МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НОВОСИБИРСКИЙ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Механико-математический факультет

Кафедра теоретической кибернетики

ЛЕОНОВА Валентина Вячеславовича

Выпускная квалификационная работа магистерская диссертация

Название работы:

# Генетический алгоритм для задачи календарного планирования с ограниченными ресурсами

Научный руководитель:

к.ф.-м.н., Гончаров Евгений Николаевич

Новосибирск - 2014

# Оглавление

# Введение

Работа посвящена задаче календарного планирования в условиях ограниченных ресурсов. Данная задача имеет важное практическое значение, она возникает во многих областях человеческой деятельности, связанной с планированием совокупностей работ, в частности, при планировании территориально-промышленных комплексов, сооружении сложных строительных объектов, разработке новых технических систем и изделий, изготовлении и сборке крупных изделий (космические корабли, суда, самолеты), проведении крупных ремонтов и реконструкций и т.д. В общей постановке задача NP-трудна в сильном смысле [2]. В связи с высокой прикладной значимостью задаче календарного планирования уделяется большое внимание, существует обширная литература. Для решения данной задачи разработаны как точные методы, основанные на идеях метода ветвей и границ, так и приближенные [3]. Кроме того, для любого $\varepsilon > 0$ маловероятно существование приближенных полиномиальных алгоритмов с гарантированной оценкой точности $n^{1-\varepsilon}$, где $n$ — размерность задачи. Таким образом, разработка метаэвристик является, по-видимому, наиболее перспективным направлением для решения данной задачи [3].

Общая постановка задачи подразумевает наличие трех типов ресурсов: возобновляемые, невозобновляемые и складируемые. Для задачи с ресурсами только складируемого типа удается построить малотрудоемкий асимптотически точный алгоритм, в случае целочисленных длительностей работ задача разрешима за полиномиальное время [1]. В данной работе рассматривается постановка задачи с ресурсами только возобновляемого типа. Такая задача NP-трудна [2].

Будет предложен генетический алгоритм и приведены результаты численных экспериментов. Алгоритм в своей основе опирается на поиск блоков работ, выполнение которых одновременно приводит к меньшему остатку неиспользованных ресурсов.

# Глава 1

## Постановка задачи и представление решений

В данной главе будет дано описание задачи и ее математическая постановка. Также опишем вид, в котором будем представлять решения — кодирование решений и укажем декодирующие процедуры — как из данной кодировки получить расписание.

## 1.1 Постановка задачи

Приведем математическую постановку для задачи календарного планирования в условиях ограниченных ресурсов. Обозначим через $J = \Big\{1, ..., n\Big\} \cup \Big\{0, n+1\Big\}$ множество работ, в котором работы 0 и $n+1$ фиктивны и задают начало и завершение всего комплекса работ. Длительность работы $j$ обозначим через $p_j$. Считаем, что $p_j \geqslant 0$, $j \in J$ — целые неотрицательные числа и $p_0 = p_{n+1} = 0$. Отношение предшествования на множестве $J$ зададим множеством пар $C = \Big\{(i,j) \;\Big|\; i$ — предшественник $j\Big\}$. Если $(i,j) \in C$, то работа $j$ не может начаться раньше завершения работы $i$. В множестве $C$ содержатся все пары $(0,j)$ и $(j, n+1)$, $j = 1, ..., n$. Через $P_j$ обозначим множество непосредственных предшественников работы $j$.

Через $K = \Big\{1, ..., \omega\Big\}$ обозначим множество ресурсов, необходимых для выполнения рассматриваемых работ. Все ресурсы предполагаются возобновляемыми, т. е. в каждый момент времени выделяется фиксированный объем ресурсов каждого типа, а остатки ресурсов пропадают. Будем считать, что объем выделяемого ресурса $R_k > 0$, $k \in K$, есть величина постоянная. Через $r_{j,k} \geqslant 0$ обозначим объем $k$-го ресурса, необходимый для выполнения $j$-й работы.

Введем переменные задачи. Через $s_j \geqslant 0$ обозначим момент начала выполнения $j$-й работы. Предполагаем, что работы выполняются без прерывания, и момент окончания $j$-й работы определяется равенством $c_j = s_j + p_j$ . Через $A(t) = \Big\{j \in J \;\Big|\; s_j \leqslant t < c_j\Big\}$ обозначим

множество работ, выполняемых в момент времени $t$. Время завершения проекта обозначим через $T(S)$. Эта величина соответствует моменту завершения последней работы проекта, т. е. $T(S) = c_{n+1}$. При этих обозначениях задача календарного планирования с ограниченными ресурсами записывается следующим образом.

$$T(S) \longrightarrow \min$$

$$c_i \leqslant s_j,\ \forall (i,j) \in C$$

$$\sum_{j \in A(t)} r_{j,k} \leqslant R_k,\ k \in K,\ t \geqslant 0$$

$$s_j \geqslant 0,\ j \in J$$

Целевая функция задает время завершения всего комплекса работ. Первое неравенство задает условия предшествования. Второе неравенство требует выполнения ресурсных ограничений.

## 1.2 Кодировка и декодирующие процедуры

Кодировки решений имеют важное значение при разработке оптимизационных алгоритмов. Ниже приводятся кодировка и декодирующие процедуры, использующиеся в алгоритме, предложенном в данной работе. В большинстве работ, в которых изучаются алгоритмы решения ЗКПОР, используется именно такая кодировка [3].

### 1.2.1 Кодировка решений

Рассмотривается кодировка решения в виде списка работ $L = (j_1, ..., j_n)$. Предполагается, что списки согласованы с условиями предшествования, т. е. для любой пары $(i, j) \in C$ позиция работы $j$ больше позиции работы $i$.

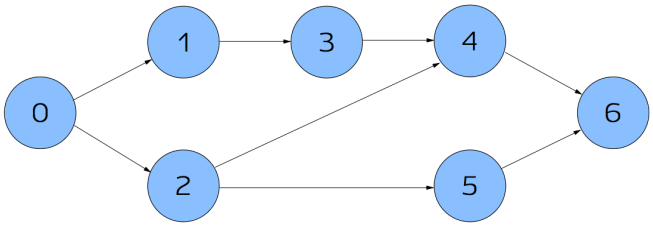


Рис. 1.1: Частичный порядок

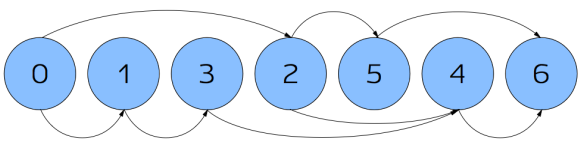


Рис. 1.2: Закодированное решение

По списку работ можно построить расписание. Ниже будут приведены три алгоритма построения расписаний работ с использованием последовательного, $T$-позднего и параллельного декодеров. Первый декодер согласно списку $L$ последовательно вычисляет наиболее раннее время начала выполнения каждой работы, учитывая ресурсные ограничения. $T$-поздний декодер действует аналогично первому, но использует список в обратном порядке. Фиксируется длина расписания $T$, и с учетом ресурсных ограничений для каждой работы вычисляется наиболее позднее время ее окончания. Как уже было сказано выше, условия предшествования учитываются при составлении списка и выполняются автоматически.

### 1.2.2 Декодеры

Декодирующие процедуры строят расписание по заданной кодировке. Далее будет подразумеваться кодировка в виде списка работ. Рассмотрим три декодера: последовательный, $T$-поздний и параллельный.

#### Последовательный декодер

На $m$-м шаге поледовательный декодер очередной работе в списке устанавливает момент начала ее выполнения как наиболее ранний из возможных с учетом условий предшествования и ограничений по ресурсам.

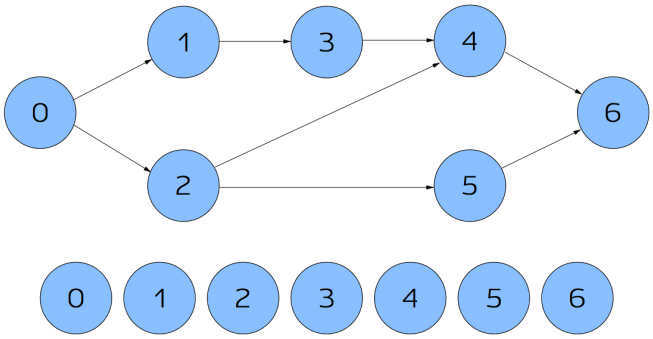


Рис. 1.3: Закодированное решение

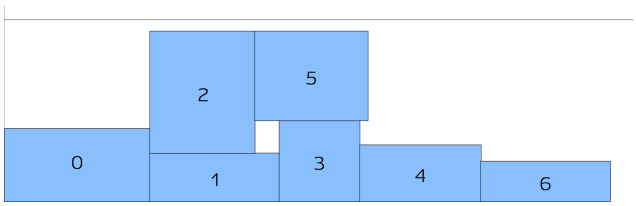


Рис. 1.4: Результат применения последовательного декодера

$SerialDecoder(L) \to S$

1. Полагается $s_0 := 0$, $c_0 := 0$.

2. Для всех $m = 1, ..., n + 1$ выполняем:

   2.1. Для $m$-й работы в списке $L$ находится

$$t_m = max\left\{c_i \;\middle|\; (i, j_m) \in C, i \in J\right\}.$$

2.2. Определяется минимальное время $t > t_m$ такое, что работа $j_m$ может выполняться без нарушения ресурсных ограничений.

2.3. Полагается $s_{j_m} := t, c_{j_m} := t + p_{j_m}$.

3. Вычисляется $T(S) = c_{n+1}$.

Шаг 2.2 требует $O(n\omega)$ операций, если в каждый момент времени $c_{j_m}$ хранится объем использованных ресурсов. Таким образом, суммарная временная сложность процедуры не превосходит величины $O(n^2\omega)$.

Последовательный декодер строит расписание, в котором ни одна работа не может начаться раньше установленного ей срока без нарушений условий предшествования или ограничений по ресурсам. Такие расписания называются активными. Известно, что среди активных расписаний есть и оптимальное расписание [6].

### $T$-поздний декодер

Наряду с активными расписаниями будем рассматривать $T$-поздние расписания. Пусть $T$ — время окончания $(n+1)$-й работы. Приведем алгоритм построения расписания, в котором ни одна работа не может закончиться позже установленного ей срока без нарушений либо условий предшествования, либо ограничений по ресурсам.

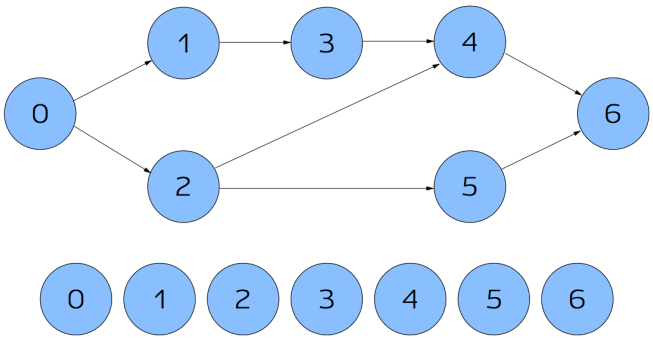


Рис. 1.5: Закодированное решение

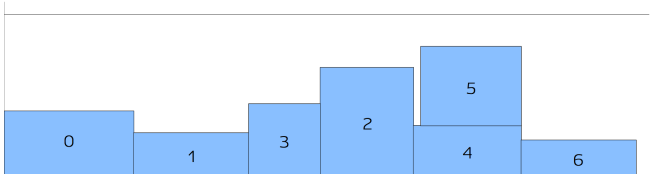


Рис. 1.6: Результат применения $T$-позднего декодера

$LateDecoder(L, T) \to S$

1. Полагается $s_{n+1} := T$, $c_{n+1} := T$.

2. Для всех $m = n, ..., 0$ выполняем:

2.1. Для $m$-й работы в списке $L$ находится

$$t_m = min\left\{ s_i \;\middle|\; (j_m, i) \in C, i \in J \right\} .$$

2.2. Определяется наиболее позднее время $t \leqslant t_m$ такое, что работа $j_m$ может выполняться без нарушения ресурсных ограничений.

2.3. Полагается $s_{j_m} := t - d_{j_m}, c_{j_m} := t$.

3. Вычисляется $T(S) = T - s_0$.

Сложность процедуры также оценивается величиной $O(n^2\omega)$.

### Параллельный декодер

Параллельный декодер имеет существенное отличие от описанных выше. Если раньше последовательно рассматривались работы, и для каждой из них вычислялся момент начала ее выполнения, то теперь последовательно рассматриваются возрастающие моменты времени, и для каждого из них определяются работы, начинающие выполняться в данный момент времени. Итак, на каждом шаге $m = 1, ..., n$ имеется некоторый момент времени $t_m$, множество $J(t_m) = \left\{ j \in J \;\middle|\; c_j \leqslant t_m \right\}$ уже завершенных работ и множество $A(t_m)$ работ, находящихся в процессе выполнения. По множеству $A(t_m)$ определяются остаточные объемы ресурсов $\widetilde{R}_k(t_m) = R_k - \sum\limits_{j \in A(t_m)} r_{i,k},\ k \in K$, и допустимое множество $D(t_m) = \Big\{ j \in J \setminus (A(t_m) \cup J(t_m)) \;\Big|\; P_j \subseteq J(t_m),\ r_{i,k} \leqslant \widetilde{R}_k(t_m),\ k \in K \Big\}$ работ, которые могут начаться в момент времени $t_m$. Шаг состоит в том, что из множества $D(t_m)$ выбирается работа с минимальной позицией в списке и она начинает выполняться в момент времени $t_m$. Формально параллельный декодер может быть представлен следующим образом.

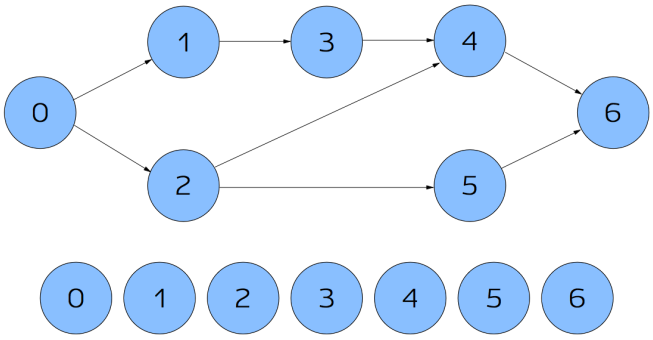


Рис. 1.7: Закодированное решение

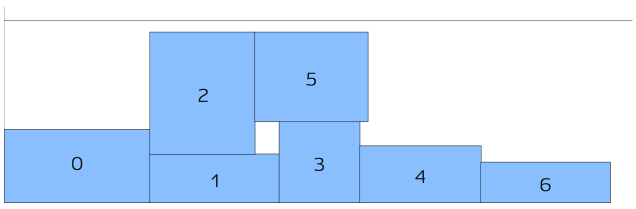


Рис. 1.8: Результат применения параллельного декодера

$ParallelDecoder(L) \to S$

1. Полагается $m := 0$, $t_m := 0$, $A(0) := \left\{0\right\}$, $J(0) := \varnothing$, $\widetilde{R}_k(0) := R_k$, $s_0 := c_0 := 0$.

2. Пока $\left|A(t_m) \cup J(t_m)\right| \leqslant n+1$ выполняем:

   2.1. Полагается $m := m+1$ и $t_m = \min\left\{c_j \;\middle|\; j \in A(t_{m-1})\right\}$.

   2.2. Вычисляется $J(t_m)$, $A(t_m)$, $\widetilde{R}_k(t_m)$, $D(t_m)$.

   2.3. Пока $D(t_m) \neq \varnothing$ проделывается следующее:

      2.3.1. Выбирается из списка $L$ работа $j \in D(t_m)$.

      2.3.2. Полагается $s_j := t_m$, $c_j := s_j + p_j$.

      2.3.3. Обновляется $A(t_m)$, $\widetilde{R}_k(t_m)$, $D(t_m)$.

3. Вычисляется $T(S) = c_{n+1}$.

Сложность данной процедуры оценивается величиной $O(n^2\omega)$.

# Глава 2

## Процедуры локального поиска

Опишем процедуры локального поиска, использующиеся в данной работе — это процедура Пинг-Понг — довольно известная процедура улучшения некоторого решения [3] [4], и процедура локального поиска с чередующимися окрестностями, взятая за основу в алгоритме, разработанном Ю.А. Кочетовым и А.А. Столяром [3].

## 2.1 Процедура Пинг-Понг

Процедура Пинг-Понг (в англоязычной литературе Forward-Backward) довольно "дешевая"(в смысле времени выполнения) процедура, которая дает неплохое улучшение решения, по крайней мере, на случайном списке работ $L = (j_1, ..., j_n)$.

Суть процедуры в следующем. Пусть дано некоторое решение $S$. Сначала положим $T := T(S)$ и по расписанию $S$ построим список $L_1$, упорядочив работы по времени их окончания, $c_{j_0} \leqslant c_{j_1} \leqslant ... \leqslant c_{j_{n+1}}$. К полученному списку применим T-поздний декодер. Получим расписание $S_1$. Построим список $L_2$, упорядочив работы по началу их выполнения $s_{j_0} \leqslant s_{j_1} \leqslant ... \leqslant s_{j_{n+1}}$, и применим последовательный декодер. Получим расписание $S_2$. Если $T > T(S_2)$, то повторим процедуру с T-поздним и последовательным декодерами.

$PingPong(S) \to S^*$

1. Полагается $T^* := s_{n+1}$, $S^* := S$

2. Работы упорядочиваются так, что $c_{j_0} \leqslant c_{j_1} \leqslant ... \leqslant c_{j_{n+1}}$, и строится T-позднее расписание $S'$.

3. Работы упорядочиваются так, что $s_{j_0} \leqslant s_{j_1} \leqslant ... \leqslant s_{j_{n+1}}$, и строится последовательное расписание $S''$.

4. Если $T^* > s_{n+1}$, то полагается $T^* := s_{n+1}$ и $S^* := S''$ и переход к шагу 2.

Каждый шаг этой процедуры является полиномиальным по числу выполняемых операций. Однако число возвращений на шаг 2 может оказаться очень большим. Оценить это число сверху полиномом от длины записи исходных данных не представляется возможным. Тем не менее, численные эксперименты показывают малое число возвращений на шаг 2. Как правило, это число значительно меньше $n$ [3].

## 2.2 Процедура локального поиска с чередующимися окрестностями

Приведенная ниже процедура локального поиска с чередующимися окрестностями, хорошо себя показала в алгоритме предложенном Ю.А.Кочетовым и А.А.Столяром в 2003 году [3].

### 2.2.1 Окрестности

Рассматриваются два типа окрестностей. Первая окрестность строится по активным расписаниям, вторая — по $T$-поздним. Активные и $T$-поздние расписания удачно дополняют друг друга. Переход от одной окрестности к другой привносит определенное разнообразие в локальный поиск, что благотворно сказывается на результатах работы алгоритма.

**Окрестность $N_A(S)$ для активного расписания $S$**

Блоком работы $j$ в активном расписании $S$ будем называть множество работ, которые выполняются параллельно с работой $j$, либо начинают выполняться сразу за работой $j$, либо заканчивают непосредственно перед ней: $B_j = \left\{ i \in J \;\middle|\; [s_i, c_i] \cap [s_j, c_j] \neq \varnothing \right\}$.

Граф $G_S$ — орграф с множеством вершин $V = J$ и множеством дуг $E = \left\{ (i,j) \;\middle|\; c_i = s_j, (i,j) \in C \right\}$

Исходящей сетью работы $j$ для расписания $S$ будем называть максимальный по включению связный подграф графа $G_S$, в котором единственным источником 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.

Пусть активное расписание $S$ получено по списку $L$ последовательным декодером, а блок работы $j$ не содержит ее предшественников, т.е. $B_j \cap P_j = \varnothing$. Выделим множество работ, время которых следует оптимизировать, если меняются сроки выполнения работы $j$. Это множество определим по сегменту $L(j)$ списка $L$. Началом сегмента является работа блока $B_j$ с минимальной позицией в списке L. Конец сегмента определяется как максимум из двух чисел: максимальных позиций в списке $L$ для работ блока $B_j$ и исходящей сети. Итак, по работе $j$ выделен сегмент $L(j)$, для которого будут определяться новые сроки их выполнения.

Представим список $L$ в виде трех последовательных списков $L = L', L(j), L''$. Для работ списка из $L'$ сохраним старые сроки их выполнения, а для работ из списка $L(j)$ 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. Введем переменные $x_i \in \Big\{0, 1\Big\},\ i \in D(t_m)$. 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

$$\underset{i\in\{0,1\}}{max} \sum_{i\in D(t_m)} x_i \sum_{k\in K} \frac{r_{i,k}}{R_k}$$

при ограничениях

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

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 разделе ниже.

С помощью указанной модификации параллельного декодера получаем новое расписание работ из списка $L(j)$. Последовательным декодером частичное расписание достраивается до расписания $S_A(j)$ всех работ множества $J$. Расписание $S_A(j)$ назовем соседним расписанием для $S$. Множество всех соседних расписаний назовем окрестностью расписания $S$ и обозначим через $N_A(S)$.

**Окрестность $N_T(S)$ для $T$-позднего расписания $S$**

Входящей сетью работы $j$ для $T$-позднего расписания $S$ будем называть максимальный по включению связный подграф графа $G_S$, в котором единственным стоком является вершина, соответствующая работе $j$.

Пусть расписание $S$ получено по списку $L$ применением $T$-позднего декодера, и блок работы $j$ не содержит ее последователей, т.е.

$$B_j \cap P_i \neq \varnothing,\ (j,i) \in C$$

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, $L = L', L(j), L''$. Сохраним старые сроки выполнения работ из 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 $L'$ времена выполнения определяются $T$-поздним декодером. Полученное расписание назовем соседним для $S$ и обозначим через $S_T(j)$. Множество всех соседних расписаний назовем окрестностью $T$-позднего расписания $S$ и обозначим через $N_T(S)$.

**Задача о многомерном рюкзаке**

Известно, что задача о многомерном рюкзаке NP-трудна даже при $\omega = 1$. В представленном алгоритме будет рассмотрен вероятностный жадный алгоритм, который позволяет быстро получать для нее приближенные решения.

Зададимся величиной $0 < q \leqslant 1$ и сформулируем 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 $\sum\limits_{k \in K} \frac{r_{i,k}}{R_k}, i \in D(q)$. Значение соответствующей переменной $x_i$ положим равным 1, уменьшим правые части ограничений и удалим из $D(t_m)$ выбранный элемент и все элементы, которые уже не помещаются в "рюкзак". Эту процедуру будем повторять до тех пор, пока множество $D(t_m)$ не станет пустым. На выходе имеем набор значений булевых переменных $x_i \in \Big\{0,1\Big\}, i \in D(t_m)$. Формально этот алгоритм

может быть представлен следующим образом.

$KP(q, t, A(t_m), D(t_m))$

1. Полагается $\bar{D} := D(t_m)$, $\widetilde{R}_k := R_k - \sum\limits_{i \in A(t_m)} r_{i,k}$, $x_i := 0$, $i \in \bar{D}$.
2. Пока $\bar{D} \neq \varnothing$ проделывается следующее:
   2.1. Выбирается в $\bar{D}$ случайным образом подмножество $D(q)$. Если $D(q) = \varnothing$, то к $D(q)$ присоединяется любой элемент из $\bar{D}$.
   2.2. Находится $i_0 \in D(q)$ с максимальным весом $\sum\limits_{k \in K} \frac{r_{i_0,k}}{R_k} = \max\limits_{i \in D(q)} \sum\limits_{k \in K} \frac{r_{i,k}}{R_k}$.
   2.3. Полагается $x_{i_0} := 1$; $\bar{D} := \bar{D} \setminus \left\{ i_0 \right\}$, $\widetilde{R}_k := \widetilde{R}_k - r_{i_0,k}$, $k \in K$.
   2.4. $\forall i \in \bar{D}$, если $r_{i,k} > \widetilde{R}_k$ для некоторого $k \in K$, то полагается $\bar{D} := \bar{D} \setminus \left\{ i \right\}$.

Временная сложность алгоритма не превосходит величины $O(\left|D(t_m)\right|(\omega + \left|D(t_m)\right|))$.

### 2.2.2 Локальный поиск с запретами

Приведем общую схему алгоритма. На каждом шаге процедуры имеется некоторое расписание $S$ и значения функции $f(S) = \sum\limits_{j \in J} s_j$ от расписаний, полученных на посделних $h$ шагах алгоритма. Набор таких значений будем называть списком запретов. Шаг 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, в 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 окрестность может оказаться пустой в результате удаления "запрещенных" элементов. В этом случае длина списка запретов уменьшается. Если подмножество $N'_A(S)$ ($N'_T(S)$) оказалось пустым, то в него добавляется произвольный незапрещенный элемент из окрестности $N_A(S)$ ($N_T(S)$). Формально алгоритм можно представить следующим образом.

$TabuSearch(S) \to S^*$

1. Полагается $T^* := T(S)$, $S^* := S$.

2. Пока не выполнен критерий остановки проделывается следующее.

   2.1. Выбирается окрестность.

   2.2. Находится соседнее расписание $S'$.

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

   2.4. Обновляется список запретов и полагается $S := S'$.

В качестве критерия остановки используется максимально число итераций. Смена окрестности производится через заданное число итераций.

# Глава 3

## Алгоритм локального поиска с чередующимися окрестностями

В данной главе будут описан алгоритм, разработанный Ю.А.Кочетовым и А.А.Столяром в 2003г, основанный на идее чередующихся окрестностей и списке запретов.

Основные идеи и процедуры описаны выше, поэтому перейдем сразу к формулировке алгоритма.

## 3.1 Формальное описание алгоритма

Опишем процесс построения начального расписания. Среди допустимых списков выберем список $L_0$, в котором для любой соседней пары работ $(j_m, j_{m+1})$, не принадлежащей множеству $C$, справедливо неравенство $\sum\limits_{k\in K} \frac{r_{j_m,k}}{R_k} \geqslant \sum\limits_{k\in K} \frac{r_{j_{m+1},k}}{R_k}$. Множество таких списков непусто. Одно из них можно построить, например, упорядочив работы по рангам сети, порожденной частичным порядком $C$, а работы с равными рангами — по весу. К построенному списку $L_0$ применим рандомизированный вариант параллельного декодера. На шаге 2.3 вместо множества $D(t_m)$ будем использовать его непустое подмножество, выбранное случайным образом. В результате получим некоторое расписание $S_0$. Далее применим к нему процедуру локального улучшения Пинг-Понг (см. п. 2.1)

Формально алгоритм можно записать следующим образом.

1. Строится начальное расписание $S$.

2. Применяется процедура локального поиска с запретами $S = TabuSearch(S)$.

# Глава 4

## Генетический алгоритм

В данной главе будут описан генетический алгоритм для ЗКПОР, основная идея которого состоит в создании дочерних расписаний на основе “удачных” блоков работ родительских расписаний.

Генетический алгоритм (genetic algorithm) — это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, напоминающих биологическую эволюцию. Отличительной особенностью генетического алгоритма является акцент на использование оператора “скрещивания”, который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе.

## 4.1 Начальная популяция

Опишем процесс создания одного расписания для начальной популяции. Сначала строится случайный список работ, по полученному списоку, с помощью параллельного декодера, строится расписание. Затем к полученному расписанию применяется процедура локального улечшения Пинг-Понг. Описанная процедура повторяется заданное число раз, в качестве итогового расписания выбраем лучшее из полученных.

$InitialPopulation(N) \to \Gamma$

1. Положить $\Gamma := \varnothing$.
2. Пока $\left|\Gamma\right| < N$ делать следующее:
   2.1. Задать рекорд $S^* = \infty$.

2.2. Зачанное число раз проделать следующее:

2.2.1. Построить случайный список работ $L$.

2.2.2. Построить расписание $S := ParallelDecoder(L)$.

2.2.3. Применить Пинг-Понг $S := PingPong(S)$.

2.2.4. Если $S$ лучше рекорда $S^*$, то поменять рекорд $S^* := S$.

2.3. Добавить рекорд в популяцию $\Gamma := \Gamma \cup \left\{S^*\right\}$.

## 4.2 Родительские расписания

Построение множества родительских расписаний осуществляется следующим образом. Просматривается вся популяция $\Gamma$ в порядке неубывания длин расписаний, каждое рассматриваемое расписание с некоторой вероятностью добавляется в множество родительских $\Gamma'$. Если просмотрена вся популяция и не набрано достаточное количество расписаний(один из параметров алгоритма) добираем лучших из недобавленных.

Далее, для скрещивания, пара родительских расписаний будет выбираться случайным образом из множества $\Gamma'$ родительских расписаний.

Похожая стратегия выбора родителей была позаимствованна из гетенического алгоритма, описанном в [5], который хорошо себя зарекомендовал на задачах с большой размерностью.

$ParentsSelection(\Gamma, N) \to \Gamma'$

1. Положить $\Gamma' := \varnothing$.
2. Упорядочить $\Gamma$ по неубыванию длин расписаний.
3. Просматривая $\Gamma$ от начала до конца, с некоторой вероятностью добавлять просматриваемое расписание в $\Gamma'$.
4. Если $\left|\Gamma'\right| < N$, то добавить в $\Gamma'$ лучшие недобавленные расписания из $\Gamma$.

## 4.3 Процедура скрещивания

Процедару скрещивания является ключевой процедурой при разработке генетического алгоритма, данная процедура дает возможность получать новые расписания из уже суще-

ствующих, использую сильные и слабые их стороны. В описываемом алгоритме будет использоваться процедура скрещивания, основанная на выделении множества работ, выполнение которых одновременно приводит к маленькому остатку неиспользованных ресурсов.

Дочернему расписанию будем стараться передавать тот участок хромосомы родительского расписания, который отвечает наименьшему остатку ресурсов.

В каждый момент времени, в течение которого выполняется проект, считается остаток ресурсов и, если он меньше заранее заданного допустимого остатка, то будем запоминать набор работ, выполняющихся в текущий момент времени.

Опишем процедуру выбора “плотных”(в смысле неиспользованных ресурсов) участков хромосомы. Под хромосомой понимается список работ, по которому построено расписание. Обозначим через $R$ допустимый остаток неиспользованных ресурсов, если остаток неиспользовананных ресурсов меньше $R$, то считаем набор работ, выполняющихся в этот момент времени, плотным.

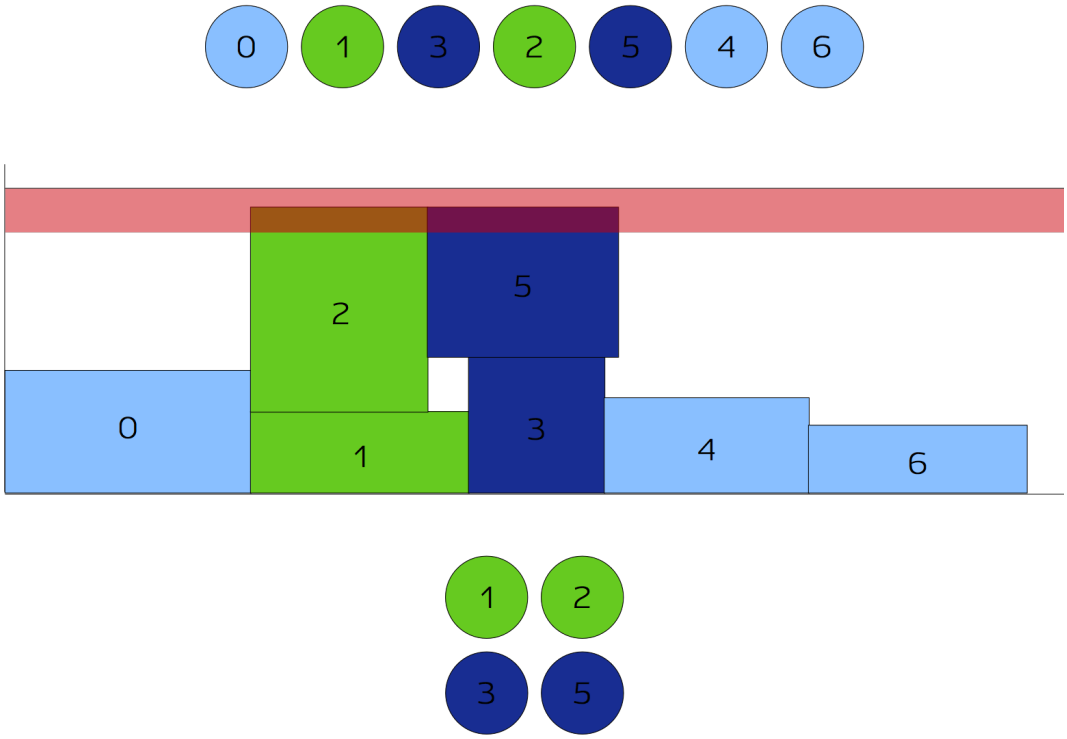


Рис. 4.1: Иллюстрация принципа работы процедуры *DenseJobs*

$DenseJobs(S, R) \to \Theta$

1. Положить $t := 0$, $\Theta = \varnothing$

2. Пока $t < T(S)$ делать следующее:

2.1. Посчитать остаток ресурсов $r$ в момент времени $t$.

2.2. Если $r > R$, то положить $t := t + 1$

2.3. Иначе обновить список плотных наборов работ $\Theta = \Theta \cup A(t)$

В процессе выделения плотных наборов работ может получиться так, что некоторые наборы будут пересекаться, в этом случае будем оставлять тот из них, который дает меньший остаток неиспользованных ресурсов.

Скрещивать две хромосомы будем следующим образом. Рассмотрим первые наборы работ в каждом расписании, тот набор, которому, соответствует меньший остаток ресурсов выберем в качестве ведущего, добавим работы от начала списка(соответствующего расписанию ведущего блока) до последней работы ведущего блока включительно в новый (дочерний) список. Затем рассматриваем первые блоки расписаний, в которых не содержатся работы уже добавленные в дочерний список. Сравнивая два новых блока, выбираем из них лучший и добавляем работы из списка работ, соответствующего выбранному блоку(кроме, конечно, тех работ которые уже добавлены в дочерний список). Описанные действия будем повторять, пока не просмотрим все блоки, затем, если это необходимо, дополним недостающие работы в том порядке, в котором они находятся в расписании с наименьшей длительностью. В итоге, по полученному списку, дочернее расписание будем строить с помощью последовательного декодера.

На рисунке 4.2 приведен пример, где разными цветами выделены плотные наборы работ. Для простоты понимания введем следующие обозначения:

- $L_1$ и $L_2$ — верхний и нижний списки работ. $S_1$ и $S_2$ — соответствующие расписания.
- $D_{i,j}$ — $j$-й набор $i$-го расписания, $i, j \in \Big\{1, 2\Big\}$.
- $E_1 \preccurlyeq E_2$ — будет означать, что $E_1$ лучше $E_2$, сравниваются ли расписания(в смысле их длительностей) или плотные наборы работ(в смысле остатка неиспользованных ресурсов).

Пусть $S_1 \preccurlyeq S_2$, $D_{1,1} \preccurlyeq D_{2,1}$ и $D_{2,2} \preccurlyeq D_{1,2}$. Тогда оператор скрещивания выберет из блоков $D_{1,1}$ и $D_{2,1}$ блок $D_{1,1}$, т.к. он лучше, добавит в дочерний список часть работ в порядке соответствующем списку работ $L_1$: $(0, 1, 2, 3, 5, 4)$. Затем будет рассматривать блоки $D_{1,2}$ и $D_{2,1}$ и выберет лучший, т.е. $D_{2,1}$, добавит соответствующие работы в дочерний список:

$(7, 9, 8, 6, 11, 10, 12, 13)$. Теперь все блоки просмотрены, но не все работы добавлены, поэтому дочерний список нужно дополнить недостающими работами в порядке $L_1$, в силу того, что расписание $S_1$ лучше: $(14, 15)$.

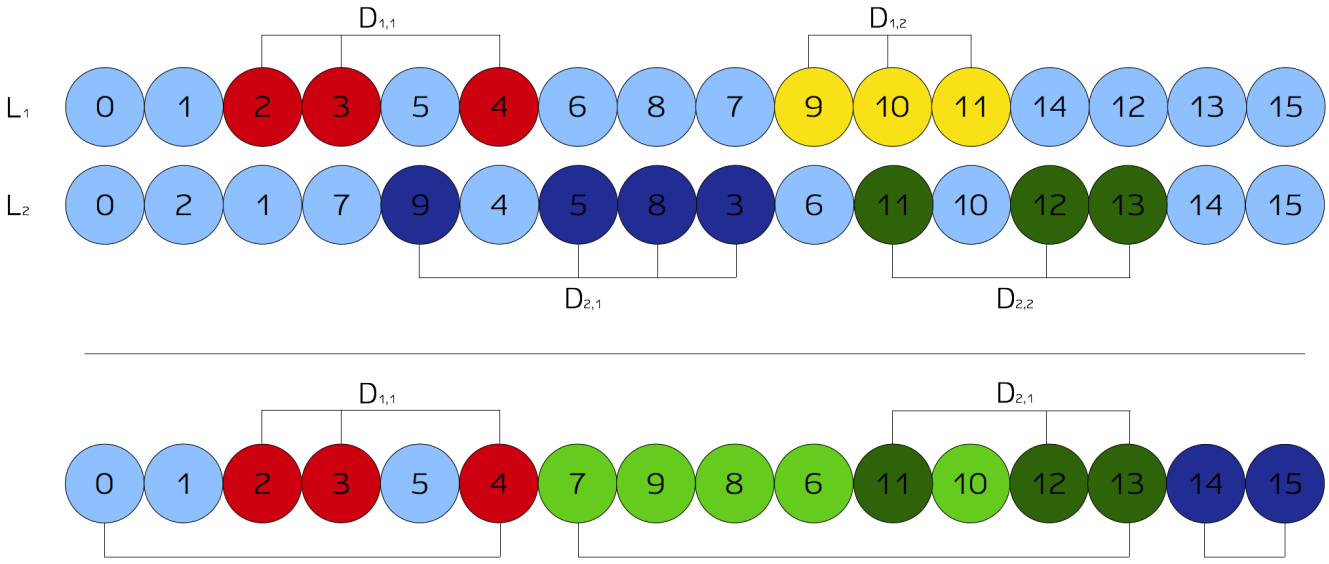


Рис. 4.2: Пример работы процедуры скрещивания

Формально процедура скрещивания может быть записана следующим образом.

$Crossing(S_1, S_2) \to S$

1. Положить список работ $L := \varnothing$.
2. Найти плотные участки для расписаний $\Theta_1 := DenseJobs(S_1)$, $\Theta_2 := DenseJobs(S_2)$.
3. Пока $\Theta_1 \neq \varnothing$ & $\Theta_2 \neq \varnothing$, выполнять:
   - 3.1. Если $\Theta_1(0) \preccurlyeq \Theta_2(0)$, то проделать следующее:
     - 3.1.1. Найти в списке $L_1$ позицию $j$ последней работы блока $\Theta_1(0)$.
     - 3.1.2. Добавить в $L$ все работы, начиная с $L_1(0)$ и заканчивая $L_1(j)$, пропуская уже добавленные работы.
     - 3.1.3. Удалить из $\Theta_1$ и $\Theta_2$ блоки содержащие работы, уже добавленные в дочерний список.
   - 3.2. Иначе применить шаги 3.1.1 – 3.1.3 к $\Theta_2(0)$.
4. Если $\Theta_1 = \varnothing$ & $\Theta_2 \neq \varnothing$, то применить шаги 3.1.1 – 3.1.3 к $\Theta_2(0)$.

5. Если $\Theta_1 \neq \varnothing$ & $\Theta_2 = \varnothing$, то применить шаги 3.1.1 – 3.1.3 к $\Theta_1(0)$.

6. Если $\Theta_1 = \varnothing$ & $\Theta_2 = \varnothing$ & (не все работы присутствуют в $L$), то добавить недостающие работы в порядке в котором они находятся в расписании лучшем из $S_1$ и $S_2$.

7. Построить расписание $S := SerialDecoder(L)$.

## 4.4 Процедура мутирования

Оператор мутации состоит в "перемешивании" работ в списке соответствующем расписанию, и построении расписания по полученному списку работ. Перемешивание будет осуществляться в два этапа. На первом этапе будем выбирать пару случайных работы в списке и менять их местами, если это не будет нарушать условия предшествования. На втором этапе будем переставлять одну случайную работу на другое место, без нарушения условий предшествования. Описанная процедура будет повторяться некоторое число раз.

Формально оператор мутации может быть записан следующим образом.

$Mutation(S, N) \to S'$

1. Выбрать случайное число $N' \leqslant N$

2. $N'$ раз проделать следующее:

   2.1. Выбрать две случайные работы, которые можно поменять местами.

   2.2. Поменять их местами.

3. Выбрать случайное число $N'' \leqslant N$

4. $N''$ раз проделать следующее:

   4.1. Выбрать случайную работу и позицию на которую ее можно переставить.

   4.2. Переставить выбранную работу на новую позицию.

## 4.5 Создание следующего поколения

Отбор в следующее поколение происходит следующим образом. Из множества дочерних расписаний выбирается определенное количество лучших расписаний и добавляется в

следующее поколение. Если расписаний меньше, чем должно быть в популяции, то будут добавляться лучшие расписания из предыдущего поколения.

## 4.6 Формальное описание алгоритма

В качестве критерия остановки было выбрано количество порожденных расписаний. Общая схема алгоритма выглядит следующим образом.

1. Создать начальную популяцию $\Gamma$ и запомнить рекорд $S^*$.
2. Положить множество дочерних расписаний $\Gamma'' := \varnothing$.
3. Пока не выполнен критерий остановки:
   - 3.1. Построить множество родительских расписаний $\Gamma' \subseteq \Gamma$.
   - 3.2. Пока не будет сгенерировано достаточное количество дочерних расписаний, делать следующее:
     - 3.2.1. Выбрать два родительских расписания $S_1$ и $S_2 \in \Gamma'$.
     - 3.2.2. Скрестить $S_1$ и $S_2$: $S' := Crossing(S_1, S_2)$.
     - 3.2.3. Применить к $S'$ последовательно операторы мутации и локального улучшения Пинг-Понг: $S' := Motation(S')$, $S' := PingPong(S')$
     - 3.2.4. Если $S'$ до этого было рекордным и после испортилось, то возвращаем $S'$ прежнее значение.
     - 3.2.5. Если $T(S') < T(S^*)$, то сменить рекорд $S^* := S'$.
     - 3.2.6. Обновить множество дочерних расписаний $\Gamma'' = \Gamma'' \cup \left\{S'\right\}$.
   - 3.3. Создать новую популяцию — следующее поколение.

# Глава 5

## Генетический алгоритм с использованием композитных расписаний

В данной главе будут описан генетический алгоритм, за основу которого взяты идеи из описанных выше алгоритмов. Будет представлен еще один оператор скрещивания, разработанный на основе выделения плотных наборов работ и построении расписаний из окрестностей $N_A(S)$ и $N_T(S)$.

Процедуры создания начальной популяции, выбора родительских расписаний, мутирования и создания следующего поколения аналогичны процедурам описанным в главе 4. В предложенном алгоритме будут использоваться две процедуры скрещивания, одна из них описана, так же, в главе 4.

## 5.1 Процедура скрещивания №2

Опишем еще одну процедуру скрещивания. Выделив в обоих расписаниях плотные наборы работ, выберем лучший из них, в смысле остатка неиспользованных ресурсов. Затем, отыскав эти работы во втором списке, найдем для каждой из них исходящую сеть (или входящую, в зависимости от того, какое расписание мы хотим построить), и, выделим сегмент в списке работ между самой левой и самой правой работами блока и исходящими (входящими) сетями.

На рисунке 5.1 приведен пример, где разными цветами выделены блоки работ. Для простоты понимания введем следующие обозначения:

- $L_1$ и $L_2$ — верхний и нижний списки рабор. $S_1$ и $S_2$ — соответствующие расписания.
- $D$ — самый плотный набор работ.

Оператор скрещивания выберет из всех плотных наборов работ, набор $D$ расписания $S_1$, т.к. он дает наименьший остаток неиспользованных ресорсов. Затем найдет работы этого набора в списке $L_2$, для каждой из них построит входящую (исходящую) сеть и выделит сегмент работ между самой левой и самой правой работами.

Дочернее расписание будет строиться по полученному сегментированному списку работ, аналогично построению расписания из окрестности $N_A(S)(N_T(S))$. Такое расписание будем называть композитным.

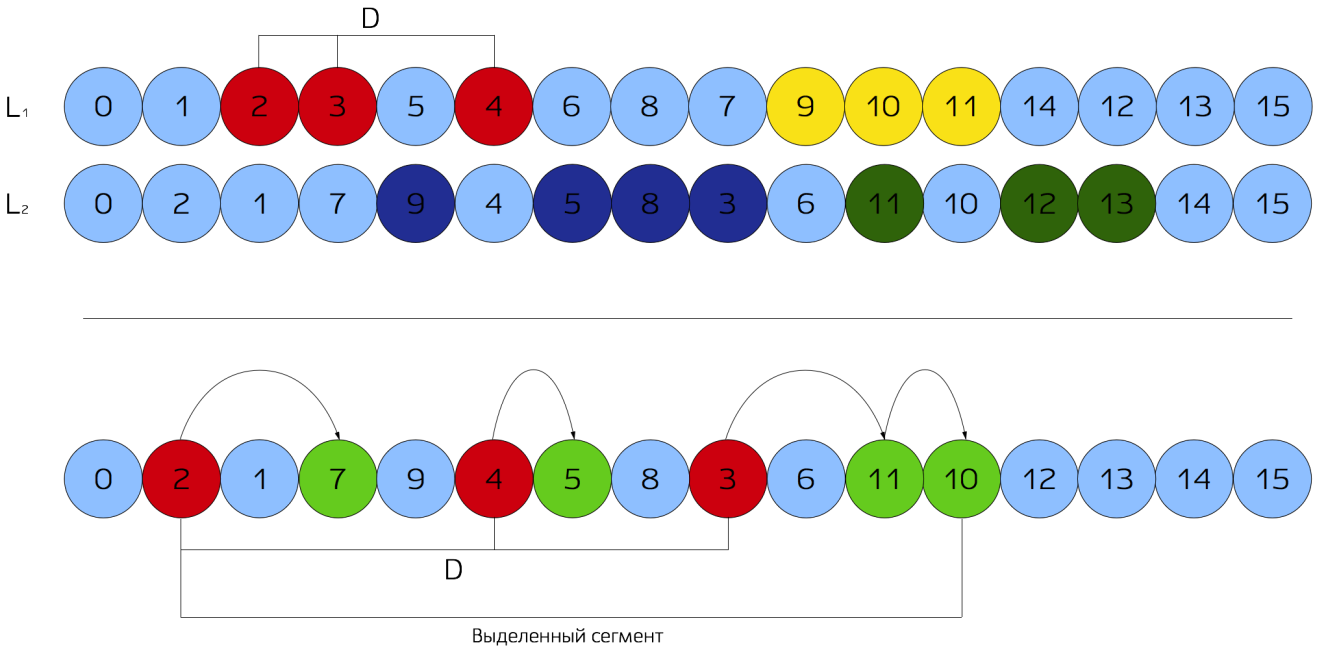


Рис. 5.1: Пример работы процедуры скрещивания

Формально процедура скрещивания может быть записана следующим образом.

$Crossing2(S_1, S_2) \to S$

1. Найти плотные участки для расписаний $\Theta_1 := DenseJobs(S_1)$, $\Theta_2 := DenseJobs(S_2)$.
2. Найти лучший участок $D := \min\limits_{D_i \in \Theta_1 \cup \Theta_2} D_i$ (пусть $D \in \Theta_a$ & $D \notin \Theta_b$, $a, b \in \left\{1, 2\right\}$, $a \neq b$).
3. Положить сегмент $\alpha := D$.
4. Для каждой работы $j \in D$ сделать:
   4.1. Добавить в $\alpha$ работы сети, исходящей из j.
5. Найти $j_1$ и $j_2$ — позиции в списке $L_b$ самой левой и самой правой работ из $\alpha$.

6. Добавить в $\alpha$ все недобавленные работы между $j_1$ и $j_2$ списка $L_b$.

7. Построить расписание $S$:

   7.1. Работы до $\alpha$ назначить на расписание как в $S_2$.

   7.2. Работы из $\alpha$ назначить на расписание параллельным декодером.

   7.3. Работы после $\alpha$ назначить на расписание последовательным декодером.

## 5.2 Формальное описание алгоритма

Опишем ключевые моменты предлагаемого алгоритма. Одна из особенностей — применение двух процедур скрещивания, выбор конкретной процедуры будет осуществляться случайно, с вероятностью $\frac{1}{2}$. Еще одной из отличительных особенностей является возможность "разбавить" популяцию — если рекорд не был обновлен за несколько (параметр алгоритма) последних шагов, то будем выбирать несколько расписаний из популяции и заменять их на новые, которые будем получать с помощью процедуры, использующейся при создании начальной популяции, затем выберем еще несколько расписаний и попытаемся их улучшить с помощью процедуры $TabuSearch$ Ю.А.Кочетова и А.А.Столяра.

В качестве критерия остановки, как и в первом генетическом алгоритме, было выбрано количество порожденных расписаний. Общая схема алгоритма выглядит следующим образом.

1. Создать начальную популяцию $\Gamma$ и запомнить рекорд $S^*$.

2. Положить множество дочерних расписаний $\Gamma'' := \varnothing$.

3. Пока не выполнен критерий остановки:

   3.1. Построить множество родительских расписаний $\Gamma' \subseteq \Gamma$.

   3.2. Пока не будет сгенерировано достаточное количество дочерних расписаний, делать следующее:

      3.2.1. Выбрать два родительских расписания $S_1$ и $S_2 \in \Gamma'$.

      3.2.2. С вероятностью $\frac{1}{2}$ скрестить $S_1$ и $S_2$ с помощью $Crossing$ или $Crossing2$: $S' \in \Big\{Crossing(S_1, S_2), Crossing2(S_1, S_2)\Big\}$.

3.2.3. Применить к $S'$ последовательно операторы мутации и локального улучшения Пинг-Понг: $S' := Motation(S')$, $S' := PingPong(S')$

3.2.4. Если $S'$ до этого было рекордным и после испортилось, то возвращаем $S'$ прежнее значение.

3.2.5. Если $T(S') < T(S^*)$, то сменить рекорд $S^* := S'$.

3.2.6. Обновить множество дочерних расписаний $\Gamma'' = \Gamma'' \cup \left\{S'\right\}$.

3.3. Создать новую популяцию — следующее поколение.

3.4. Если $S^*$ давно не обновлялось проделать следующее:

3.4.1. Выбрать некоторые расписания и заменить их на новые.

3.4.2. Выбрать некоторые расписания и применить к ним $TabuSearch$.

# Глава 6

## Численные эксперименты

Все численные эксперименты проводились на примерах из библиотеки PSPLib [8]. В этой библиотеке содержится много примеров разной размерности: по 480 примеров для размерности 30, 60, 90 и 600 примеров для 120, число типов ресурсов равно 4. Примеры разбиты на классы. В каждом классе имеется 10 примеров, порожденных с помощью датчика псевдослучайных чисел при фиксированных значениях трех параметров [3]:

- $NC \in \left\{1.5; 1.8; 2.1\right\}$ — среднее число непосредственных предшественников каждой работы.
- $RF \in \left\{1; 2; 3; 4\right\}$ — число типов ресурсов, необходимых для выполнения каждой работы.
- $RS \in \left\{0.2; 0.5; 0.7; 1.0\right\}$ — объем выделяемых ресурсов в каждый момент времени; значение RS = 0.2 соответствует примерам с минимальным объемом выделяемых ресурсов, достаточным для разрешимости задачи; значение RS = 1 соответствует случаю неограниченных ресурсов.

Известно, что значения параметров RF = 4, RS = 0.2 соответствуют достаточно трудным классам [3]. Их индентификаторы:

- Для $n = 30$ это: j3013, j3029, j3045
- Для $n = 60$ это: j6013, j6029, j6045
- Для $n = 120$ это: X16, X36, X56

Программа была написанна на языке программирования `C++`. Тестирование проводилось на вычислительной машине с частотой процессора 2.3 Ghz.

Таблица 6.1: Обозначения

| | |
|---|---|
| $N$ | Размер популяции |
| $N_p$ | Количество родительских расписаний |
| $N_{ch}$ | Количество дочерних расписаний |
| $N^*_{children}$ | Количество дочерних расписаний в следующем поколении |
| $I$ | Начальное решение |
| $P_{KP}$ | Вероятность для задачи о рюкзаке |
| $P_p$ | Вероятность выбора родителя |
| $R$ | Остаток неиспользованных ресурсов для выделения плотного набора работ |
| $M$ | Мутация |
| $L$ | Локальное улучшение |

Таблица 6.2: Параметры

| Алгоритм | GA | GA-Composite |
|---|---|---|
| $N_p$ | 20 | 20 |
| $N_{ch}$ | 40 | 40 |
| $N^*_{ch}$ | 10 | 10 |
| $I$ | $PingPong(10)$ | $PingPong(10)$ |
| $P_{KP}$ | 0.5 | 0.5 |
| $R$ | 0.9 | 0.9 |
| $M$ | $Mutation(10) \rightarrow PingPong(10)$ | $Mutation(10) \rightarrow PingPong(10)$ |
| $L$ | – | $TabuSearch(0)$ |

Параметры настраивались экспериментальным путем. Для алгоритма *LocalSearch* Ю.А. Кочетова и А.А. Столяра, большая часть параметров была взята из статьи [3].

В таблице 6.3 преведены средние относительные погрешности, результаты сравнивались с решениями приведенными в библиотеке PSPLib [8].

Результаты сравнения алгоритмов, предложенных в данной статье, с другими известными алгоритмами [5] приведены в таблице 6.4. В таблице указана средняя относительная погрешность по отношению к нижней оценке, полученной с помощью метода критического

Таблица 6.3: Результаты : размерность 120, $N = 50000$

| Класс | GA | GA-Composite |
|---|---|---|
| 1 – 60 | 2.02 | 1.91 |
| 16 | 5.74 | 5.28 |
| 36 | 5.56 | 5.58 |
| 56 | 5.40 | 5.25 |

Таблица 6.4: Сравнение с другими алгоритмами : размерность 120

| Алгоритм | Авторы | Относительная погрешность, % | | |
|---|---|---|---|---|
| | | $N = 1000$ | $N = 5000$ | $N = 50000$ |
| GANS | Proon, Jin | 33.45 | 31.51 | 30.45 |
| DBGA | Debels, Vanhouchke | 33.55 | 32.18 | 30.69 |
| GA | Debels, Vanhouchke | 34.19 | 32.34 | 30.82 |
| GA - Hibrid, FBI | Valls et al. | 34.07 | 32.54 | 31.24 |
| Scatter search - FBI | Debels et al. | 35.22 | 33.1 | 31.57 |
| GA - FBI | Valls et al. | 35.39 | 33.24 | 31.58 |
| GA, TS - Path re-linking | Кочетов, Столяр | 34.74 | 33.36 | 32.06 |
| GA - Composite | Эта работа | 36.22 | 34.43 | 32.31 |
| GA | Эта работа | 36.32 | 34.76 | 32.45 |
| GA - Self-adapting | Hartmann | 37.19 | 35.39 | 33.21 |
| GA - Activity list | Hartmann | 39.37 | 36.74 | 34.04 |
| Sampling - LFT, FBI | Tomas, Lova | 36.49 | 35.81 | 35.01 |
| TS - Activity list | Nonobe, Ibaraki | 40.86 | 37.88 | 35.85 |
| GA - Late join | Ceolho, Tavares | 39.97 | 38.41 | 36.44 |
| SA - Activity list | Bouleimen, Lecocq | 42.81 | 37.68 | – |
| GA - Priority rule | Hartmann | 39.93 | 38.49 | 36.51 |
| Sampling - LFT | Kolisch | 39.6 | 38.75 | 37.74 |
| Sampling - WCS | Kolisch | 39.65 | 38.77 | – |
| Sampling - Adaptive | Kolisch, Drexl | 41.37 | 40.45 | – |
| GA - Problem space | Leon, Ramamoorthy | 42.91 | 40.69 | – |
| Sampling - Random | Kolisch | 44.46 | 43.05 | 41.44 |

пути.

# Заключение

Для задачи календарного планирования с ограниченными ресурсами разработан генетический алгоритм, в своей основе опирающийся на выделение плотных(в смысле остатка неиспользованных ресурсов) наборов работ, и построение композитных расписаний. Использование идей построения композитных расписаний позволило улучшить генетический алгоритм. Численные эксперименты показали, что предложенный алгоритм дает неплохие результаты по сравнению с лучшими на сегодняшний день алгоритмами.

# Литература


1. *Э.Х. Гимади, В.В. Залюбовский, С.В. Севастьянов* Полиномиальная разрешимость задач календарного планирования со складируемыми ресурсами и директивными сроками: Дискретный анализ и исследование операций, 2000. Сер. 2. Т. 7. №1. С. 9–34.

2. *J. Blażewicz, J.K. Lenstra, A.H.G. Rinnoy Kan* Scheduling subject to resource constraints: Classification and complexity: Discrete Applied Math, 1983. V. 5. N 1. P. 11–24.

3. *Ю.А. Кочетов, А.А. Столяр* Использование чередующихся окрестностей для приближенного решения задачи календарного планирования с ограниченными ресурсами: Дискретный анализ и исследование операций, 2003. Сер. 2. Т. 10. С. 29–55.

4. *R. Kolisch, S. Hartmann* Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: an update: European Journal of Operational Research, 2006. V. 174. P. 23–37.

5. *S. Proon, M. Jin* A Genetic Algorithm with Neighborhood Search for the Resource-Constrained Project Scheduling Problem: Naval Research Logistics, 2011. V. 58. N. 73–82.

6. *R. Kolich* Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation: European Journal of Operational Research, 1996. V. 90. N 2. P. 320—333.

7. *R. Kolisch, S. Hartmann* Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In J. Weglarz, editor, Project scheduling: Recent models, algorithms and applications. Kluwer Academic Publishers, 1999. P. 147–178.

8. *PSPLib* http://129.187.106.231/psplib/