

Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего профессионального образования

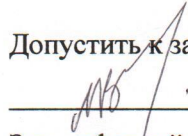
**«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»**

**Институт математики и компьютерных наук
Кафедра алгебры и дискретной математики**

Задача построения графика горячей прокатки, сбалансированного по видам продукции.

Исследование графика

Допустить к защите:


Зав. кафедрой

д. ф.-м. н.,

профессор,

Волков М. В.

27/05/16

Квалификационная работа на соискание
степени магистра наук по направлению

«Фундаментальная информатика и
информационные технологии»

студентки группы МгФиит-2

 Леоновой С.И.

Научный руководитель:

д. ф.-м. н.,

профессор,

 Баранский В. А.

Екатеринбург

2016

Оглавление

Введение	4
1 Формализация основных понятий процесса построения графика прокатки	7
1.1 Характеристики партий	7
1.2 Условия предшествования партий в графике прокатки	7
1.3 Дополнительные обозначения	9
2 Задача построения графика горячей прокатки, сбалансированного по видам продукции	11
2.1 Практическая задача построения графика прокатки. Критерий сбалансированности	11
2.2 Математическая модель	11
2.3 Метод решения	12
3 Исследование структуры графа предшествования партий	13
3.1 Критерий неразделимости	13
3.2 Выделение блоков в графе G_w	16
3.3 Простые цепи в графе G_w	19
3.4 Максимальные простые цепи в графе G	23
4 Программная реализация	27
4.1 Алгоритм построения блоков	27
4.1.1 Построение графа G	27
4.1.2 Поиск блоков в графе G	28
4.2 Результаты расчетов	29
4.2.1 Построение графа G	30
4.2.2 Построение блоков графа G	30
Заключение	32
Литература	33
Приложение	34

Введение

В данной работе исследуется задача оперативного планирования (графикования) работы непрерывных полосовых станов горячей прокатки стали (стан г/п). Непрерывный полосовой стан горячей прокатки служит для получения горячекатанных полос различных ширин и толщин. В качестве исходного сырья или, как говорят, подката используются слябы — стальные заготовки в форме параллелепипедов, имеющих толщину, ширину и длину. Обычно ширина сляба заметно больше его толщины. Стан преобразует сляб в полосу, обрабатывая металл давлением в клетях. Клеть — это основной рабочий механизм стана, состоящий из рабочих валков, которые непосредственно воздействуют на заготовку (давлением), опорных валков, которые располагаются над и под рабочими валками и служат для уменьшения прогиба рабочих валков, то есть увеличения жесткости валковой системы. Также в состав клетки входят другие конструкции и механизмы, обеспечивающие работу клетки. Непрерывный стан имеет несколько клеток, и полоса в стане может находиться во всех или в нескольких клетях. Стан горячей прокатки обычно состоит из черновой группы клеток, которая формирует так называемый раскат, то есть раскатанную заготовку для последующей обработки в чистовой группе клеток. Функция чистовой группы состоит в формировании окончательной толщины полосы. После прокатки полоса может сматываться в рулон. Подрезка кромки горячекатанных полос для получения ровного края производится обычно на отдельных агрегатах. Ширина прокатанной полосы близка к ширине сляба, из которого она изготовлена. Толщина полосы обычно на порядок или два меньше толщины сляба. Длина полосы может исчисляться в сотнях метров, длина сляба от нескольких метров до десяти-двенадцати метров.

Можно выделить два рабочих цикла непрерывного стана горячей прокатки. Первый или большой цикл заключен между двумя последовательными заменами опорных валков стана. Замена делается обычно один раз в несколько дней. Второй или малый цикл заключен между двумя последовательными заменами рабочих валков стана. Такая операция при полноценной загрузке стана производится с промежутком в несколько часов. Оперативное планирование работы стана г/п имеет приблизительно суточный горизонт и связано с формированием (и корректировкой, по необходимости) последовательности прокатываемых партий, собранных в несколько последовательных графиков прокатки, каждый из которых представляет собой последовательность всех планируемых к прокатке партий в рамках одного малого цикла работы стана.

График прокатки представляет собой упорядоченный набор партий. Партия состоит из набора слябов, которые прокатываются в полосы. Все слябы имеют одинаковые размеры и химический состав (с погрешностью, которой мы пренебрегаем). Для всех полос партии заданы одни и те же значения ширины и толщины, которые полосы прокатанные из этих слябов должны получить на выходе из стана. Таким образом, слябы в партии считаются идентичными, как и прокатываемые из них полосы. После того, как все слябы прокатаны в полосы, полученную совокупность также называют партией. Далее по контексту будет понятно, о каком состоянии партии идет речь.

В ряде работ (например, [5], [6]) приведены практические постановки задач совместного планирования непрерывной разливки стали и последующей горячей прокатки. Подробно обсуждаются различные варианты организации работы предприятия, когда разлитый металл полностью остывает, прежде, чем идет в нагревательные печи и на прокатку, либо остывает частично в виду наличия транспортного плеча, остывает мало и идет, как принято говорить на отечественных предприятиях, в прокатку горячим садом. Иллюстрируется эффект энергосбережения и прочее. Также в работах приводятся основные технологические ограничения, связанные с формированием графиков прокатки, которые используются в аналогичных условиях большинством предприятий. Приводятся некоторые математические модели, используемые при расчете графиков прокатки, для таких задач в иностранной литературе приняты названия "hot strip mill planning" и HRBPPs (hot rolling batch planning problems). Приводимые в работах модели представляют собой аналоги и обобщения задач коммивояжера TSP (Travelling Salesman Problem) и планирования маршрутов транспорта VRT (vehicle routing problem, [10]). Существуют модели PCTSP (Prize Collecting TSP), предложенная Балашем ([7], [8]) и PCVRP (Prize Collecting VRP, [9]), в которых для каждой пары партий прокатки задана стоимость перехода с одной на другую, то есть стоимость того, что в графике прокатки они непосредственно следуют друг за другом. Также задается награда (приз) за то, что партия входит в построенный график прокатки, и возможен штраф за невхождение партии в график прокатки. Результатом является построение графиков с максимизацией получаемых призов, за вычетом штрафов за переходы с одной партии на другую. Указанные задачи являются в общем случае труднорешаемыми.

Построение графика обусловлено несколькими критериями его оценки и набором технологических ограничений, которым он должен удовлетворять. Мы рассматриваем ситуацию, когда для каждой партии прокатки известно для какого вида продукции она изготавливается, пойдут ли полосы партии сразу на отгрузку потребителю (после порезки) или предназначены для дальнейшей переработки в других цехах предприятия (травление, холодная прокатка и прочее). При формировании каждого следующего графика прокатки известно текущее состояние складов полуфабрикатов и готовой продукции, а также состав уже сформированных планируемых до него графиков прокатки. На основании этих данных и информации о трудоемкости логистических операций, производительности последующих в технологических цепочках цехов и агрегатов, рассчитывается задание для формирования следующего графика прокатки. Это задание включает в себя ограничения на общий объем металла для прокатки, а также ограничения по отдельным видам продукции и направлениям в виде допустимых диапазонов, сколько их возможно включить в график прокатки, чтобы обеспечить выполнение заказов, отгрузку продукции непосредственно потребителю и другим цехам предприятия.

Цели настоящей работы состоят в построении математической модели для задачи формирования сбалансированного графика прокатки и ее исследования.

Приведем теперь основные определения из теории графов, которые мы будем использовать в дальнейшем ([1]).

Обыкновенным графом G называется пара множеств (V, E) , где E - произвольное подмножество из $V^{(2)}$. Элементы множеств V и E называют соответственно *вершинами* и *ребрами* графа G .

Ориентированным графом (или *орграфом*) G называется тройка (V, D, ϕ) , где ϕ - некоторое отображение множества D в множество V^2 . Элементы множеств V и D называются соответственно *вершинами* и *дугами* орграфа G .

Маршрутом (или (v_0, v_i) - *маршрутов*) в графе G называется чередующаяся по-

следовательность вершин и ребер

$$v_0, e_1, v_1, \dots, v_{t-1}, e_t, v_t,$$

в которой $e_i = v_{i-1}v_i$ ($1 \leq i \leq t$).

Цепь - это маршрут без повторяющихся ребер. Цепь называется *простой цепью*, если в ней нет повторяющихся вершин кроме, быть может, совпадающих конечных вершин. Для удобства простую цепь будем обозначать просто

$$v_0v_1v_2 \dots v_{t-1}v_t$$

.

На множестве вершин графа G определим *отношение связности* \sim , полагая

$$u \sim v \Leftrightarrow \text{существует } (u, v) \text{ - маршрут.}$$

Отношение связности является отношением эквивалентности. V_1, V_2, \dots, V_k - классы этого отношения. Графы $G_i = G(V_i)$, порожденные множествами V_i ($1 \leq i \leq k$) называются *компонентами связности* графа G .

Пусть $G = (V, E)$ - произвольный граф. Вершина v называется *точкой сочленения*, если граф $G - v$ имеет больше компонент связности, чем граф G .

Связный граф называется *неразделимым*, если он не содержит точек сочленения. *Блоком* графа G называется любой его максимальный неразделимый подграф.

Глава 1

Формализация основных понятий процесса построения графика прокатки

График прокатки представляет собой упорядоченный набор партий. Партия состоит из набора слэбов, которые прокатываются в полосы. Для всех полос партии заданы одни и те же желаемые значения ширины и толщины, которые полосы должны получить на выходе из стана.

1.1 Характеристики партий

Обозначим через P множество партий, $|P| \in \mathbb{N}$. Введем следующие обозначения.

- $w : P \rightarrow \mathbb{R}^+$ — ширина полос в партии, значение w на элементе $p \in P$ будем обозначать w_p . Множество всех ширин партий обозначим через $W = \{w_p | p \in P\}$.
- $t : P \rightarrow \mathbb{R}^+$ — толщина полос в партии, значение t на элементе $p \in P$ будем обозначать t_p .
- $l : P \rightarrow \mathbb{R}^+$ — весовая функция, характеризующая полезность партии, значение l на элементе $p \in P$ будем обозначать $l(p)$. В зависимости от конкретной задачи, в качестве полезности партии могут выступать разные характеристики. Этими характеристиками могут быть как длина (в километрах), вес (в тоннах) партии, а также более сложные величины, учитывающие спрос, время изготовления, стоимость продукции и т.д.

1.2 Условия предшествования партий в графике прокатки

Как уже было сказано в начале главы, график прокатки представляет собой упорядоченный набор партий. Необходимость упорядочивать партии обусловлена техническими особенностями прокатного стана. В процессе прокатки рабочие валки изнашиваются, поэтому требуется, чтобы ширина полос партий в графике прокатки не возрастала и не менялась слишком сильно. Нарушение этого условия может привести к дефектам на поверхности прокатанных полос. Также требуется, чтобы рабочие валки подстраивались под конкретную толщину полос партии, поэтому необходимо,

чтобы толщина полос соседних партий не сильно менялась. Это обеспечивает меньшую нагрузку на валки. На рисунке 1.1 схематически изображен график прокатки в разрезе ширины (w) и толщины (t) партий, который удовлетворяет описанным техническим особенностям.



Рис. 1.1: Характеристика графика прокатки.

Для формализации требований на порядок следования партий в графике прокатки введем следующие обозначения.

- $r : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ — функция, определяющая максимальную величину, на которую может отличаться значение толщины партии от значения толщины соседней партии в графике прокатки. Данная функция должна быть монотонно неубывающей, т.е. $\forall t_1, t_2 \in \mathbb{R}^+ t_1 \leq t_2 \Rightarrow r(t_1) \leq r(t_2)$.
- $\delta \in \mathbb{R}^+$ — величина максимальной допустимой разности ширин полос партии между любыми двумя соседними партиями в графике прокатки.

Ограничение на предшествование партий определим следующим образом: партия q может непосредственно следовать за партией p в графике прокатки в том и только в том случае, когда выполнено

1. ограничение «для перехода по толщине» : $|t_p - t_q| \leq \min\{r(t_p), r(t_q)\}$;
2. ограничение «для перехода по ширине» : $0 \leq w_p - w_q \leq \delta$.

Рассмотрим следующий пример.

Пример. Пусть $\delta = 250$, а $r(t) = \begin{cases} 0.8, & \text{если } t \in [1.29, 2] \\ 1.5, & \text{если } t \in (2, 3] \\ 2, & \text{если } t \in (3, 16] \\ 4, & \text{если } t \in (16, 20] \end{cases}$

Также пусть имеются 4 партии p_1, p_2, p_3, p_4 со следующими характеристиками:

- $w_{p_1} = 350, t_{p_1} = 2$
- $w_{p_2} = 600, t_{p_2} = 3.1$
- $w_{p_3} = 600, t_{p_3} = 2.8$
- $w_{p_4} = 900, t_{p_4} = 2$

Для начала применим к партиям ограничение «для перехода по ширине». У партии p_4 значение ширины равно 900, и оно отличается от остальных более чем на δ , равное 250, откуда следует, что партия p_4 не может быть предшественником любой из оставшихся партий. Также заметим, что партия p_4 не может следовать ни за одной из партий p_1, p_2, p_3 в графике прокатки, поскольку значение w_{p_4} является наибольшим. Партии p_3 и p_2 могут быть предшественниками партии p_1 , а так же предшественниками друг друга.

Теперь применим к партиям ограничение «для перехода по толщине». Для начала необходимо отметить, что для партии p_1 значение функции $r(t)$ равно 0.8, а для партий p_2 и p_4 значение функции $r(t)$ равно 1.5. Поскольку в ограничении «для перехода по толщине» берется минимум от значений функции r на толщинах соседних партий, то можно сразу сказать, что партия p_1 может быть соседней в графике прокатки только с партией p_3 , так как значение толщины партии p_2 отличается от значения толщины партии p_1 более, чем на 0.8. Партии p_2 и p_3 могут быть предшественниками друг друга, поскольку их значения толщины отличаются друг от друга не более, чем на 1.5.

Исходя из описанных заключений, для данного множества партий существует два допустимых графика прокатки: $p_2p_3p_1$ и p_4 .

Из приведенного примера видно, что для произвольного множества партий графиков прокатки, удовлетворяющих всем ограничениям, может быть несколько, или ни одного. Выбирая различные критерии оценки графиков прокатки, а также используя характеристику полезности партии $l(p)$, можно среди всех допустимых графиков прокатки выбрать наиболее подходящий для конкретной задачи.

Множество всех допустимых графиков прокатки удобнее всего представить в виде ориентированного графа.

Пусть G ориентированный граф на множестве узлов P с множеством дуг $E \subseteq P \times P$ таким, что $pq \in E$ в том и только в том случае, когда партии p и q различны, и q может непосредственно следовать за p в графике прокатки в соответствии с ограничениями 1 и 2. Такой граф G назовем *графом предшествования партий*.

Любой дуге в графе G соответствует упорядоченная пара партий, которая является технологически пригодной для прокатки в таком порядке. Поэтому множеству всех простых цепей в G соответствует множество всех допустимых графиков прокатки для заданного набора партий.

1.3 Дополнительные обозначения

Пусть p — произвольный узел в G . Обозначим через:

- $P_w = \{p | w_p = w\}$, где $w \in W$ - множество партий ширины w ,
- G_w — граф, индуцированный множеством P_w .

Пусть $w \in W$. Рассмотрим граф G_w . В данном графе ограничение «для перехода по ширине» всегда разрешено, поскольку все партии имеют одинаковую ширину, а

следовательно, дуга pq существует в том и только в том случае, когда существует дуга qp . Поэтому граф G_w можно рассматривать как обыкновенный граф.

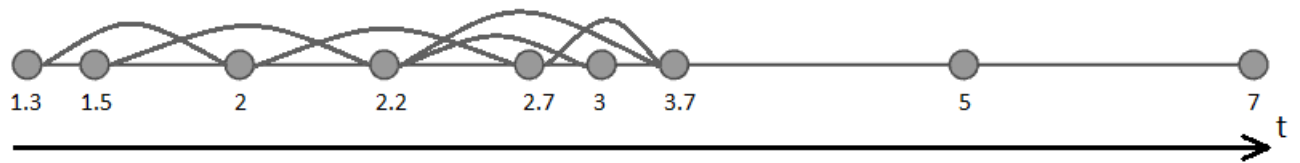
Пусть

- G_w^o — обыкновенный граф на множестве узлов P_w .

В графе G_w^o вершины p и q связаны ребром, если узлы p и q связаны дугами pq и qp в графе G_w . Аналогично обозначим:

- $P(N)$ — множество узлов подграфа N графа G_w^o .

На рисунке 1.2 изображен пример графа G_w^o с отмеченными значениями толщины партий при заданной функции $r(t)$.



t	[1.29, 2]	(2, 3]	(3, 16]	(16, 20]
r(t)	0.8	1.5	2	4

Рис. 1.2: Граф в одной ширине.

Глава 2

Задача построения графика горячей прокатки, сбалансированного по видам продукции

2.1 Практическая задача построения графика прокатки. Критерий сбалансированности

Для определения практической постановки задачи построения графика прокатки вернемся к характеристикам партий. Функции w и t , обозначающие значения ширины и толщины партии, необходимы для определения ограничений, регламентирующих порядок партий в графике прокатки. Таким образом, данные функции позволяют определить набор всех допустимых графиков прокатки и сформировать граф предшествования партий G . Практическая задача построения графика прокатки состоит в том, чтобы из всех допустимых графиков прокатки выбрать наиболее подходящий в соответствии определенным критерием. Этот критерий характеризует функция полезности партии l . В предыдущих работах ([4], [3], [2]) в качестве полезности партии рассматривалась ее длина. В результате был предложен кубический алгоритм, возвращающий график прокатки наибольшей длины.

В данной работе был выбран более сложный критерий оценки графика прокатки - сбалансированность видов готовой продукции для партий в графике. Виды готовой продукции и их объемы меняются в зависимости от спроса. "Сбалансированность" определяется полезностью партии, то есть весовой функцией l . Данный критерий оценки графика прокатки задает желаемое распределение (в виде пропорций) видов готовой продукции для партий.

Пример. Пусть партии после прокатки представляют собой горячекатаные рулоны следующих видов готовой продукции: A , B и C . Предположим, что критерий сбалансированности звучит так: в графике прокатки должно быть 50% рулонов вида A , 30% рулонов вида B и 20% рулонов вида C . Тогда задача построения графика прокатки звучит так: из всего множества партий необходимо выбрать такой набор партий, который будет удовлетворять ограничению «для перехода по ширине», ограничению «для перехода по толщине», критерию сбалансированности.

2.2 Математическая модель

Пусть $F = (f_1, f_2, \dots, f_s)$ - виды готовой продукции для партий. Введем следующие обозначения:

- $f : P \rightarrow F$ - функция, обозначающая вид готовой продукции для каждой партии.
- $\mu : F \rightarrow \mathbb{R}^+$ - желаемое распределение различных видов продукции в графике прокатки.

Оптимизационная задача : найти в \mathbb{B} простую цепь z , где

$$\sum_{i=1}^s \left| \mu(f_i) - \frac{\sum_{f(p_j)=f_i, p_j \in z} l(p_j)}{\sum_{p_j \in z} l(p_j)} \right| \rightarrow \min, p_j \in z$$

2.3 Метод решения

Для начала обратим внимание на иерархическую структуру графа предшествования партий (рисунок 2.1). В силу ограничения «для перехода по толщине» графы вида G_w имеют много неориентированных ребер. А в силу ограничения «для перехода по ширине» ориентированными ребрами связаны только те вершины, которые находятся в различных подграфах в одной ширине G_w . Поэтому граф G можно представлять как ориентированный граф, узлами которого являются графы G_w .

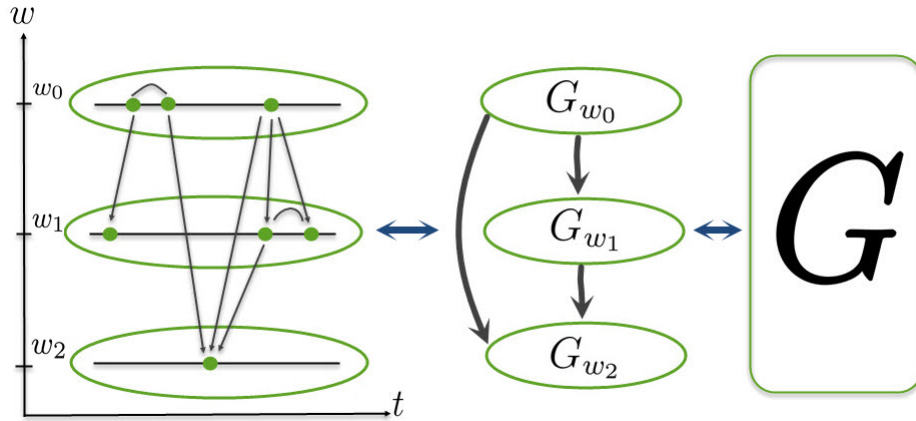


Рис. 2.1: Иерархическая структура графа предшествования партий.

Теперь вернемся к задаче построения графика прокатки, сбалансированного по видам готовой продукции.

В силу того, что критерий сбалансированности задан в виде пропорций, добавление одной партии в график прокатки может значительно повлиять на текущие значения пропорций во всем графике. Поэтому, наиболее понятный способ построения графика прокатки состоит в следующем: параллельно из каждого подграфа G_w выбираем некоторое количество партий и собираем их в один сбалансированный график прокатки.

Чтобы понять, каким образом выбирается "некоторое количество" партий, необходимо изучить структуру графов в одной ширине.

Глава 3

Исследование структуры графа предшествования партий

3.1 Критерий неразделимости

Пусть далее $w \in W$. Рассмотрим граф в одной ширине G_w^o , при этом вершины графа будем рассматривать в порядке увеличения значений функции t (см. рисунок 1.2). Таким образом, для любого подграфа B графа G_w^o , если $P(B) = \{p_1, p_2, \dots, p_k\}$, где $k \in \mathbb{N}$, то $\forall i \in \overline{1, k-1} \ t_{p_i} \leq t_{p_{i+1}}$.

Для простоты изложения введем следующие определения для вершин из множества $P(G)$ (см. рисунок 3.1).

- $p, q \in P_w$. Вершина p *левее* вершины q (обозначим $p < q$), если $t_p < t_q$.
- $p, q \in P_w$. Вершина p *не правее* вершины q (обозначим $p \leq q$), если $t_p \leq t_q$.
- $p, q, r \in P_w$. Вершина p *между* вершинами r и q если $r \leq p$ и $p \leq q$.
- $p, q \in P_w$. Вершина p *правее* вершины q (обозначим $p > q$), если $t_p > t_q$.
- $p, q \in P_w$. Вершина p *не левее* вершины q (обозначим $p \geq q$), если $t_p \geq t_q$.

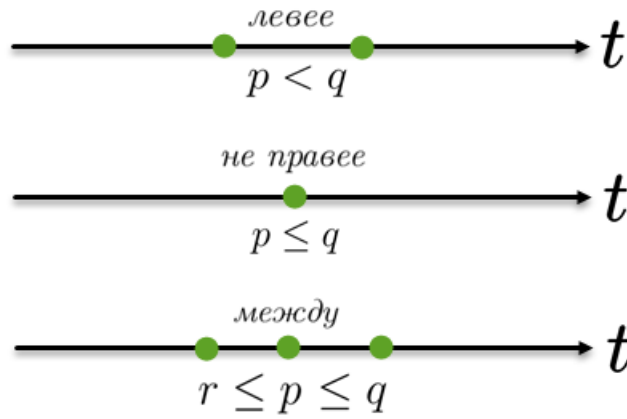


Рис. 3.1: 1) p левее q , 2) p не правее q , 3) p между r и q .

Замечание 1. В графе G_w^o наличие ребра между двумя вершинами проверяется лишь ограничением «для перехода по толщине». Ограничение «для перехода по ширине» тривиально выполняется, так как значения w всех вершин в графе G_{w_i} равны. В этом случае, когда две вершины p и q не связаны ребром и лежат между вершинами p' и q' , то p' и q' также не связаны ребром. В самом деле, не ограничивая общности можно считать, что вершина p не правее вершины q и вершина p' не правее вершины q' . Тогда $r_{t_{p'}} \leq r_{t_p} \leq r_{t_q} \leq r_{t_{q'}}$, отсюда $|t_{p'} - t_{q'}| \geq |t_p - t_q| > \min\{r(t_p), r(t_q)\} \geq \min\{r(t_{p'}), r(t_{q'})\}$. Следовательно, согласно определению p' и q' не связаны ребром (рисунок 3.2 п. б).

Аналогично можно показать, что если две вершины p и q связаны ребром, то находящиеся между ними вершины p' и q' также связаны ребром (рисунок 3.2 п. а)).

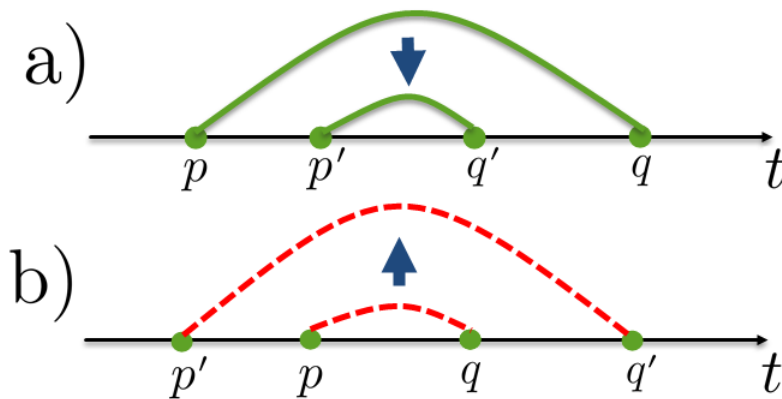


Рис. 3.2: Свойства ограничения «для перехода по толщине».

Лемма 1 (Критерий неразделимости). Пусть B — подграф графа G_w^o , $P(B) = (p_1, p_2, \dots, p_k)$, где $k > 2$. Тогда следующие условия эквивалентны:

1. B — неразделимый подграф.
2. $\forall i \in \overline{2, k-1}$ $p_{i-1}p_{i+1} \in E$, т.е. при упорядочении вершин графа по возрастанию толщины, существуют дуги через одну вершину (рисунок 3.3).

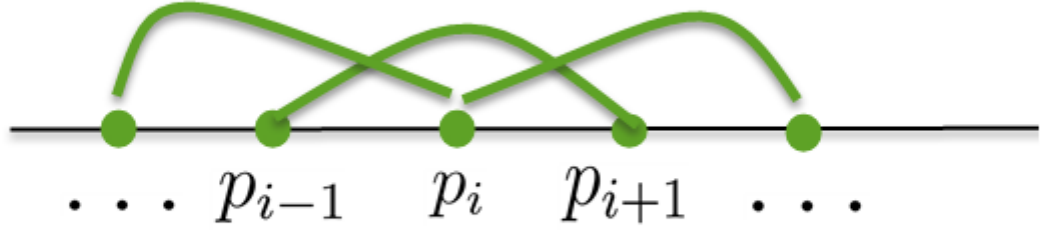


Рис. 3.3: Критерий неразделимости.

Доказательство. $(1 \Rightarrow 2)$ Рассмотрим B — неразделимый подграф графа G_w^o . Покажем, что $\forall i \in \overline{2, k-1}$ существует ребро $p_{i-1}p_{i+1}$. Предположим, что $\exists j \in \overline{2, k-1} : p_{j-1}p_{j+1} \notin E$. Тогда

$\forall l : l < j-1$ по замечанию 1 $p_l p_{j+1} \notin E$, так как $t_{p_{j+1}} - t_{p_{j-1}} \leq t_{p_{j+1}} - t_{p_l}$, и аналогично

$\forall m : m > j+1$ имеем $p_{j-1}p_m \notin E$, так как $t_{p_{j+1}} - t_{p_{j-1}} \leq t_{p_m} - t_{p_{j-1}}$.

Отсюда следует, что при удалении вершины p_j из B , вершины левее p_j не будут связаны ребрами с вершинами правее p_j , а значит, количество компонент связности увеличится. Следовательно, p_j — точка сочленения подграфа B , что противоречит определению неразделимого графа. Таким образом, $\forall i \in \overline{2, k-1}$ $p_{i-1}p_{i+1} \in E$.

$(1 \Leftarrow 2)$ Рассмотрим подграф B графа G_w^o . По условию 2 имеем $\forall i \in \overline{2, k-1}$ $p_{i-1}p_{i+1} \in E$, откуда следует, что $\forall i \in \overline{1, k-1}$ $p_i p_{i+1} \in E$, так как $t_{p_{i+1}} - t_{p_i} \leq t_{p_{i+1}} - t_{p_{i-1}}$. Таким образом, B — связный подграф.

Докажем, что в B нет точек сочленения. Рассмотрим произвольную вершину $p_j \in B$. В графе B' , индуцированном множеством $P(B) \setminus \{p_j\}$, вершины p_{j-1} и p_{j+1} соединены ребром, а значит, B' связен. Из этого следует, что любая вершина подграфа B не является точкой сочленения, и, следовательно, B — неразделимый подграф.

Замечание. В случаях $|P(B)| = 1$ и $|P(B)| = 2$ подграф B является неразделимым, так как в принципе не может содержать точек сочленения. \square

3.2 Выделение блоков в графе G_w

Теперь найдем в графе G_w максимальный неразделимый подграф, то есть блок.

Лемма 2. (Критерий максимальности неразделимого графа) Пусть B — неразделимый подграф графа G_w^o , $P(B) = \{p_1, p_2, \dots, p_k\}$, $k \in \mathbb{N}$. Тогда следующие условия эквивалентны:

1. B — максимальный неразделимый подграф (блок).
2.
 - Либо $P(B) = \{p_1\}$, где p_1 — изолированная вершина,
 - либо $\forall p \in P_w \setminus P(B) \implies t_p < t_{p_1}$, $pp_2 \notin E$ или $t_p > t_{p_k}$, $p_{k-1}p \notin E$. То есть вершины графа G_w^o , не вошедшие в B , либо меньше, либо больше всех вершин из B , и при этом они не удовлетворяют критерию неразделимости (см. рисунок 3.4).

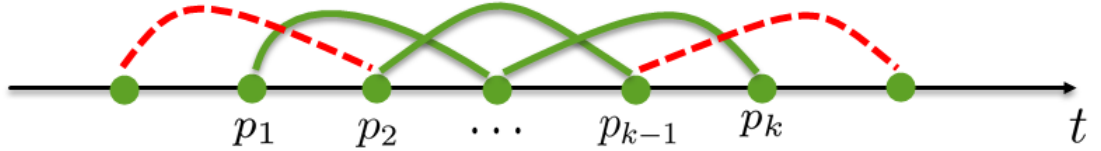


Рис. 3.4: Критерий максимальности неразделимого графа.

Доказательство. ($1 \Rightarrow 2$) Пусть B — максимальный неразделимый подграф. Возьмем вершину $p \in P_w \setminus P(B)$.

Пусть $k = 1$. Из максимальности B следует, что $pp_1 \notin E$, а значит, p_1 — изолированная вершина.

Пусть $k > 1$. Далее рассмотрим граф B' , индуцированный множеством $P(B) \cup \{p\}$. Максимальность B означает, что для B' свойство неразделимости нарушается.

Пусть p между p_1 и p_k .

Пусть $k = 2$. Из связности подграфа B следует, что $p_1p_2 \in E$, что в свою очередь по замечанию 1 влечет существование ребер p_1p и pp_2 . Тогда по лемме 1 граф B' является неразделимым подграфом графа G_w^o , что противоречит максимальности B .

Пусть $k > 2$. Тогда $\exists i \in \overline{1, k-1}$, что p между p_i и p_{i+1} .

Если $i \in \overline{2, k-2}$, то по лемме 1 имеем $p_{i-1}p_{i+1} \in E$ и $p_ip_{i+2} \in E$. Это в силу замечания 1 влечет $p_ip_{i+1} \in E$. Из существования ребер $p_{i-1}p_{i+1}, p_ip_{i+2}, p_ip_{i+1}$ следует,

что существуют ребра $p_{i-1}p, p_ip, pp_{i+1}$ и pp_{i+2} , а значит, в силу леммы 1 граф B' является неразделимым подграфом графа G_w^o , что противоречит максимальнойности B .

Если $i = 1$, то по лемме 1 имеем $p_1p_3 \in E$. Это в силу замечания 1 влечет существование ребер p_1p, pp_2 и pp_3 , а значит, по лемме 1 B' является неразделимым подграфом графа G_w^o , что противоречит максимальнойности B .

Случай $i = k - 1$ аналогичен случаю $i = 1$.

Показано, что при любом $k > 1$ p не может быть между p_1 и p_k , то есть либо p левее p_1 , либо p правее p_k .

Пусть p левее p_1 . Если $pp_2 \in E$, то в силу леммы 1 B' — неразделимый подграф. Аналогично, если p правее p_k и $p_{k-1}p \in E$, то B' — неразделимый подграф. Эти условия противоречат максимальнойности B .

Таким образом, если $p \in P_w \setminus P(B)$ и $k > 1$, то либо p левее p_1 и $pp_2 \notin E$, либо p правее p_k и $p_{k-1}p \notin E$.

(1 \Leftarrow 2) Возьмем вершину $p \in P_w \setminus P(B)$ и рассмотрим граф B' , индуцированный множеством $P(B) \cup \{p\}$.

Условие $P(B) = \{p_1\}$, где p_1 — изолированная вершина, означает, что B' не является связным, следовательно, B — максимальный неразделимый подграф.

Условие $\forall p \in P_w \setminus P(B) \implies t_p < t_{p_1}, pp_2 \notin E$ или $t_p > t_{p_k}, p_{k-1}p \notin E$ означает, что вершина p не удовлетворяет 2 пункту леммы 1, а значит, граф B' не является неразделимым, следовательно, B — максимальный неразделимый подграф. \square

Следствие. Свойства блоков графа G_w^o .

Свойство 1. Пусть B — блок графа G_w^o , $p, q \in P(B)$ и вершина p левее q . Тогда для любой вершины $r \in P_w$, если вершина r между вершинами p и q , то r принадлежит блоку B (рисунок 3.5).

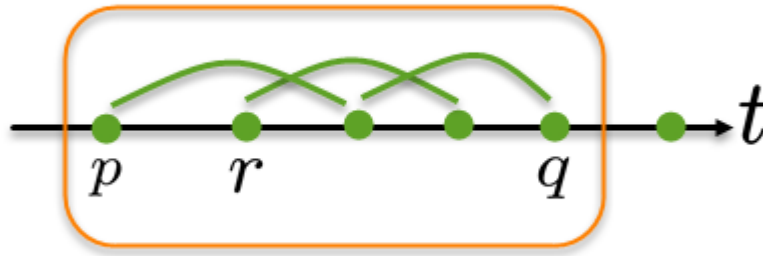


Рис. 3.5: Свойства блока графа G_w^o .

Доказательство. Упорядочим вершины блока B по возрастанию значений функции t : $P(B) = (b_1, b_2, \dots, b_i = p, \dots, b_j = q, \dots, b_k)$. Поскольку B - блок, в графе G_w^o существует цепь $h = b_i b_{i+1} \dots b_{j-1} b_j$. Вследствие того, что r между p и q , найдется такое m : $i \leq m \leq j-1$, что r между b_m и b_{m+1} . Так как h - цепь, то существует ребро $b_m b_{m+1}$. А значит, по замечанию 1, существуют ребра $b_m r$ и $r b_{m+1}$. Следовательно, по критерию неразделимости вершина r принадлежит блоку B . \square

Свойство 2. Два блока графа G_w^o могут иметь не более одной общей вершины (рисунок 3.6).

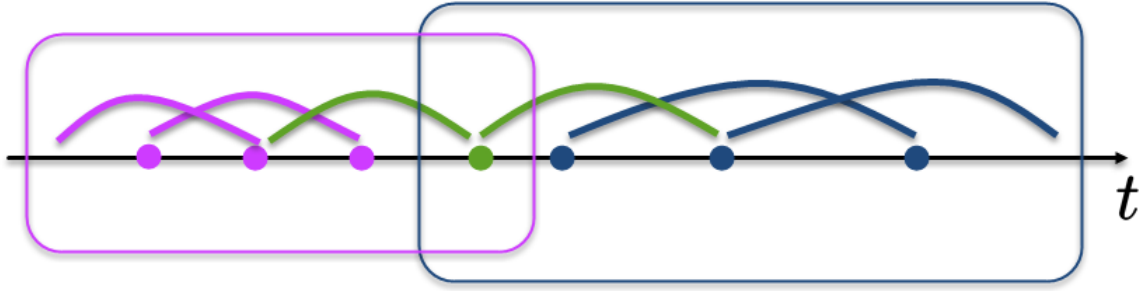


Рис. 3.6: Блоки графа G_w^o .

Доказательство. Пусть в графе G_w^o имеется два блока B_1 и B_2 .

1) Пусть p - произвольная вершина блока B_1 . Тогда либо p не правее всех вершин блока B_2 , либо p не левее всех вершин блока B_2 , либо находится между некоторыми двумя вершинами блока B_2 . Если вершина p находится между некоторыми двумя вершинами блока B_2 , то по свойству 1 вершина p принадлежит блоку B . Это противоречит максимальности блока, а значит любая вершина блока B_1 не левее или не правее любой вершины блока B_2 . 2) Пусть все вершины блока B_1 не правее всех вершин блока B_2 . Предположим, что блоки B_1 и B_2 имеют две общие вершины p и q . Тогда p и q являются самыми правыми в блоке B_1 и самыми левыми в блоке B_2 , то есть $P(B_1) = \{p_1, p_2, \dots, p_{k-1} = p, p_k = q\}$, $P(B_2) = \{q_1 = p, q_2 = q, \dots, q_{l-1}, q_l\}$ и $p_1 \leq p_2 \leq \dots \leq p \leq q \leq \dots \leq q_{l-1} \leq q_l$. В силу того, что B_1 и B_2 - блоки, существуют ребра $p_{k-2}q$ и $p q_3$, а значит по критерию неразделимости B_1 и B_2 образуют один блок. Это противоречит максимальности блока. Поэтому блоки B_1 и B_2 могут иметь не более 1 общей вершины. \square

3.3 Простые цепи в графе G_w .

Для доказательства следующей леммы удобно ввести определение. Пусть $H_1 = p_1 p_2 \dots p_n$ и $H_2 = q_1 q_2 \dots q_m$ - две произвольные простые цепи. Если H_1 и H_2 не имеют общих вершин и $p_n q_1 \in E$, то *суммой* H_1 и H_2 назовем простую цепь $H = p_1 \dots p_n q_1 \dots q_m$. Если H_1 и H_2 имеют одну общую вершину $p_n = q_1$, то *суммой* H_1 и H_2 назовем простую цепь $H = p_1 \dots p_n q_2 \dots q_m$. Если указанные условия не выполнены, то сумма цепей не определена. Обозначим сумму цепей $H = H_1 + H_2$. Легко видеть, что сумма цепей является ассоциативной и некоммутативной частичной операцией.

Лемма 3. Пусть B — блок графа G_w^o , различные вершины s и e принадлежат блоку B , вершина s не правее e . Справедливы следующие утверждения :

1. Если существует хотя бы одна вершина $q \in P(B)$, отличная от s и e и такая, что q между s и e , то существует простая (s, e) -цепь, содержащая все вершины блока.
2. Если для любой вершины $q \in P(B)$, отличной от s и e , справедливо, что s не правее q , то существует простая (s, e) -цепь, содержащая все вершины блока.
3. Если для любой вершины $q \in P(B)$, отличной от s и e , справедливо, что q не левее e , то существует простая (s, e) -цепь, содержащая все вершины блока.

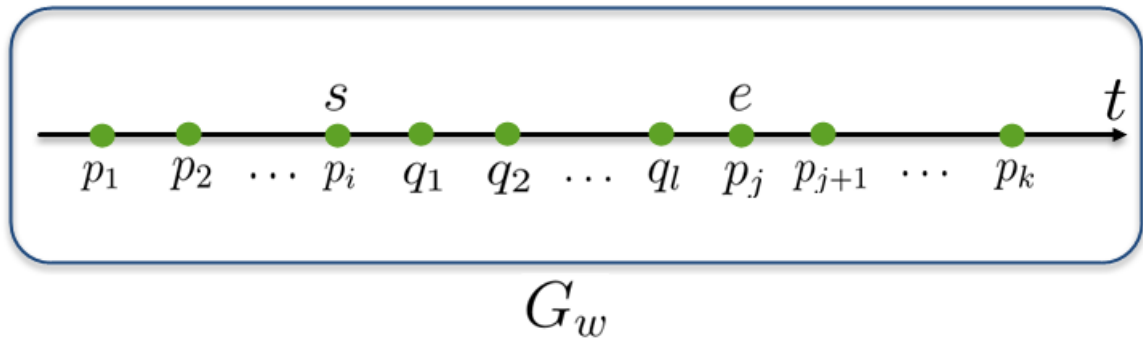


Рис. 3.7: Максимальная простая цепь в блоке графа G_w^o .

Доказательство. Рассмотрим вершины блока в порядке увеличения значений функции t .

1. Рассмотрим множество вершин блока $P(B) = (p_1, p_2, \dots, p_i = s, q_1, q_2, \dots, q_l, p_j = e, p_{j+1}, \dots, p_k)$. По условию, между s и e существует хотя бы одна вершина, а значит $j \geq 1$. Построим простую (s, e) - цепь H , содержащую все вершины блока B .
 - (а) Для начала построим простую (s, q_1) -цепь, содержащую все вершины блока B , которые не правее s . Пусть цепь H_1 устроена следующим образом:

будем «шагать» из вершины s «назад» через одну вершину, пока не достигнем первой вершины блока p_1 , затем будем «шагать» через одну вершину «вперед», пока не достигнем вершины q_1 . Эта цепь является простой, поскольку вершины не повторяются. В простой цепи H_1 все ребра существуют, поскольку B - блок, а значит по критерию неразделимости существуют ребра $p_{m-1}p_{m+1}$ для любого $1 < m < i$.

Простая цепь H_1 имеет следующий вид.

$$H_1 = \begin{cases} p_i p_{i-2} p_{i-4} \dots p_2 p_1 p_3 p_5 \dots p_{i-3} p_{i-1} q_1, & \text{если } i \text{ — четное;} \\ p_i p_{i-2} p_{i-4} \dots p_3 p_1 p_2 p_4 \dots p_{i-3} p_{i-1} q_1, & \text{если } i \text{ — нечетное.} \end{cases}$$

Легко видеть, что H_1 содержит в себе все p_m для любого $1 \leq m \leq i$, а значит все вершины левее s , кроме q_1 .

- (b) Теперь построим простую (q_l, e) -цепь, содержащую все вершины блока B , которые не левее e . Пусть цепь H_2 устроена следующим образом: будем «шагать» из вершины q_l «вперед» через одну вершину, пока не достигнем последней вершины блока p_k , затем будем «шагать» через одну вершину «назад», пока не достигнем вершины e . Эта цепь является простой, поскольку вершины не повторяются. В простой цепи H_2 все ребра существуют, поскольку B - блок, а значит по критерию неразделимости существуют ребра $p_{m-1}p_{m+1}$ для любого $j < m < k$.

Простая цепь H_2 имеет следующий вид.

$$H_2 = \begin{cases} q_l p_{j+1} p_{j+3} \dots p_{k-2} p_k p_{k-1} p_{k-3} \dots p_{j+4} p_{j+2} p_j, & \text{если } (k-j) \text{ — четное;} \\ q_l p_{j+1} p_{j+3} \dots p_{k-1} p_k p_{k-2} p_{k-4} \dots p_{j+4} p_{j+2} p_j, & \text{если } (k-j) \text{ — нечетное.} \end{cases}$$

В H_2 содержатся все вершины правее e , кроме q_l .

- (c) И наконец, построим простую цепь из q_1 в q_l , содержащую все вершины блока B между s и e . Поскольку B -блок, существуют ребра вида $q_{m-1}q_{m+1}$, а значит существуют ребра $q_{m-1}q_m$ для любого $1 < m \leq l$. Поэтому $H_3 = q_1 q_2 \dots q_l$ - это простая (q_1, q_l) -цепь, содержащая все вершины блока B между s и e .
- (d) Цепи H_1 и H_3 имеют общую вершину q_1 , а цепи H_2 и H_3 имеют общую вершину q_l . Следовательно, можно построить простую (s, e) -цепь H , которая является суммой цепей H_1 , H_3 и H_2 .

$$H = H_1 + H_3 + H_2.$$

На рисунке 3.8 изображен схематический пример построения простой цепи в блоке B .

В силу определения суммы простых цепей и того, что в совокупности цепи H_1 , H_2 и H_3 содержат в себе все вершины блока B , $H = H_1 + H_3 + H_2$ - (s, e) -цепь, содержащая все вершины блока B .

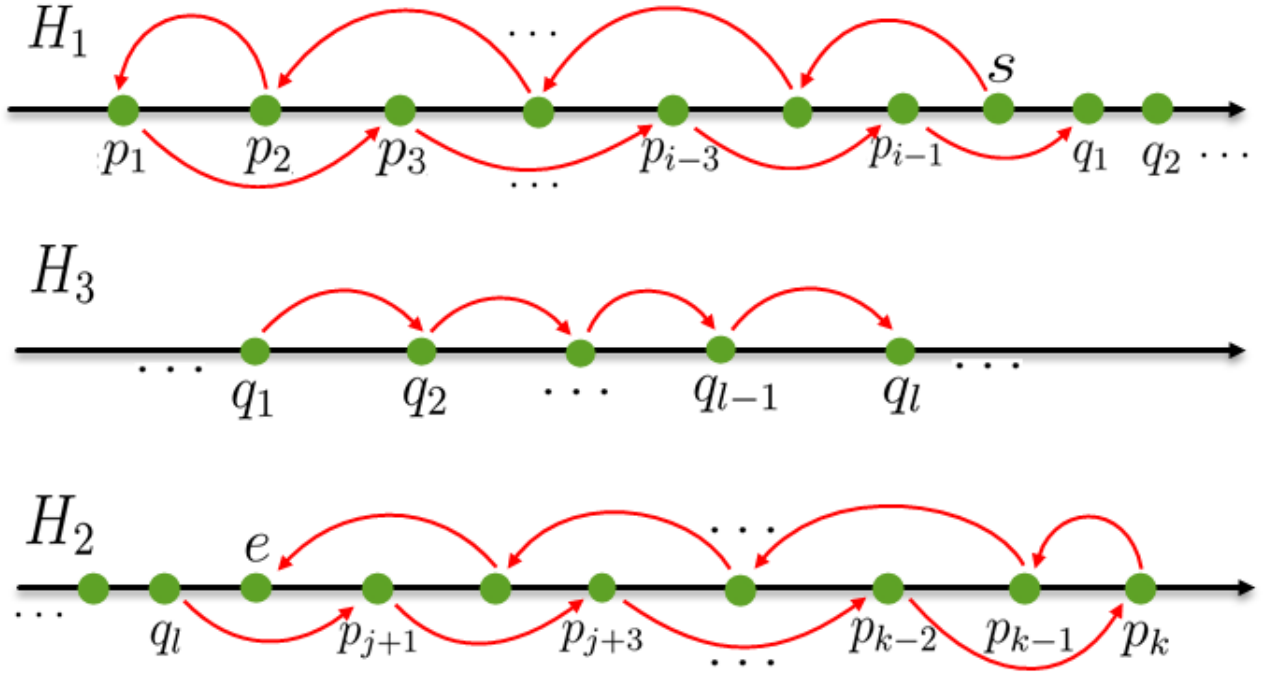


Рис. 3.8: Построение максимальной простой цепи в блоке графа G_w^o .

2. Рассмотрим множество вершин блока B , в котором все вершины не левее вершины s , то есть $P(B) = (p_1 = s, \dots, p_j = e, p_{j+1}, \dots, p_k)$. Построим простую (s, e) -цепь, содержащую все вершины блока B .

- (a) Пусть $j = 1$. Тогда $s = e$, что не возможно в силу условий леммы.
- (b) Пусть $j > 2$. Это значит, что существует хотя бы одна вершина q , отличная от s и e и такая, что q между s и e . Тогда по уже доказанному пункту 1 леммы 3 существует простая (s, e) -цепь H , содержащая все вершины блока.
- (c) Пусть $j = 2$. Тогда множество вершин $P(B) = (s, e, q_1, q_2, \dots, q_m)$. Этот случай аналогичен случаю построения (q_l, e) -цепи H_2 в пункте 1 леммы, где в качестве q_l выступает вершина s . Таким образом, простая (s, e) -цепь H устроена так:

$$H = \begin{cases} sq_1q_3 \dots q_{m-3}q_{m-1}q_mq_{m-2} \dots q_2e, & \text{если } m \text{ — четное;} \\ sq_1q_3 \dots q_{m-2}q_mq_{m-1}q_{m-3} \dots q_2e, & \text{если } m \text{ — нечетное.} \end{cases}$$

В H содержатся все вершины блока B .

3. Теперь рассмотрим множество вершин блока B , в котором все вершины не правее вершины e , то есть $P(B) = (p_1, p_2, \dots, p_j = s, p_{j+1}, \dots, p_k = e)$. Построим простую (s, e) -цепь, содержащую все вершины блока B .

- (a) Пусть $j = k$. Тогда $s = e$, что не возможно в силу условий леммы.
- (b) Пусть $j < k - 1$. Это значит, что существует хотя бы одна вершина q , отличная от s и e и такая, что q между s и e . Тогда по уже доказанному пункту 1 леммы 3 существует простая (s, e) -цепь H , содержащая все вершины блока.

(с) Пусть $j = k - 1$. Тогда множество вершин $P(B) = (q_1, q_2, \dots, q_m, s, e)$.

Этот случай аналогичен случаю построения (s, q_1) - цепи H_1 в пункте 1 леммы, где в качестве q_1 выступает вершина e . Таким образом, простая (s, e) -цепь H устроена так:

$$H = \begin{cases} sq_{m-1}q_{m-3} \dots q_1q_2q_4 \dots q_me, & \text{если } m \text{ — четное;} \\ sq_{m-1}q_{m-3} \dots q_2q_1q_3 \dots q_me, & \text{если } m \text{ — нечетное.} \end{cases}$$

В H содержатся все вершины блока B .

□

Лемма 4. Пусть B — блок графа G_w^o , H — простая (s, e) - цепь в графе G . Пусть $M \subset B$ — множество всех вершин блока B , которые входят в простую цепь H . Тогда для любых $p, q \in M$ не найдется такая вершина r , вошедшая в H и не вошедшая в M , что цепь H имеет вид $H = s \dots p \dots r \dots q \dots e$. Другими словами, любая простая цепь не может выйти из блока, а потом снова зайти в него.

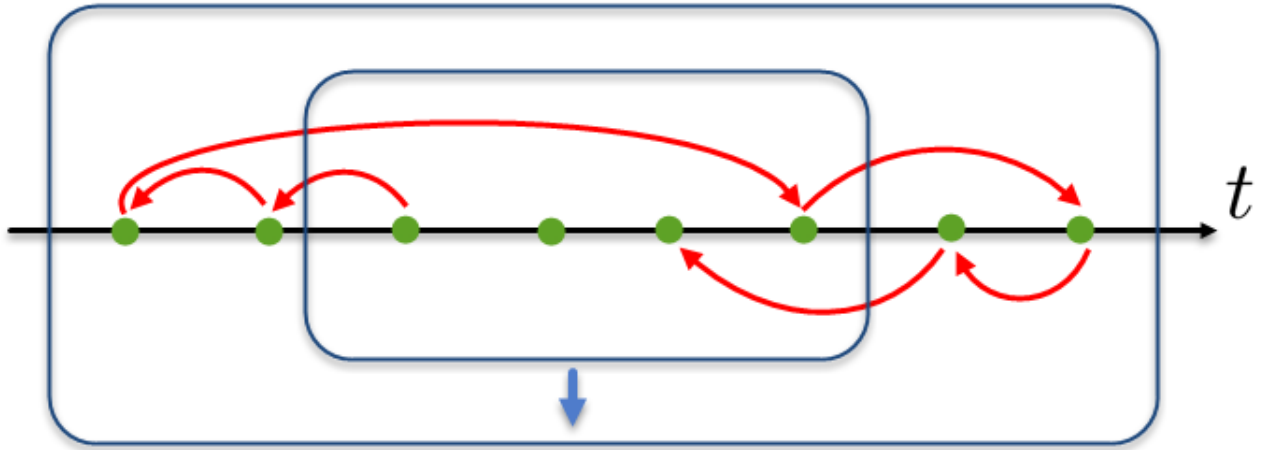


Рис. 3.9: Вхождение блока в простую цепь графа G_w^o .

Доказательство. Допустим противное: пусть в цепи H нашлась такая вершина $r \notin M$, что $H = s \dots p \dots r \dots q \dots e$. Из условия, что $r \notin M$ следует, что r не содержится в блоке B , а значит r не может быть между p и q по **свойству 1.**

Из представления $H = s \dots p \dots r \dots q \dots e$ следует, что существует простая (p, r) -цепь H_1 и простая (r, q) - цепь H_2 в графе G_w^o .

Поскольку p и q выбираются произвольно, будем считать, что $p \leq q$.

1. Пусть M_1 - множество вершин, попавших в цепь H_1 и не попавших в блок B . Пусть M_2 - множество вершин, попавших в цепь H_2 и не попавших в блок B . Заметим, что множества M_1 и M_2 не имеют общих вершин, потому что H - простая цепь.

2. Рассмотрим граф G' , порожденный множеством $P(B) \cup M_1 \cup M_2$. Докажем, что в графе G' нет точек сочленения. Допустим противное: пусть нашлась вершина v , которая является точкой сочленения графа G' .
3. Пусть $v \in M_1$. При удалении вершины v из графа G' все вершины из M_1 будут связаны с остальными вершинами графа G' через вершину q , а значит количество компонент связности не увеличится.
4. Пусть $v \in M_2$. При удалении вершины v из графа G' все вершины из M_2 будут связаны с остальными вершинами графа G' через вершину p , а значит количество компонент связности не увеличится.
5. Пусть $v \in P(B)$. Если вершина v не связана с вершинами из M_1 и M_2 , то ее удаление не увеличит количество компонент связности, так как B - блок. Если вершина v соединена ребрами с вершинами из M_1 , то при ее удалении вершины множества M_1 будут связаны с остальными вершинами графа G' через вершину q . Аналогично, если вершина v соединена ребрами с вершинами из M_2 , то при ее удалении вершины множества M_2 будут связаны с остальными вершинами графа G' через вершину p . Следовательно, количество компонент связности не увеличится.

Рассмотрим последний случай, когда $v \in P(B)$ и при этом v связана ребрами и с вершинами из M_1 и с вершинами из M_2 . Поскольку H - простая цепь, в ней нет повторяющихся вершин, а значит удаление v не может удалить связность одновременно и у вершин из M_1 , и у вершин из M_2 .

Следовательно, в любом случае при удалении вершины v количество компонент связности графа G' не увеличится, а значит G' - неразделимый подграф, что противоречит максимальнойности блока B . Отсюда следует, что предположение о существовании вершины r было неверным. \square

3.4 Максимальные простые цепи в графе G

Теорема. Пусть H — максимальная по включению вершин простая цепь в G , $w \in W$. Тогда для любого блока B графа G_w^o либо H содержит все вершины из B , либо ровно одну вершину, либо H и B не имеют общих вершин.

Доказательство. Рассмотрим произвольный блок B и максимальную по включению вершин простую цепь H в G . Предположим, что B имеет с H не менее двух общих вершин. Докажем, что в этом случае все вершины из B содержатся в H .

В силу леммы 4, если некоторое количество вершин из блока содержится в простой цепи, то они обязательно идут подряд. Значит, простая цепь H представляет собой последовательность вершин, где каждая вершина принадлежит некоторому блоку, и этой последовательности соответствует последовательность неповторяющихся блоков.

Рассмотрим две вершины s и e такие, что s - первая вершина из блока B в пути H , а e - последняя вершина из блока B в пути H . Другими словами s и e — вход и выход в блок B в простой цепи H (см. рисунок 3.10). Пусть $s \leq e$. Если эти вершины удовлетворяют условиям леммы 3, тогда существует простая (s, e) - цепь, содержащая все вершины блока B , а значит в силу максимальнойности H все вершины блока B должны в нее попасть.

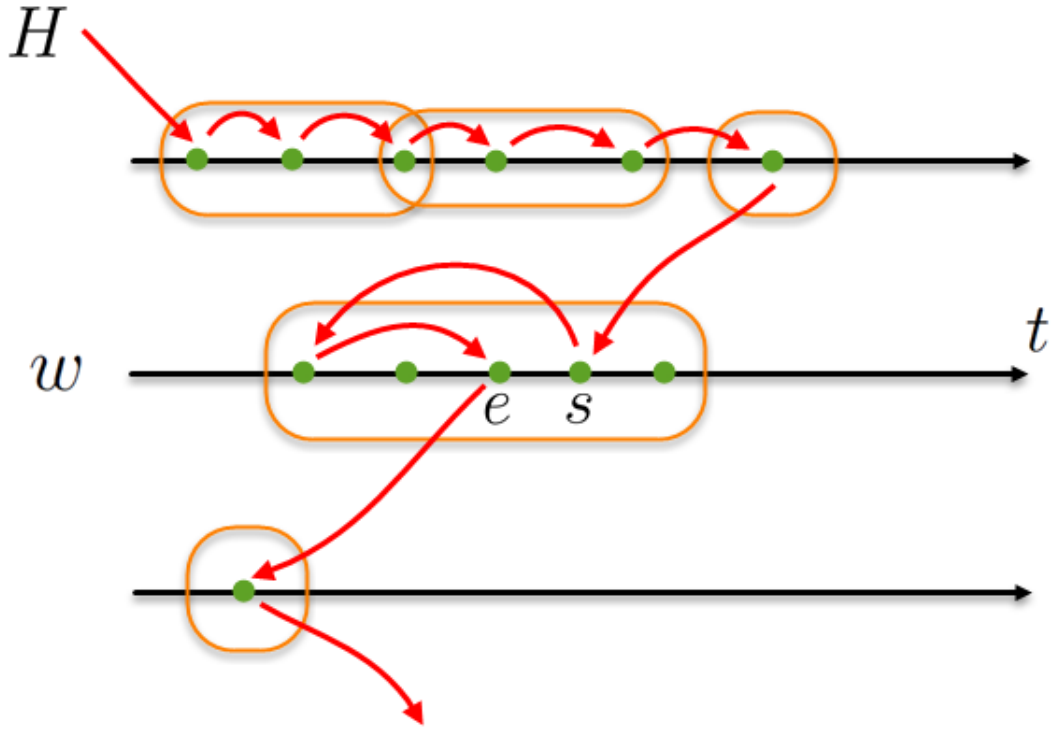


Рис. 3.10: Простая цепь в графе G , с входом s и выходом e в блоке B .

Рассмотрим случай, когда s и e не удовлетворяют условиям леммы 3, то есть для любой вершины $h \in B$, либо $h < s$, либо $h > e$, причем существует хотя бы одна вершина $s' \in B : s' < s$ и существует хотя бы одна вершина $e' \in B : e' > e$. Среди всех таких s' и e' будем рассматривать те, что ближе всего к s и e соответственно. Тогда выберем такую s' - самую правую среди вершин блока B , которые левее s , и такую вершину e' - самую левую среди вершин блока B , которые правее e . А это значит, что существуют ребра $s's$ и ee' и в силу свойств блока также существуют ребра $s'e$ и se' .

Пусть p — вершина, предшествующая s в простой цепи H . Так как s - первая вершина из блока B в пути H , то $w_p \leq w_s$. Предположим, что существует дуга ps' . По лемме 3 существует простая (s', e) -цепь, содержащая все вершины блока B . Тогда если рассмотреть простую цепь H' , которая до вершины p совпадает с H и после вершины e совпадает с H , а между ними совпадает с максимальной по включению вершин (s', e) -цепью (см. рисунок 3.11), то цепь H' будет максимальной по включению в G и при этом будет содержать все вершины из B , а следовательно, и путь H в силу максимальной будет содержать все вершины из B .



1. Из того, что существует ребро $s'e$ (поскольку p является предшественником s в цепи H) следует, что p не меньше s' . В противном случае в силу **замечания 1** должно существовать ребро ps' .
2. Если p между s' и e , то тогда по **замечанию 1** должно существовать ребро ps' .
3. Пусть $p > e$. Пусть q — следующая после e вершина в простой цепи H . То есть $w_q \leq w_e$.

- Таким образом, если не существует ребра $e'q$, то существует ребро $s'q$ ((см. рисунок 3.12)).

- (а) $p > e$. Если p между s и e' , то существует ребро pe' .

- (b) Если $p \geq e'$, то e' между s и p , и так как существует ребро ps , то существует ребро pe' .

Таким образом, существует ребро pe' ((см. рисунок 3.12)).

Заметим, что по лемме 3 существует простая (s', e') -цепь, содержащая все вершины блока B . Тогда если рассмотреть простую цепь H' , которая до вершины p совпадает с H и после вершины q совпадает с H , а между ними совпадает с максимальной (s', e') -цепью, то простая цепь H' (см. рисунок 3.12) будет максимальной по включению вершин в G и при этом будет содержать все вершины из B , а следовательно, и путь H в силу максимальнойности будет содержать все вершины из B .

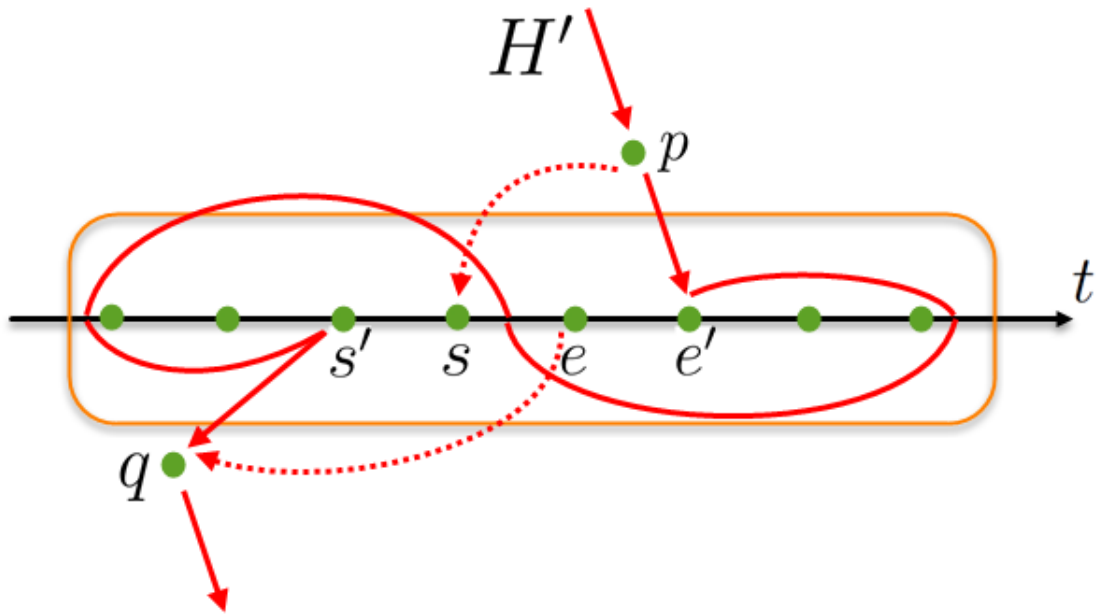


Рис. 3.12: Простая максимальная цепь H' .

5. Случай $s > e$ аналогичен случаю $s < e$, поскольку в блоке B любую простую (s, e) - цепь можно рассмотреть в обратном порядке как простую (e, s) - цепь. Таким образом, заменив s на e , а e на s , мы снова получим простую максимальную по включению вершин цепь H , содержащую все вершины блока.

□

Глава 4

Программная реализация

В данной главе рассматриваются реализация алгоритма поиска блоков в графе G и результаты численных экспериментов.

4.1 Алгоритм построения блоков

4.1.1 Построение графа G

Для начала определим основные структуры данных, используемые в алгоритме поиска блоков графа G .

1. *Vertex* p представляет вершину p в графе G . Определим для такой вершины
 - $p.width$ - значение функции w , то есть ширина соответствующей партии p .
 - $p.thick$ - значение функции t , то есть толщина соответствующей партии p .
 - $p.measure$ - значение функции l , то есть полезность соответствующей партии p .
 - $p.flowType$ - вид готовой продукции, которой станет партия p после прокатки.
2. *Edge* e представляет ребро $e = pq$ в графе G . Определим для ребра e
 - $e.from$ - вершина *Vertex* p , являющаяся началом ребра e .
 - $e.to$ - вершина *Vertex* q , являющаяся концом ребра e .
3. *Crit* предназначен для проверки того, может ли вершина q следовать за вершиной p в графике прокатки. *Crit* содержит в себе следующие функции:
 - $widthCrit(Vertex\ from, Vertex\ to)$ - функция, отвечающая на вопрос "удовлетворяют ли ограничению «для перехода по ширине» вершины $from$ и to ?"
 - $thicknessCrit(Vertex\ from, Vertex\ to)$ - функция, отвечающая на вопрос "удовлетворяют ли ограничению «для перехода по толщине» вершины $from$ и to ?"
 - $test(Vertex\ from, Vertex\ to)$ - функция, отвечающая на вопрос "может ли вершина to непосредственно следовать за вершиной $from$ в графике прокатки?". Данная функция проверяет, что для различных вершин $from$ и to выполняются с положительными результатами функции $widthCrit(from, to)$ и $thicknessCrit(from, to)$. Таким образом, функция $test(from, to)$ определяет набор ребер в графе G .

4. *Graph g* представляет граф G с вершинами $g.vertex$ и ребрами $g.edges$.

Более подробное описание функций, используемых для построения графа G предложено в работах [2] и [4].

4.1.2 Поиск блоков в графе G

Для построения блока *Block b* определим следующие функции.

- Функция $b.addR(Vertex v)$ проверяет, можно ли добавить вершину v к блоку b справа. Если можно, то $addR(v)$ добавляет v к множеству вершин блока $b.vert$, упорядоченных по возрастанию функции t , и возвращает *true*, а в противном случае - возвращает *false*.

```
function addR(Vertex v):boolean
  var int k=vert.size()-1
  begin
    if (k = 0) then
      vert.add(v)
      return true
    else
      \\если вершина v не левее вершин из блока,
      \\то проверяем критерий неразделимости с двумя последними вершинами блока
      if (v.thick >= vert(k).thick) then
        if thicknessCrit(v, vert(k)) = true then
          if thicknessCrit(v, vert(k-1)) = true then
            vert.add(v)
            return true
          else
            return false
        else
          return false
      else
        return false
    end
```

Описанная функция позволяет для заданного набора вершин одной ширины строить блок начиная с какой-то вершины вправо. Таким образом, можно выбрать самую маленькую по толщине вершину и построить блок, добавляя последовательно вершины при помощи функции *addR* до тех пор, пока она не вернет *false*.

- Опишем функцию *getBlocks*, которая по заданному набору вершин в одной ширине, упорядоченных по возрастанию толщины, то есть вершин графа G_w^o , возвращает набор блоков этого графа.

```
function getBlocks(Vertex[] vertex):Block[] blocks
    var Block block, Block[] blocks, int k = 0, int n = vertex.size()
    begin
        while (k < n) do
            while block.addR(vertex[k]) = true do
                k = k + 1
            blocks.add(block)
            if test(vertex[k-1], vertex[k+1]) = true then
                k = k - 1
            block = new Block
        return blocks
    end
```

Функция *getBlocks(vertex)* работает следующим образом:

1. Берется вершина $v = vertex[0]$ - самая левая в *vertex*. К ней последовательно добавляются вершины из *vertex* при помощи функции *addR* до тех пор, пока не будет построен первый блок. Заметим, что последняя (самая правая) вершина блока в множестве *vertex* будет иметь позицию $k - 1$.
2. Следующий блок может начинаться либо с вершины $vertex[k - 1]$, либо с вершины $vertex[k]$ в силу **свойства 2** блоков.
3. Если существует ребро $vertex[k - 1]vertex[k + 1]$, то по критерию неразделимости вершина $vertex[k - 1]$ должна принадлежать следующему блоку. Поэтому при выполнении условия $test(vertex[k - 1], vertex[k + 1]) = true$ необходимо вернуться на позицию назад.
4. В противном случае, следующий блок начнется с вершины $vertex[k]$.
5. После построенного второго блока, будет строиться третий и так далее до тех пор, пока алгоритм не просмотрит все вершины из *vertex*.

4.2 Результаты расчетов

Расчеты, приведенные в данной главе, проводились на базе данных Магнитогорского Металлургического Комбината (ММК). Выбиралось произвольное задание, содержащее в себе некоторое количество партий, из которых необходимо собрать оптимальный график прокатки. Для каждой партии использовались ее ширина, толщина, вес (полезность) и вид готовой продукции. Также в задании хранилась информация о величине δ и функции $r(t)$, задающие ограничения на порядок партий в графике. Значения δ и $r(t)$, приведенные в примере из главы 1.2, были также взяты из заданий и использованы при расчетах.

4.2.1 Построение графа G

В данном пункте приведен фрагмент графа G , построенного с учетом всех ограничений на порядок партий в графике прокатки.

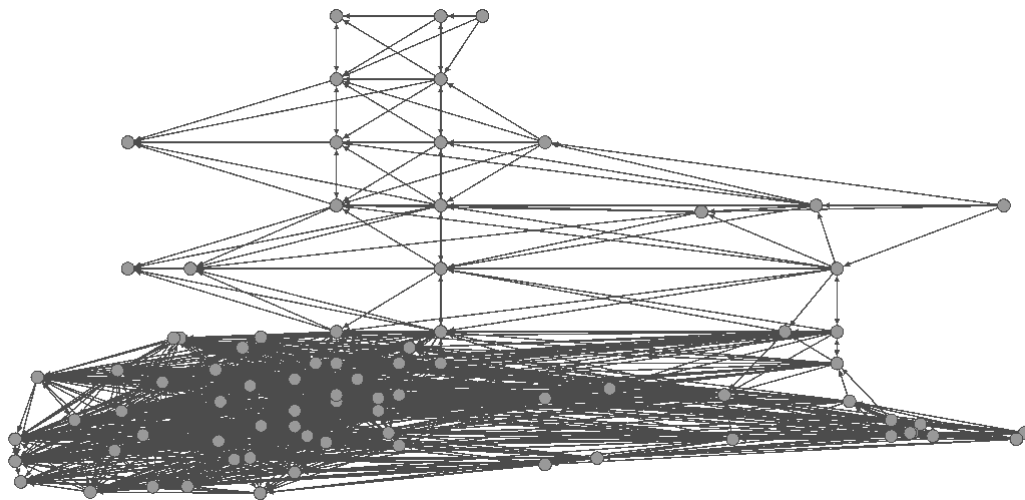


Рис. 4.1: Пример графа G .

На рисунке 4.1 видно, что граф содержит в себе очень большое количество ребер. Обычно на 300 партий (среднее количество партий в задании), другими словами на граф G , содержащий 300 вершин, приходится порядка 20000 ребер.

Из данной статистики видно, что поиск оптимального графика прокатки в графе G обычным перебором практически невозможен за разумное время.

4.2.2 Построение блоков графа G

Было выбрано некоторое количество заданий из базы данных ММК. По каждому заданию были построены граф G и его блоки.

Таблица 4.1 показывает количество вершин и количество блоков в графах G .

В среднем, отношение количества блоков к количеству вершин графа G равно 0.35. Из приведенной статистики видно, что размерность задачи значительно уменьшается при использовании блоков.

Таблица 4.1: Результаты расчетов

Количество вершин	Количество блоков
236	83
228	78
199	71
177	71
147	50
136	50
331	94
251	83
203	65
175	58
152	55
419	115
398	113
345	106
250	100
191	92
276	119
235	98
217	89
195	83
181	87
268	111
236	90
212	84
310	121
306	116
270	110
243	104
211	89
270	96
257	91

Заключение

В настоящей работе предложена математическая модель задачи построения графика прокатки, сбалансированного по видам готовой продукции. Для решения данной задачи было проведено исследование свойств графика горячей прокатки. Был построен граф предшествования партий и изучена его иерархическая структура. Данная структура состоит из трех уровней:

- Граф предшествования партий,
- Граф партий в одной ширине,
- Блоки графа партий в одной ширине.

На основе доказанных лемм и теорем был предложен алгоритм, выполняющий поиск блоков в графе предшествования партий. Была осуществлена программная реализация данного алгоритма и проведены численные эксперименты, указывающие на то, что выявленная структура графа предшествования партий может существенно упростить задачу построения графика прокатки за счет уменьшения ее размерности.

Литература

- [1] *Асанов М.О., Баранский В.А., Расин В.В.* Дискретная математика: Графы, Матроиды, Алгоритмы. - Ижевск: НИЦ "РХД 2001, 288 стр.
- [2] *Березин А.А.* Разработка и исследование алгоритмов построения экстремальных цепей в вершинно-взвешенных ориентированных графах. Общй алгоритм//Бакалаврская диссертация, ИМКН УрФУ, Екатеринбург, 2014
- [3] *Березин А.А., Вакула И.А., Леонова С.И.* Задача планирования горячей прокатки//Труды 46-й Международной молодежной школы-конференции, ИММ УрО РАН, Екатеринбург, 2015
- [4] *Леонова С.И.* Разработка и исследование алгоритмов построения экстремальных цепей в вершинно-взвешенных ориентированных графах. Частный алгоритм//Бакалаврская диссертация, ИМКН УрФУ, Екатеринбург, 2014
- [5] Lixin Tang, Jiyin Liu, Aiyong Rong, Zihou Yang. A review of planning and scheduling systems and methods for integrated steel production//European Journal of Operational Research. Volume 133, Issue 1, 16 August 2001, PP. 1–20.
- [6] P. Cowling, W. Rezig. Integration of continuous caster and hot strip mill planning for steel production//Journal of Scheduling, Volume 3, Issue 4, July/August 2000, PP. 185–208.
- [7] E. Balas, The prize collecting travelling salesman problem//Networks 19, 1989, PP. 621-636.
- [8] E. Balas and H. M. Clarence. Combinatorial optimization in steel rolling//Workshop on Combinatorial Optimization in Science and Technology, April, 1991.
- [9] Shixin Liu. Model and Algorithm for Hot Rolling Batch Planning in Steel Plants//International Journal of Information and Management Sciences. 21 (2010), PP. 247-263.
- [10] G. B. Dantzig and J. H. Ramser. Management Science//Vol. 6, No. 1 (Oct., 1959), PP. 80-91.
- [11] Andreas Stenger, Michael Schneider, Dominik Goeke. The prize-collecting vehicle routing problem with single and multiple depots and non-linear cost//EURO Journal on Transportation and Logistics, May 2013, Volume 2, Issue 1-2, PP. 57-87.
- [12] K. Ioannidou, S. D. Nikolopoulos. The Longest Path Problem is Polynomial on Cocomparability Graphs//Algorithmica. January 2013, Volume 65, Issue 1, PP. 177-205. Rolling Batch Planning in Steel Plants//International Journal of Information and Management Sciences. 21 (2010), PP. 247-263.

Приложение

```
public Vertex(double width, double thick, int i) {
    this.width = width;
    this.thick = thick;
    this.id = i;
}

public Edge(Vertex from, Vertex to) {
    this.from = from;
    this.to = to;
}

public class Crit {

    public double w;

    public double r(double t) {
        if ((t >= 1.29) && (t <= 2))
            return 0.8;
        else if ((t > 2) && (t <= 3))
            return 1;
        else if ((t > 3) && (t <= 8))
            return 1.5;
        else if ((t > 8) && (t <= 20))
            return 2;
        return -1;
    }

    public boolean widthCrit(Vertex from, Vertex to) {□}

    public boolean thicknessCrit(Vertex from, Vertex to) {□}

    public boolean test(Vertex from, Vertex to) {□}
}

// ограничение "для перехода по ширине"
public boolean widthCrit(Vertex from, Vertex to) {
    if ((from.width - to.width <= w) && (from.width - to.width >= 0))
        return true;
    return false;
}
```

```

// ограничение "для перехода по толщине"
public boolean thicknessCrit(Vertex from, Vertex to) {
    double eps = 1e-3;
    if (Math.abs(to.thick - from.thick) <=
        Math.min(r(to.thick), r(from.thick)) + eps) {
        return true;
    } else {
        return false;
    }
}

// существует ли ребро e = fromto
public boolean test(Vertex from, Vertex to) {
    if ((from.id != to.id) && (widthCrit(from, to)) &&
        (thicknessCrit(from, to))) {
        return true;
    } else {
        return false;
    }
}

public class Graph {
    public ArrayList<Vertex> vertex;
    public ArrayList<Edge> edges;
    public ArrayList<ArrayList<Vertex>> pred;
    public ArrayList<ArrayList<Vertex>> succ;
    public ArrayList<ArrayList<Vertex>> in;
    public ArrayList<ArrayList<Vertex>> out;

    public void fill() {}

    public Graph(ArrayList<Vertex> argArrVert, Crit crit) {}

    public int cont(ArrayList<Vertex> arr, Vertex v) {}

    public boolean critEdge(Edge e) {}

    public void clearGraph() {}
}

public Graph(ArrayList<Vertex> argArrVert, Crit crit) {
    vertex = argArrVert;
    edges = new ArrayList<Edge>();

    for (int k = 0; k < vertex.size(); k++) {
        for (int j = 0; j < vertex.size(); j++) {
            if (crit.test(vertex.get(k), vertex.get(j))) {
                edges.add(new Edge(vertex.get(k), vertex.get(j)));
                out.get(k).add(vertex.get(j));
                in.get(j).add(vertex.get(k));
            }
        }
    }

    // заполнение списков pred и succ
    fillPredSucc();
}

```

```

// заполнение списков in и out
public void fillInOut() {
    in = new ArrayList<ArrayList<Vertex>>();
    out = new ArrayList<ArrayList<Vertex>>();
    for (int i = 0; i < this.vertex.size(); i++) {
        Vertex v = this.vertex.get(i);
        in.add(new ArrayList<Vertex>());
        out.add(new ArrayList<Vertex>());
        for (Edge e : this.edges) {
            if (e.to.equals(v)) {
                in.get(i).add(e.from);
            }
            if (e.from.equals(v)) {
                out.get(i).add(e.to);
            }
        }
    }
}

public boolean critEdge(Edge e) {
    Vertex u = e.from;
    Vertex v = e.to;
    ArrayList<Vertex> Out = out.get(u.id);
    ArrayList<Vertex> In = in.get(v.id);
    for (int i = 0; i < Out.size(); i++) {
        Vertex tu = Out.get(i);
        if ((tu.id != v.id) && (tu.width < u.width))
            for (int j = 0; j < In.size(); j++) {
                Vertex tv = In.get(j);
                if ((tv.id != u.id) && (tv.width > v.width))
                    if ((tu.width >= tv.width)
                        && ((contains(succ.get(tu.id), tv) > -1) ||
                            (tu.id == tv.id)))
                        return false;
            }
    }
    return true;
}

public void clearGraph() {
    for (int i = 0; i < edges.size(); i++) {
        Edge e = edges.get(i);
        if (!critEdge(e)) {
            edges.remove(i);
            i--;
        }
    }
    this.fill();
}

public Block(ArrayList<Vertex> vert) {
    this.vert = vert;
    Collections.sort(this.vert, new SortThick());
    minT = this.vert.get(0).thick;
    maxT = this.vert.get(this.vert.size() - 1).thick;
    width = this.vert.get(0).width;
}

```

```

//добавление вершины к блоку справа
public boolean addR(Vertex v) {
    int k = vert.size() - 1; //последняя позиция в блоке
    if (k == -1) { // если он был пустым, то просто добавляем эту вершину
        vert.add(v);
        return true;
    }
    if (v.thick >= vert.get(k).thick) {
        width = v.width;
        if (crit.thicknessCrit(v, vert.get(k))) {
            //если дотягивается до последней
            if (k == 0) { // одна вершина в блоке
                vert.add(v);
                return true;
            } else {
                k--;
                if (crit.thicknessCrit(v, vert.get(k))) {
                    vert.add(v);
                    return true;
                }
            }
        }
    }
    return false;
}

```

```

//получение блоков из заданного набора вершин в одной ширине
public static ArrayList<Block> getBlocks(ArrayList<Vertex> vertex,) {
    Collections.sort(vertex, new SortThick());
    ArrayList<Block> blocks = new ArrayList<Block>();
    Block block = new Block(step);
    int k = 0;
    while (k < vertex.size()) {
        while ((k < vertex.size()) && (s.addR(vertex.get(k)))) {
            k++;
        }
        if (test(vertex.get(k-1),vertex.get(k+1))) {
            k--;
        }
        blocks.add(block);
        block = new Block();
    }
    return blocks;
}

```