | 0,25 | 0,3 | 1,66 | 1,19 | 1,19 |
| J120 24 | 1,8 | 0,25 | 0,4 | 1,17 | 1,17 | 1,17 |
| J120 25 | 1,8 | 0,25 | 0,5 | 0,75 | 0,75 | 0,75 |
| J120 26 | 1,8 | 0,50 | 0,1 | 80,43 | 77,76 | 76,40 |
| J120 27 | 1,8 | 0,50 | 0,2 | 39,26 | 37,91 | 37,07 |
| J120 28 | 1,8 | 0,50 | 0,3 | 8,05 | 7,43 | 6,81 |
| J120 29 | 1,8 | 0,50 | 0,4 | 3,59 | 3,46 | 3,35 |
| J120 30 | 1,8 | 0,50 | 0,5 | 0,72 | 0,48 | 0,48 |

Таблица B.6: Относительная погрешность в различных классах, $n = 120$, часть 1

| Класс | Значения параметров | | | Критерий остановки | | |
|---|---|---|---|---|---|---|
| | $NC$ | $RF$ | $RS$ | 1000 | 5000 | 50000 |
| J120 31 | 1,8 | 0,75 | 0,1 | 125,65 | 123,12 | 121,32 |
| J120 32 | 1,8 | 0,75 | 0,2 | 47,11 | 46,25 | 44,83 |
| J120 33 | 1,8 | 0,75 | 0,3 | 23,56 | 22,65 | 22,24 |
| J120 34 | 1,8 | 0,75 | 0,4 | 8,13 | 7,26 | 7,03 |
| J120 35 | 1,8 | 0,75 | 0,5 | 0,86 | 0,64 | 0,64 |
| J120 36★ | 1,8 | 1,00 | 0,1 | 144,54 | 141,97 | 139,58 |
| J120 37 | 1,8 | 1,00 | 0,2 | 73,39 | 72,00 | 70,88 |
| J120 38 | 1,8 | 1,00 | 0,3 | 29,47 | 28,54 | 28,13 |
| J120 39 | 1,8 | 1,00 | 0,4 | 12,13 | 11,82 | 11,52 |
| J120 40 | 1,8 | 1,00 | 0,5 | 0,76 | 0,51 | 0,51 |
| J120 41 | 2,1 | 0,25 | 0,1 | 27,81 | 26,56 | 25,49 |
| J120 42 | 2,1 | 0,25 | 0,2 | 12,42 | 11,91 | 11,80 |
| J120 43 | 2,1 | 0,25 | 0,3 | 4,99 | 4,68 | 4,68 |
| J120 44 | 2,1 | 0,25 | 0,4 | 1,91 | 1,91 | 1,91 |
| J120 45 | 2,1 | 0,25 | 0,5 | 0,18 | 0,18 | 0,18 |
| J120 46 | 2,1 | 0,50 | 0,1 | 84,77 | 82,94 | 80,98 |
| J120 47 | 2,1 | 0,50 | 0,2 | 40,01 | 37,87 | 37,09 |
| J120 48 | 2,1 | 0,50 | 0,3 | 16,78 | 15,56 | 15,06 |
| J120 49 | 2,1 | 0,50 | 0,4 | 5,78 | 5,65 | 5,46 |
| J120 50 | 2,1 | 0,50 | 0,5 | 0,96 | 0,74 | 0,74 |
| J120 51 | 2,1 | 0,75 | 0,1 | 141,64 | 138,03 | 136,05 |
| J120 52 | 2,1 | 0,75 | 0,2 | 62,79 | 61,13 | 59,81 |
| J120 53 | 2,1 | 0,75 | 0,3 | 23,91 | 23,04 | 22,36 |
| J120 54 | 2,1 | 0,75 | 0,4 | 10,92 | 10,33 | 9,92 |
| J120 55 | 2,1 | 0,75 | 0,5 | 1,01 | 0,91 | 0,81 |
| J120 56★ | 2,1 | 1,00 | 0,1 | 171,09 | 169,10 | 166,60 |
| J120 57 | 2,1 | 1,00 | 0,2 | 72,74 | 71,55 | 70,26 |
| J120 58 | 2,1 | 1,00 | 0,3 | 36,33 | 35,95 | 35,14 |
| J120 59 | 2,1 | 1,00 | 0,4 | 15,07 | 14,87 | 14,18 |
| J120 60 | 2,1 | 1,00 | 0,5 | 5,72 | 5,60 | 5,38 |

Таблица B.7: Относительная погрешность в различных классах, $n = 120$, часть 2