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. Алгоритмы построения графика

Допущен к защите

_____

«__»_____ 2016 г.

Квалификационная работа
на степень магистра наук
по направлению «Математика
и компьютерные науки»
студента группы МГКН-2
Березина Антона Александровича

Научный руководитель
Баранский Виталий Анатольевич,
доктор физико-математических
наук, профессор

Екатеринбург
2016

РЕФЕРАТ

Березин А.А. ЗАДАЧА ПОСТРОЕНИЯ ГРАФИКА ГОРЯЧЕЙ ПРОКАТКИ, СБАЛАНСИРОВАННОГО ПО ВИДАМ ПРОДУКЦИИ. АЛГОРИТМЫ ПОСТРОЕНИЯ ГРАФИКА,

выпускная квалификационная работа на степень магистра наук: стр. 41, рис. 5, форм. 22, библ. 11 назв., табл. 1.

Ключевые слова: ГОРЯЧАЯ ПРОКАТКА, ОРИЕНТИРОВАННЫЕ ГРАФЫ, МАТЕМАТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Объект исследования — раскрашенные вершинно-взвешенные ориентированные графы специального вида, возникающие при планировании расписания для стана горячей прокатки.

Цель работы — поиск в этих графах простых цепей, в которых суммарный вес вершин находится в заданном диапазоне, и суммарный вес вершин каждого цвета находится в соответствующем заданном диапазоне. Простая цепь, удовлетворяющая этим двум требованиям, называется сбалансированной.

Решение задачи поиска сбалансированной простой цепи сводится к решению серии задач математического программирования. Этот подход основан на результатах Леоновой С.И. по изучению структуры рассматриваемых графов ([2]) и позволяет существенно снизить размерность получаемых задач математического программирования по сравнению с прямым использовании классических техник для их построения (например, [7]). Полученные результаты применяются для решения более общих задач в области планирования металлургического производства.

Оглавление

Введение

В настоящей работе исследуется задача оперативного планирования работы непрерывных полосовых станов горячей прокатки стали.

Горячая прокатка стали это процесс производства полос различной ширины и толщины из стальных заготовок, имеющих форму параллелепипедов. В металлургии такие заготовки называются слябами. Обычно ширина сляба заметно больше его толщины. Во время прокатки слябы один за другим проходят через непрерывный полосовой стан горячей прокатки, который преобразует их в полосы давлением в клетях. Клеть это основной рабочий механизм стана, состоящий рабочих валков, которые непосредственно воздействуют на заготовку, и опорных валков, которые располагаются над и под рабочими валками и служат для увеличения жесткости валковой системы. При этом ширина прокатанной полосы близка к ширине сляба, из которого она получена, а ее толщина на порядок или два меньше толщины сляба. Длина полосы может исчисляться в сотнях метров, длина сляба от нескольких метров до десяти-двенадцати метров.

При планировании работы стана оперируют не отдельными слябами, а их группами, так называемыми партиями прокатки. Партия прокатки представляет собой набор слябов, имеющих одинаковые геометрические размеры и одинаковый химический состав (с погрешностью, которой можно пренебречь). Для всех слябов в партии заданы одни и те же значения ширины и толщины, которые должны получить прокатанные из них полосы на выходе из стана. Таким образом, слябы каждой 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. Во время прокатки слябы каждой партии следуют друг за другом непрерывно, не перемежаясь слябами из других партий, то есть партии прокатываются целиком и не дробятся на части.

Можно выделить два рабочих цикла стана. 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. Длительность цикла работы стана между двумя последовательными заменами рабочих валков составляет несколько часов. Задача оперативного планирования работы стана состоит в формировании последовательности партий, которая будет прокатана на стане в период между двумя очередными заменами рабочих валков. Такую последовательность партий называют графиком прокатки. Обычно в сутки требуется сформировать примерно четыре-шесть последовательных графиков прокатки.

В данной работе рассматривается ситуация, когда для каждой партии прокатки известно, для какого вида продукции она изготавливается, пойдут ли полосы партии сразу на отгрузку потребителю (после порезки) или будут направлены для дальнейшей обработки в другие цеха предприятия (например, травление, холодная прокатка и прочее). В зависимости от того, куда будет направлена партия после прокатки, ее относят к тому или другому виду продукции. Обычно оперируют десятью-

пятнадцатью видами продукции.

Для того, чтобы обеспечить выполнение заказов, отгрузку продукции непосредственному потребителю и равномерную загрузку сырьем для работы других цехов предприятия, при формировании каждого следующего графика прокатки создается так называемое задание на прокатку. Задание на прокатку включает в себя общее количество металла, которое должен содержать формируемый график, а также количество металла по отдельным видам продукции. Целевые значения количества металла рассчитываются на момент планирования исходя из текущего состояния складов полуфабрикатов, количества готовой продукции, состава уже сформированных графиков прокатки, информации о трудоемкости логистических операций, производительности последующих в технологических цепочках цехов и агрегатов и так далее. Целевые значения количества металла в графике прокатки даются вместе с диапазонами допустимого отклонения от них, составляющими несколько процентов. Обычно вместо целевого значения количества металла по какому-либо виду продукции рассматривают сразу диапазон допустимых значений. Например, задание на прокатку может выглядеть так: из данного множества партий, включающего в себя партии вида «a», «b» и «c», требуется сформировать график прокатки, в котором суммарная длина прокатанных полос находится в диапазоне от 100 до 120 километров, суммарная длина полос партий вида «a» находится в диапазоне от 35 до 40 километров, суммарная длина полос партий вида «b» находится в диапазоне от 50 до 60 километров, а на суммарную длину полос вида «c» ограничение не накладывается.

Существует несколько критериев оценки качества графика прокатки. В данной работе основным критерием оценки качества графика прокатки выбрана степень соответствия суммарной длины полос графика и суммарной длины полос партий по каждому виду продукции заданным диапазонам допустимых значений. Если в графике прокатки суммарная длина полос партий по каждому виду продукции находится в заданном диапазоне, то такой график прокатки будем называть сбалансированным по видам продукции, или просто сбалансированным.

Задача формирования сбалансированного графика прокатки является весьма нетривиальной, поскольку существует целый ряд технологических ограничений, которым график прокатки должен удовлетворять.

Основными технологическими ограничениями являются ограничения на допустимый порядок следования партий друг за другом в графике прокатки. В данной работе рассмотрены два из них, оба ограничения возникают из-за конструктивных особенностей стана горячей прокатки. Первое ограничение состоит в том, что требуется формировать только такие графики прокатки, в которых ширина каждой следующей партии не возрастает. Оно обусловлено тем, что во время прокатки рабочие валки стана сильно изнашиваются, и на их поверхности появляется углубление по ширине прокатываемых заготовок. Поэтому если после заготовки с меньшей шириной пропустить через стан заготовку с большей шириной, то на поверхности последней могут появиться дефекты. Также к первому ограничению относится требование, чтобы ширина каждой следующей партии в графике прокатки не уменьшалась больше, чем на заданную величину. Второе ограничение на порядок следования партий друг за другом в графике прокатки заключается в том, что толщина каждой следующей партии не должна слишком сильно отличаться от толщины предыдущей. Это ограничение обусловлено тем, что во время прокатки необходимо подстраивать расстояние между верхними и нижними валками стана в соответствии с толщиной прокатываемой заготовки, при этом хочется минимизировать эти перестроения валков.

Поскольку обычно производственный цикл металлургического предприятия является непрерывным, то при формировании каждого следующего графика прокат-

ки помимо текущего множества готовых к прокатке партий, также известно, какие еще партии будут произведены и станут готовы к прокате в ближайшем будущем. Эти еще не произведенные партии участвуют в формировании графиков прокатки наравне с уже готовыми партиями. Рассматривается ситуация, когда для каждой партии известен момент времени, начиная с которого партия готова к прокатке, а также известно какой промежуток времени требуется на прокатку данной партии. Эта информация позволяет формировать графики, в которых некоторые партии еще не готовы к прокатке на момент начала прокатки графика. При этом накладывается требование, чтобы к моменту, когда очередь дойдет до каждой следующей партии, она была готова к прокатке.

В настоящей работе будет предложена математическая модель для задачи формирования сбалансированного графика прокатки и алгоритм ее решения.

Глава 1

Формализация задачи построения сбалансированного графика прокатки

Обозначим все множество партий через P. Рассматривается конечный набор партий, то есть $|P| \in \mathbb{N}$.

1.1 Ограничения на порядок следования партий

Для начала формализуем ограничения на порядок следования партий друг за другом в графике прокатки.

Введем следующие обозначения.

- $w : P \to \mathbb{R}^+$ — ширина полос, получаемых из слябов партии, значение w на элементе $p \in P$ будем обозначать w_p.

  Множество всех ширин партий обозначим через $W = \{w_p \mid p \in P\}$.

- $\delta \in \mathbb{R}^+$ — максимальная величина допустимой разности значений ширины полос между любыми двумя соседними партиями.

- $t : P \to \mathbb{R}^+$ — толщина полос, получаемых из слябов партии, значение t на элементе $p \in P$ будем обозначать t_p.

- $r : \mathbb{R}^+ \to \mathbb{R}^+$ — 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, то есть $\forall t_1, t_2 \in \mathbb{R}^+ \; t_1 \leq t_2 \Rightarrow r(t_1) \leq r(t_2)$.

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»: $0 \leq w_p - w_q \leq \delta$.

Определенное таким образом ограничение представляет собой формализацию базовых условий предшествования партий, используемых на некоторых отечественных

предприятиях. В частности, на Магнитогорском металлургическом комбинате используется значение $\delta = 250\ mm$. Далее представлен пример функции r.

$$r(t) = \begin{cases} 0.8, & \text{если } t \in [1.29, 2]; \\ 1.5, & \text{если } t \in (2, 3]; \\ 2, & \text{если } t \in (3, 16]; \\ 4, & \text{если } t \in (16, 20]; \\ \text{не определена}, & \text{если } t \in [0, 1.29) \cup (20, +\infty). \end{cases}$$

Пример. Пусть имеются две партии p и q.

Если $w_p = w_q$, это означает, что для этих партий тривиально выполнено «ограничение для перехода по ширине».

Если $t_p = 2, t_q = 3$, то для этих партий «ограничение для перехода по толщине» не выполнено.

Если $t_p = 2.5, t_q = 2$, то для этих партий «ограничение для перехода по толщине» выполнено.

1.2 Сбалансированность графика прокатки

Обозначим множество видов продукции партий через F. Различных видов продукции конченое число, то есть $|F| \in \mathbb{N}$. Введем следующие обозначения.

- $f : P \to F$ — функция, сопоставляющая каждой партии соответствующий ей вид продукции.
- $l : P \to \mathbb{R}^+$ — 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 длина (в километрах), вес (в тоннах) партии, а также более сложные величины, учитывающие спрос, время изготовления, стоимость продукции и так далее.

  В данной работе $l(p)$ — суммарная длина полос партии p. Поскольку функция l названа «весовой», то далее будем говорить про вес партии, имея ввиду значение $l(p)$.
- $\mu : F \to \mathbb{R}^+$ — функция, задающая целевые значения суммарного веса партий каждого вида продукции в графике прокатки.
- $\Delta : F \to \mathbb{R}^+$ — функция, задающая абсолютную величину допустимого отклонения от целевого значения суммарного веса партий по каждому виду продукции в графике прокатки.
- $L_{min}, L_{max} \in \mathbb{R}^+$ — числа, задающие диапазон допустимых значений суммарного веса всех партий в графике прокатки.

1.3 Граф предшествования партий

В настоящей работе предложен подход, при котором ограничения на порядок следования партий формализуются при помощи графа предшествования партий. Пусть G раскрашенный вершинно-взвешенный ориентированный граф со множеством вершин P и с множеством ориентированных ребер $E \subseteq P \times P$ 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. Весом каждой вершины в G положим вес соответствующей партии $l(p)$, цвет каждой вершины в G соответствует виду продукции, к которому относится соответствующая партия.

В прошлой работе ([4]) был придуман удобный способ укладки графа предшествования партий на координатной плоскости, при котором абсциссой каждой вершины является значение функции t, а ординатой является значение функции w. На рисунке 1.1 изображен фрагмент графа предшествования партий.

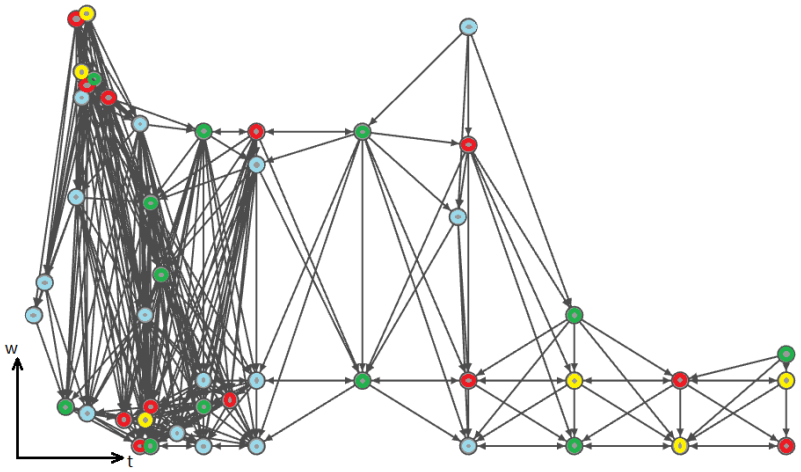

Рис. 1.1: Граф предшествования партий

На этом рисунке можно наглядно увидеть ограничение «для перехода по ширине». Оно заключается в том, что все ориентированные ребра в графе предшествования партий направлены либо сверху вниз, либо горизонтально, и в графе отсутствуют ориентированные ребра направленные снизу вверх. Ограничение «для перехода по толщине» также проиллюстрировано тем, что между собой смежны лишь те вершины, которые имеют достаточно близкие абсциссы.

Любому ребру 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 рассматриваемого множества партий P. Множество всех простых цепей в G обозначим через $\mathbb{B}(G)$.

Далее будем отождествлять партии прокатки и вершины графа предшествования партий.

1.4 Задача существования сбалансированной простой цепи

Пусть $P(M)$ — множество вершин произвольного подграфа графа G.

Для простой цепи $H \in \mathbb{B}(G)$ через $l(H)$ обозначим её вес.

$$l(H) \triangleq \sum_{p \in P(H)} l(p) \tag{1.1}$$

Пронумеруем виды продукции, пусть $F = \{f_1, f_2, \ldots, f_s\}$.

Сформулируем задачу поиска допустимого решения — найти в $\mathbb{B}(G)$ такую цепь H, что выполнены ограничения:

$$L_{min} \leq \sum_{p \in P(H)} l(p) \leq L_{max} \tag{1.2}$$

$$\forall k \in \overline{1,s} \quad \mu_k - \Delta_k \leq \sum_{p \in P(H), f(p)=f_k} l(p) \leq \mu_k + \Delta_k \tag{1.3}$$

1.5 Актуальность задачи существования сбалансированной простой цепи

В предыдущих работах ([3,4]) приводится обзор схожих по тематике работ и поясняется преимущество рассмотрения задачи построения графика прокатки именно как задачи поиска простой цепи в графе предшествования партий.

В данной работе, в отличие от предыдущих, выбран другой критерий оценки графика прокатки — сбалансированность. Существует несколько способов того, как можно формализовать практическую задачу построения графика прокатки сбалансированного по видам продукции. По существу это оптимизационная задача, в которой требуется приблизить распределение суммарного веса партий по каждому виду продукции максимально близко к целевым значениям.

Можно поставить, например, такую задачу с одним критерием оптимизации: найти в $\mathbb{B}(G)$ простую цепь H, где

$$\sum_{k=1}^{s} \left| \mu(f_k) - \frac{\sum\limits_{f(p_i)=f_k, p_i \in P(H)} l(p_i)}{\sum\limits_{p_i \in P(H)} l(p_i)} \right| \to \min, p_i \in P(H) \tag{1.4}$$

Такая задача имеет недостатком то, что в ней учитывается только «общая» сбалансированность решения. При этом нет гарантии, что в решении не будет «перекосов» и сильных отклонений по отдельным видам продукции, что является нежелательным с практической точки зрения. Можно рассматривать многокритериальную постановку оптимизационной задачи, однако такая постановка существенно сложнее в том числе и с вычислительной точки зрения. В данной работе выбрана именно задача существования сбалансированной простой цепи, поскольку она является хорошим инструментом для решения практической задачи, при этом постановка является достаточно простой, что позволяет решать ее за приемлемое с практической точки зрения время.

1.6 Дополнительные обозначения

Пусть p — произвольная вершина в G. Для дальнейшего изложения удобно ввести следующие обозначения.

- $In(p)$ — множество вершин q таких, что ребро $qp \in E$,
  $|In(p)|$ — входящая валентность вершины p.
- $In^+(p) = \{q \in In(p) \mid w_q > w_p\}$.
- $Out(p)$ — множество вершин q таких, что ребро $pq \in E$,
  $|Out(p)|$ — исходящая валентность вершины p.

Также рассмотрим множество всевозможных ширин вершин $W = \{w_p \mid p \in P\}$. Для $w \in W$ обозначим через $P_w = \{p \in P \mid w_p = w\}$ множество вершин, имеющих данное значение w. G_w — граф, индуцированный множеством P_w.

Глава 2

Метод решения и структура работы

Инструментом для решения практической задачи построения сбалансированного графика прокатки является задача существования сбалансированной простой цепи в графе предшествования партий. В предыдущих работах ([3, 4]) рассмотрена оптимизационная постановка задачи поиска простой цепи в графе предшествования партий, в которой критерием оптимальности выбран суммарный вес вошедших в простую цепь вершин. Эту задачу удалось решить, предложив конструктивный полиномиальный алгоритм построения простой цепи с временной сложностью $o(n^3)$, где n — число вершин в графе предшествования партий. Для решения задачи построения сбалансированной простой цепи создание подобного полиномиального алгоритма оказалось гораздо более сложным, поскольку помимо «маршрутных» ограничений на порядок следования вершин в задаче присутствуют «объемные» ограничения на суммарный вес вершин по каждому виду продукции и ограничение на суммарный вес всех вершин искомой простой цепи. По этой причине было решено прибегнуть к аппарату математического программирования, поскольку он позволяет единообразно рассматривать как «маршрутные», так и «объемные» ограничения. Этот подход ценен еще и по той причине, что позволяет расширять рассматриваемую постановку, добавляя другие ограничения, которые существуют в реальной практической задаче. «Объемные» ограничения не исчерпывают все виды ограничений, которые можно рассматривать при моделировании практической задачи. Существуют еще, например, «временные» ограничения, связанные со временем готовности партий к прокатке и длительностью прокатки каждой партии. Используемый в данной работе подход позволяет учесть их наравне с остальными ограничениями, тогда как конструктивный полиномиальный алгоритм скорее всего стал бы непригоден для решения новой задачи.

Итак, в данной работе решение задачи существования сбалансированной простой цепи сводится к решению серии задач целочисленного программирования (которые, в свою очередь, сводятся к задачам линейного программирования, и результат проверяется на целочисленность). Формализация «маршрутных» ограничений на порядок следования вершин в искомой простой цепи базируется на исследовании структуры графа предшествования партий ([2, 3]). Благодаря этому удается существенно снизить размерность получаемых задач линейного программирования по сравнению с прямым использовании классических техник для решения маршрутных задач (например, [7]).

2.1 Локальные задачи существования сбалансированной простой цепи

Решение задачи существования сбалансированной простой цепи начнем с рассмотрения некоторых приближенных и упрощенных вариантов, затем рассмотрим исходную задачу. Рассматриваемые приближенные варианты задачи ценны, поскольку для их решения требуется существенно меньше времени, в некоторых ситуациях это может быть полезно. Кроме того, начиная изложение с приближенных задач, можно на более простых примерах продемонстрировать идеи, которые используются и для решения исходной задачи.

Для ориентированного графа предшествования партий G рассмотрим его подграфы G_w, где $w \in W$. Для каждого фиксированного $w \in W$ подграф G_w порожден вершинами p исходного графа G, имеющих значение $w_p = w$.

Для любых двух принадлежащих G_w вершин p и q ограничение «для перехода по ширине» тривиально выполняется, поэтому наличие ребра pq в графе предшествования партий определяется лишь ограничением «для перехода по толщине». Поскольку это ограничение является симметричным, то из существования ребра pq следует существование ребра qp, поэтому подграф G_w можно рассматривать как обыкновенный.

Итак, граф G состоит из некоторого количества обыкновенных подграфов. Если вершины графа G расположить на плоскости в соответствии со значением w и t (см. раздел 1.3 Граф предшествования партий), то каждый обыкновенный подграф G_w будет состоять из вершин, имеющих одинаковую ординату w. При этом в соответствии с ограничением «для перехода по ширине» ребра между вершинами различных подграфов могут идти только сверху вниз, и не могут идти снизу вверх. Поэтому любая достаточно длинная простая цепь в графе G будет проходить через вершины подграфов G_w сверху вниз (рисунок 2.1).

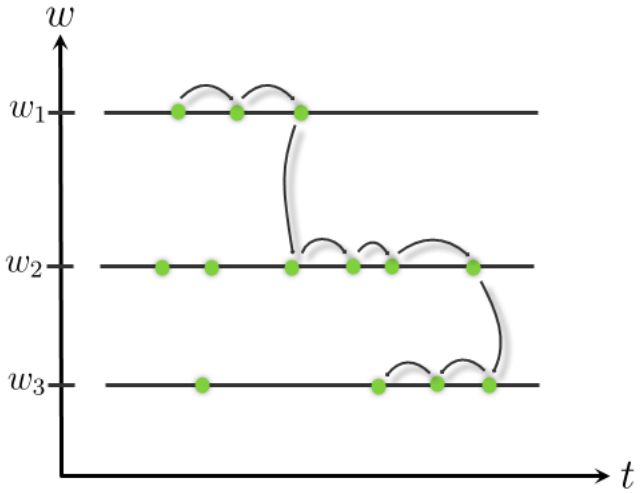

Рис. 2.1: Структура достаточно длинной простой цепи в G

Решение задачи существования сбалансированной простой цепи в G проходит через какую-то последовательность подграфов $\{G_{w_i}\}$. Идея, на которой основано создание приближенных задач, заключается в том, что можно заранее выбрать и зафиксировать некоторую последовательность подграфов, и потребовать, чтобы искомая простая цепь в обязательном порядке проходила через выбранные подграфы. Это позволяет решать задачу существенно меньшей размерности, чем исходная задача существования, поскольку фиксированная последовательность подграфов, через

которую должна пройти искомая простая цепь, позволяет исключить многие «маршрутные» ограничения на порядок следования вершин. Задачу существования сбалансированной простой цепи с выбранной последовательностью подграфов назовем локальной задачей существования. Подробное описание локальной задачи существования будет изложено в следующем разделе.

Открытый вопрос, который сопутствует локальной задаче существования это то, каким образом можно эффективно выбирать последовательность подграфов. Далее предложено некоторое эвристическое соображение, используемое в этой работе.

Разработанный в прошлых работах ([3, 4]) алгоритм построения простой цепи максимального веса в графе G перебирает подграфы G_w в порядке от большего значения w к меньшему и строит максимальную по весу цепь H_{max}. Эта цепь отлично подходит в качестве некого начального приближения к решению задачи поиска сбалансированной простой цепи в G, поскольку обычно вес H_{max} заведомо больше, чем требуется для сбалансированной цепи (а если он меньше, то значит, поставленная задача существования сбалансированной цепи несовместна), и кроме того в H_{max} обычно заведомо больше вершин, относящихся к каждому виду продукции, чем того требует условие сбалансированности.

Далее дадим неформальное описание двух видов локальных задач существования.

2.1.1 Задача с фиксированными входами и выходами

Пусть H — простая цепь в графе G. Зафиксируем последовательность подграфов $\{G_{w_i}\}$, через которые проходит простая цепь H в порядке уменьшения значения w_i. Пронумеруем вершины цепи $H = (q_1, q_2, \ldots, q_n)$. Ввиду того, что в графе G ребра, соединяющие различные подграфы G_w направлены только от подграфа с большим значением w к подграфу с меньшим значением w, вершины цепи H идут через подграфы последовательности $\{G_{w_i}\}$ по порядку. Для каждого подграфа зафиксируем вершину, являющуюся первой вершиной цепи H, принадлежащей данному подграфу, и назовем эту вершину входом в подграф для цепи H. Зафиксируем также вершину, являющуюся последней вершиной цепи H, принадлежащей данному подграфу, и назовем эту вершину выходом из подграфа для цепи H. На рисунке 2.2 подписаны входы и выходы в подграфы G_w (s_i и e_i соответственно).

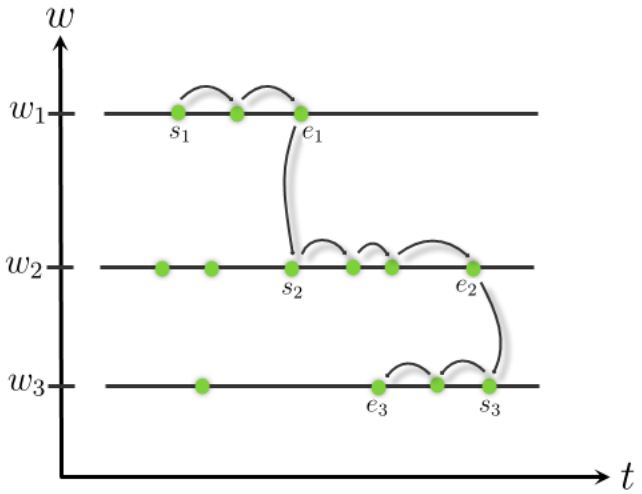

Рис. 2.2: Входы и выходы подграфов G_w

Потребуем, чтобы входы и выходы каждого подграфа обязательно входили в

простую цепь, являющуюся решением локальной задачи существования сбалансированной простой цепи. Это требование позволяет обеспечить связность частей цепи-решения, относящихся к различным подграфам $\{G_{w_i}\}$. Благодаря этому, при формализации локальной задачи существования в виде задачи математического программирования для обеспечения связности всей цепи в целом потребуется описать только «маршрутные» ограничения на порядок следования вершин в рамках отдельных подграфов.

Заметим, что входом и выходом в подграфе может быть одна и та же вершина p. В этом случае часть простой цепи-решения, относящаяся к данному подграфу будет состоять из единственной вершины p.

Заметим, что для того, чтобы поставить локальную задачу существования сбалансированной простой цепи с фиксированными входами и выходами не обязательно иметь какую-либо простую цепь H. Можно выбирать последовательность подграфов $\{G_{w_i}\}$ и входы и выходы в них произвольным образом. Однако на практике решению этой задачи предшествует выбор цепи, являющейся начальным приближением.

Итак, локальная задача существования сбалансированной простой цепи с фиксированными входами и выходами формулируется следующим образом.

Пусть заданы

- граф предшествования партий G,
- последовательность подграфов $\{G_{w_i}\}$,
- пары вершин $s_i,\ e_i$ — вход и выход для каждого подграфа,
- ограничения предшествования вершин в рамках отдельных подграфов,
- ограничения на суммарный вес вершин в цепи и на суммарный вес по каждому виду продукции.

Существует ли сбалансированная простая цепь при заданных ограничениях?

2.1.2 Задача со свободными входами и выходами

Зафиксируем последовательность подграфов $\{G_{w_i}\}$ в порядке уменьшения значения w_i. Цепь, являющаяся решением локальной задачи существования должна в обязательном порядке пройти через каждый из выбранных подграфов. При этом не будем фиксировать, какие именно вершины должны быть входами или выходами. Наличие фиксированной последовательности подграфов $\{G_{w_i}\}$ позволяет при формализации локальной задачи существования описать только «маршрутные» ограничения на порядок следования вершин в рамках отдельных подграфов и ограничения на порядок следования вершин между соседними подграфами в последовательности.

Пусть заданы

- граф предшествования партий G,
- последовательность подграфов $\{G_{w_i}\}$,
- ограничения предшествования вершин в рамках отдельных подграфов,
- ограничения предшествования для связи вершин между соседними подграфами,
- ограничения на суммарный вес вершин в цепи и суммарный вес по каждому виду продукции.

Существует ли сбалансированная простая цепь при заданных ограничениях?

2.2 Исходная задача существования сбалансированной простой цепи

Для того, чтобы формализовать исходную задачу существования простой цепи во всем графе G, рассматривается последовательность всех подграфов $\{G_{w_i}\}$ в порядке уменьшения значения w_i, при этом не накладывается требование на то, чтобы цепь-решение в обязательном порядке проходила через каждый подграф последовательности, и не накладывается требование на то, какие именно вершины должны быть входами и выходами в подграфах. В этом случае при формализации задачи существования сбалансированной простой цепи в виде задачи математического программирования требуется описать как ограничения на порядок следования вершин в рамках отдельных подграфов, так и ограничения на порядок следования вершин между любыми подграфами, а не только соседними, как это было в локальной задачи существования сбалансированной простой цепи со свободными входами и выходами.

Пусть заданы

- граф предшествования партий G,
- последовательность подграфов $\{G_{w_i}\}$,
- ограничения предшествования для вершин между всеми подграфами,
- ограничения предшествования вершин в рамках отдельных подграфов,
- ограничения на суммарный вес вершин в цепи и суммарный вес по каждому виду продукции.

Существует ли сбалансированная простая цепь при заданных ограничениях?

Глава 3

Формализация в виде задачи математического программирования

Пусть $G_{w_1}, G_{w_2}, \ldots, G_{w_m}$ — набор подграфов графа предшествования партий G с множеством вершин P.

Пронумеруем вершины подграфов в порядке увеличения значения функции t. Пусть p_{ij} — j-я вершина в i-м подграфе. Каждой вершине p_{ij} поставим в соответствие булевскую переменную b_{ij} по следующему правилу.

$$b_{ij} = \begin{cases} 1, & \text{если } p_{ij} \text{ входит в цепь-решение;} \\ 0, & \text{если } p_{ij} \text{ не входит в цепь-решение.} \end{cases}$$

Обозначим

- $n_i = |P(G_{w_i})|$ — количество вершин в подграфе G_{w_i},
- $l_{ij} = l(p_{ij})$ — вес вершины p_{ij},
- L_{min}, L_{max} — заданные минимальное и максимальное значения суммарного веса вершин в цепи-решении,
- $f_1, f_2, \ldots, f_s$ — набор видов продукции,
- $\mu_i = \mu(f_i) = (\underline{\mu}_i, \overline{\mu}_i)$ — распределение видов продукции, заданное допустимым диапазоном значения суммарного веса вершин, имеющих f_i вид продукции,
- $f_{ij} = f(p_{ij})$ — вид продукции, соответствующий вершине p_{ij}.

Ограничение на суммарный вес цепи

$$L_{min} \leq \sum_{i=1}^{m} \sum_{j=1}^{l_i} b_{ij} l_{ij} \leq L_{max} \tag{3.1}$$

Ограничение на суммарный вес по видам продукции

$$\forall k \in \overline{1, s} \quad \underline{\mu}_k \leq \sum_{f_{ij}=f_k} b_{ij} l_{ij} \leq \overline{\mu}_k \tag{3.2}$$

Указанные «объемные» ограничения являются базовыми, они присутствуют в любой постановке задачи существования сбалансированной простой цепи. Способы формализации «маршрутных» ограничений зависят от выбранной постановки. Прежде чем перейти к их описанию введем некоторые вспомогательные обозначения.

Дополнительные обозначения

Пусть p, q и r — вершины подграфа G_{w_i}, то есть $p, q \in P_{w_i}$. Введем вспомогательные понятия.

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

Обозначим множество вершин подграфа G_{w_i}, находящихся правее вершины p через $P_i^{>,p}$, зафиксируем множество индексов этих вершин и обозначим его через $J_i^{>,p}$.

- $P_i^{>,p} = \{q \in P_{w_i} \mid q > p\}$
- $J_i^{>,p} = \{j \mid p_{ij} \in P_i^{>,p}\}$

Обозначим множество вершин подграфа G_{w_i}, находящихся правее вершины p и смежных с ней через $\overset{o}{P}{}_i^{>,p}$, зафиксируем множество индексов этих вершин и обозначим его через $\overset{o}{J}{}_i^{>,p}$.

- $\overset{o}{P}{}_i^{>,p} = \{q \in P_{w_i} \mid q > p, \ \ |t_q - t_p| \leq min\{r(t_q), r(t_p)\}\}$
- $\overset{o}{J}{}_i^{>,p} = \{j \mid p_{ij} \in \overset{o}{P}{}_i^{>,p}\}$

Аналогичные обозначения можно ввести для множества вершин подграфа G_{w_i}, находящихся левее вершины p, не правее вершины p и не левее вершины p.

Множество вершин подграфа G_{w_i}, находящихся между вершинами p и q (предположим, $p < q$) обозначим через $P_i^{p,\geq,q}$. Соответственно множество индексов этих вершин $J_i^{p,\geq,q}$.

Замечание 1. В подграфе G_{w_i} 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.

Аналогично можно показать, что если две вершины p и q связаны ребром, то находящиеся между ними вершины p' и q' также связаны ребром.

3.1 Локальные задачи существования сбалансированной простой цепи

3.1.1 Задача с фиксированными входами и выходами

- G_{w_i} — подграф графа G, s_i, e_i — заданная пара вершин, вход и выход для подграфа G_{w_i}. Допускается, что вход и выход это одна и та же вершина.

Пусть $p \in P$, соответствующую вершине p переменную обозначим b^p.

Тогда b^{s_i}, b^{e_i} это переменные, соответствующие входу и выходу. Потребуем, чтобы вход и выход были включены в цепь-решение.

$$b^{s_i} = 1, \qquad b^{e_i} = 1 \tag{3.3}$$

- Пусть $p_{ij_0} \in G_{w_i}$ — отличная от входа и выхода вершина, то есть $p_{ij_0} \neq s_i$ и $p_{ij_0} \neq e_i$.

Замечание 2. Поскольку в графе G_{w_i} любые две вершины p и q связаны ребром pq тогда и только, когда они связаны ребром qp, то для любой простой цепи цепь с обратным порядком вершин также существует. Следовательно, не ограничивая общности, далее можно считать, что вход всегда не правее выхода, поскольку в противном случае вход и выход можно поменять местами.

Если $s_i < p_{ij_0} < e_i$, то

$$b_{ij_0} \leq \sum_{j' \in \overset{o}{J}{}_i^{>,p}} b_{ij'}, \qquad b_{ij_0} \leq \sum_{j' \in \overset{o}{J}{}_i^{<,p}} b_{ij'} \tag{3.4}$$

Это означает, что вершина p_{ij_0}, находящаяся между входом и выходом может быть включена в цепь-решение, если среди смежных с ней вершин в цепь-решение включена хотя бы одна вершина, находящаяся правее, и хотя бы одна вершина, находящаяся левее p_{ij_0}.

Если $p_{ij_0} < s_i$, то

$$2b_{ij_0} \leq \sum_{j' \in \overset{o}{J}{}_i^{>,p}} b_{ij'} \tag{3.5}$$

Это означает, что вершина p_{ij_0}, находящаяся левее входа может быть включена в цепь-решение, если среди смежных с ней вершин в цепь-решение включены хотя бы две вершины, находящиеся правее p_{ij_0}.

Если $e_i < p_{ij_0}$, то

$$2b_{ij_0} \leq \sum_{j' \in \overset{o}{J}{}_i^{<,p}} b_{ij'} \tag{3.6}$$

Это означает, что вершина p_{ij_0}, находящаяся правее выхода может быть включена в цепь-решение, если среди смежных с ней вершин в цепь-решение включены хотя бы две вершины, находящиеся левее p_{ij_0}.

Если $t_{p_{ij_0}} = t_{s_i}$ или $t_{p_{ij_0}} = t_{e_i}$, то ограничение на возможность включения вершины p_{ij_0} в цепь-решение не накладывается.

Локальная задача существования состоит в том, чтобы найти решение следующей системы уравнений и неравенств, либо установить, что его не существует.

$$
\begin{cases}
L_{min} \le \sum\limits_{i=1}^{m}\sum\limits_{j=1}^{n_i} b_{ij}l_{ij} \le L_{max};\\
\forall k \in \overline{1,s} \quad \underline{\mu_k} \le \sum\limits_{f_{ij}=f_k} b_{ij}l_{ij} \le \overline{\mu_k};\\
\forall i \in \overline{1,m}:\\
\quad \begin{cases}
b^{s_i} = 1,\ b^{e_i} = 1;\\
\forall j_0 \in \overline{1,n_i}(p_{ij_0} \notin \{s_i, e_i\}):\\
b_{ij_0} \le \sum\limits_{j' \in \overset{o}{J_i}^{>,p}} b_{ij'}, \quad b_{ij_0} \le \sum\limits_{j' \in \overset{o}{J_i}^{<,p}} b_{ij'}, \quad \text{если } s_i < p_{ij_0} < e_i;\\
2b_{ij_0} \le \sum\limits_{j' \in \overset{o}{J_i}^{>,p}} b_{ij'}, \quad \text{если } p_{ij_0} < s_i;\\
2b_{ij_0} \le \sum\limits_{j' \in \overset{o}{J_i}^{<,p}} b_{ij'}, \quad \text{если } e_i < p_{ij_0}.
\end{cases}
\end{cases} \qquad (*)
$$

Решение вложенной подсистемы неравенств $(*)$ означает, что из вершин, вошедших в решение, принадлежащих одному подграфу вместе с входом и выходом, возможно построить простую цепь в G с началом во входной вершине и окончанием в выходной вершине. Возможность собрать все вершины в решении локальной задачи в одну простую цепь гарантируется в том случае, если входы и выходы в следующих друг за другом подграфах выбраны согласованно.

Теорема 1. *В подграфе G_w существует простая (s,e)-цепь тогда и только тогда, когда существует решение системы $(*)$.*

Доказательство. $\Rightarrow$ Пусть в G_{w_i} существует простая (s_i, e_i)-цепь H, покажем, что тогда существует решение системы $(*)$. Рассмотрим набор переменных $\{b_{ij}\}$, соответствующих вершинам G_{w_i}. Присвоим переменным значения в соответствии с цепью H. Если вершина p_{ij} входит в цепь H, то присвоим $b_{ij} = 1$, в противном случае присвоим $b_{ij} = 0$.

Рассмотрим $j_0 \in \overline{1,n_i}$.

Если $p_{ij_0} \notin P(H)$, то все неравенства, соответствующие переменной b_{ij_0} выполнены, поскольку в их левой части стоит 0, а в правой части стоит сумма неотрицательных чисел.

Пусть $p_{ij_0} \in P(H)$. Это значит, что $b_{ij_0} = 1$.

- Если $p_{ij_0} < s_i$, то рассмотрим множества $P_i^{\le,p_{ij_0}}$ и $P_i^{>,p_{ij_0}}$. Поскольку простая (s,e)-цепь H посещает вершину $p_{ij_0} \in P_i^{\le,p_{ij_0}}$ и заканчивается в вершине $e_i \in P_i^{>,p_{ij_0}}$, то среди ребер цепи H найдется пара ребер, соединяющих эти множества. То есть $\exists u_1, u_2 \in P_i^{\le,p_{ij_0}} \cup P(H)$ и $\exists v_1, v_2 \in P_i^{>,p_{ij_0}} \cup P(H)$ ($v_1 \ne v_2$, так как H — простая), что $u_1v_1 \in E$ и $u_2v_2 \in E$.

  Поскольку $u_1 \le p_{ij_0} < v_1$ и $u_2 \le p_{ij_0} < v_2$, то по замечанию 1 $p_{ij_0}v_1 \in E$ и $p_{ij_0}v_2 \in E$. Заметим, что $v_1, v_2 \in P(H)$, значит $b^{v_1} = 1$ и $b^{v_2} = 1$. Таким образом, неравенства, соответствующие переменной b_{ij_0} выполнены, поскольку имеют вид

  $$2b_{ij_0} \le b^{v_1} + b^{v_2} + K,$$

  где K — сумма неотрицательных чисел.

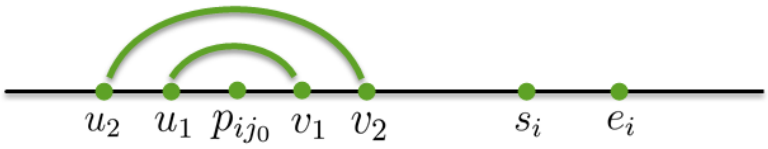

Рис. 3.1: Пример взаимного расположения вершин

- Случай $e_i < p_{ij_0}$ рассматривается аналогично предыдущему с использованием множеств $P_i^{<,p_{ij_0}}$ и $P_i^{\geq,p_{ij_0}}$.

- Если $s_i < p_{ij_0} < e_i$, то снова рассмотрим множества $P_i^{\leq,p_{ij_0}}$ и $P_i^{>,p_{ij_0}}$. Поскольку простая (s,e)-цепь H посещает вершину $p_{ij_0} \in P_i^{\leq,p_{ij_0}}$ и заканчивается в вершине $e_i \in P_i^{>,p_{ij_0}}$, то среди ребер цепи H найдется ребро, соединяющее эти множества. То есть $\exists u \in P_i^{\leq,p_{ij_0}} \cup P(H)$ и $\exists v \in P_i^{>,p_{ij_0}} \cup P(H)$, что $uv \in E$. Поскольку $u \leq p_{ij_0} < v$, то по замечанию 1 $up_{ij_0} \in E$ и $p_{ij_0}v \in E$. Заметим, что $u, v \in P(H)$, значит $b^u = 1$ и $b^v = 1$. Таким образом, неравенства, соответствующие переменной b_{ij_0} выполнены, поскольку имеют один из двух видов

  $$b_{ij_0} \leq b^u + K_1, \qquad b_{ij_0} \leq b^v + K_2,$$

  где K_1 и K_2 — суммы неотрицательных чисел.

- В случае $t_{p_{ij_0}} = t_{s_i}$ или $t_{p_{ij_0}} = t_{e_i}$ переменной b_{ij_0} не соответствует никаких неравенств.

Таким образом, набор значений переменных b_{ij_0}, выбранный в соответствии с цепью H, удовлетворяет всем неравенствам и является решением системы $(*)$.

$\Leftarrow$ Пусть существует решение системы $(*)$, покажем, что в подграфе G_{w_i} существует простая (s_i, e_i)-цепь. Выберем множество вершин подграфа G_{w_i}, которым соответствуют переменные, имеющие значение 1, обозначим его M. Множество M не пусто, поскольку существует хотя бы одна ненулевая переменная $b^{s_i} = 1$ или $b^{e_i} = 1$, возможно это одна и та же переменная, если $s_i = e_i$.

В случае, если $s_i = e_i$, то простая (s_i, e_i)-цепь состоит из одной вершины s_i.

Рассмотрим случай, когда $s_i \neq e_i$.

Упорядочим вершины множества M в порядке увеличения t. Поскольку множество M составляют вершины, соответствующие решению системы $(*)$, то каждая вершина множества M смежна по крайней мере еще с двумя соседними с ней вершинами (за исключением, быть может, вершин s_i и e_i). В этом случае простую (s_i, e_i)-цепь H можно построить используя частный алгоритм построения максимальной по включению вершин простой цепи, применив его к множеству вершин M (подробное описание частного алгоритма представлено в работе [3]). $\square$

3.1.2 Задача со свободными входами и выходами

В рамках этой задачи также рассматривается фиксированная последовательность подграфов $\{G_{w_i}\}$. Отличие от предыдущей задачи в том, что теперь не зафиксировано то, какие вершины должны быть входами и выходами, а значит, необходимо ввести ограничения, описывающие связность цепи-решения между соседними подграфами.

Добавляется два набора переменных. Каждой вершине p_{ij} соответствуют

$$s_{ij} = \begin{cases} 1, & \text{если } p_{ij} \text{ является входом в } G_{w_i}; \\ 0, & \text{если } p_{ij} \text{ не является входом в } G_{w_i}; \end{cases}$$

$$e_{ij} = \begin{cases} 1, & \text{если } p_{ij} \text{ является выходом из } G_{w_i}; \\ 0, & \text{если } p_{ij} \text{ не является выходом из } G_{w_i}. \end{cases}$$

В задаче со свободными входами и выходами рассматривается следующий ряд ограничений.

В каждом подграфе должен быть ровно один вход и ровно один выход, то есть

$$\forall i \in \overline{1,m} \sum_{j=1}^{n_i} s_{ij} = 1, \qquad \forall i \in \overline{1,m} \sum_{j=1}^{n_i} e_{ij} = 1 \tag{3.7}$$

Вершины, являющиеся входами или выходами должны быть включены в цепь-решение, то есть

$$\forall i \in \overline{1,m}\, \forall j \in \overline{1,n_i} \quad b_{ij} \geq s_{ij}, \qquad \forall i \in \overline{1,m}\, \forall j \in \overline{1,n_i} \quad b_{ij} \geq e_{ij} \tag{3.8}$$

Рассмотрим подграф G_{w_i}. Предположим, что существует вершина x, имеющая значения $t = t_x$ и $w = w_i$. Обозначим множество вершин подграфа G_{w_i} смежных с вершиной x через $P_i(t_x)$.

$$P_i(x) = \{p \in P_{w_i} \mid |t_p - t_x| \leq min\{r(t_p), r(t_x)\}\}$$

Зафиксируем множество индексов этих вершин.

$$J_i(x) = \{j \mid p_{kj} \in P_i(t_x)\}$$

Смысл следующего ограничения в том, что для каждой вершины-входа в подграф, среди вершин-предков этого входа, принадлежащих предыдущему подграфу, должна существовать вершина, являющийся выходом из предыдущего подграфа.

$$\forall i = 2, ..., m\, \forall j_0 \in \overline{1,n_i} \quad s_{ij_0} \leq \sum_{j \in P_{i-1}(p_{ij_0})} e_{i-1j} \tag{3.9}$$

Выполнение указанного ограничения для соседних подграфов означает, что входы и выходы в них выбраны согласованно, что гарантирует возможность собрать фрагменты цепи-решения соответствующие отдельным подграфам в одну простую цепь.

По сравнению с задачей с фиксированными входами и выходами ограничения предшествования в рамках отдельных подграфов (система $(*)$) заменяются следующими.

$$\forall i \in \overline{1,m}\, \forall j_0 \in \overline{1,n_i} : \begin{cases} 2b_{ij0} \leq \sum\limits_{j' \in \overset{o}{J_i}^{>,p_{ij_0}}} b_{ij'} + (2 - \sum\limits_{j \in J_i^{>,p_{ij_0}}} (s_{ij} + e_{ij})) \\ 2b_{ij0} \leq \sum\limits_{j' \in \overset{o}{J_i}^{<,p_{ij_0}}} b_{ij'} + (2 - \sum\limits_{j \in J_i^{<,p_{ij_0}}} (s_{ij} + e_{ij})) \end{cases} \tag{3.10}$$

Легко видеть, что при каждом возможном взаимном расположении вершин s_i, e_i и p_{ij_0} система 3.10 накладывает такие же требования, что и система $(*)$ из задачи со свободными входами и выходами.

В самом деле, если вершина p_{ij_0} расположена левее, чем вход, то первое неравенство превращается в $2b_{ij0} \leq \sum\limits_{j' \in \overset{o}{J_i}^{>,p_{ij_0}}} b_{ij'}$, которое в точности есть неравенство из системы $(*)$ для случая $p_{ij_0} < s_i$.

Второе неравенство превращается в $2b_{ij0} \leq \sum\limits_{j' \in \overset{o}{J_i}^{<,p_{ij_0}}} b_{ij'} + 2$ и выполняется тождественно.

Если вершина p_{ij_0} расположена правее, чем выход, то второе неравенство превращается в $2b_{ij0} \leq \sum\limits_{j' \in \overset{o}{J_i}^{<,p_{ij_0}}} b_{ij'} + 2$, которое в точности есть неравенство из системы $(*)$ для случая $e_i < p_{ij_0}$.

Первое неравенство превращается в $2b_{ij0} \leq \sum\limits_{j' \in \overset{o}{J_i}^{>,p_{ij_0}}} b_{ij'} + 2$ и выполяется тождественно.

Если же вершина p_{ij_0} расположена между входом и выходом, то получается следующая пара неравенств.

$$2b_{ij0} \leq \sum_{j' \in \overset{o}{J_i}^{>,p_{ij_0}}} b_{ij'} + 1, \qquad 2b_{ij0} \leq \sum_{j' \in \overset{o}{J_i}^{<,p_{ij_0}}} b_{ij'} + 1 \tag{3.11}$$

Эта пара неравенств означает, что вершина p_{ij_0} может быть включена в цепь-решение, если в цепь-решение включена хотя бы одна смежная с ней вершина, находящаяся справа, и хотя бы одна смежная с ней вершина находящаяся слева. Это требование в точности соответствует неравенству системы $(*)$ для случая $s_i < p_{ij_0} < e_i$.

Таким образом справедливо следующее утверждение.

Теорема 2. *В подграфе G_{w_i} существует простая (s_i, e_i)-цепь тогда и только тогда, когда существует решение системы*

$$\begin{cases} \forall i \in \overline{1,m} \sum\limits_{j=1}^{n_i} s_{ij} = 1, \\ \forall i \in \overline{1,m} \sum\limits_{j=1}^{n_i} e_{ij} = 1, \\ \forall i \in \overline{1,m}\, \forall j \in \overline{1,n_i} \quad b_{ij} \geq s_{ij}, \\ \forall i \in \overline{1,m}\, \forall j \in \overline{1,n_i} \quad b_{ij} \geq e_{ij}, \\ \forall i \in \overline{1,m}\, \forall j_0 \in \overline{1,n_i}: \\ \begin{cases} 2b_{ij0} \leq \sum\limits_{j' \in \overset{o}{J_i}^{>,p_{ij_0}}} b_{ij'} + (2 - \sum\limits_{j \in J_i^{>,p_{ij_0}}} (s_{ij} + e_{ij})) \\ 2b_{ij0} \leq \sum\limits_{j' \in \overset{o}{J_i}^{<,p_{ij_0}}} b_{ij'} + (2 - \sum\limits_{j \in J_i^{<,p_{ij_0}}} (s_{ij} + e_{ij})) \end{cases} \end{cases} \tag{3.12}$$

3.2 Исходная задача существования сбалансированной простой цепи

В предыдущей задаче была задана фиксированная последовательность подграфов, в том смысле что для построения цепи-решения необходимо было, чтобы в нее были включены вершины из каждого подграфа последовательности. Теперь же сформулируем ограничения для связности подграфов таким образом, чтобы избежать этой необходимости. Это позволит рассматривать произвольные последовательности подграфов, то есть, перейти от локальных задач существования к исходной.

Для каждого подграфа G_{w_i} графа G поставим в соответствие переменную

$$q_i = \begin{cases} 1, & \text{если подграф } G_{w_i} \text{ включен в цепь-решение (хотя бы 1 узел из } G_{w_i}); \\ 0, & \text{иначе.} \end{cases}$$

В исходной задаче существования сбалансированной простой цепи рассматривается следующий ряд ограничений.

Ограничения предшествования вершин в рамках подграфов такие же как и в локальной задаче существования со свободными входами и выходами.

$$\forall i \in \overline{1,m}\ \forall j_0 \in \overline{1,n_i} : \begin{cases} 2b_{ij0} \le \sum\limits_{j' \in \overset{o}{J_i}^{>,p_{ij_0}}} b_{ij'} + (2 - \sum\limits_{j \in J_i^{>,p_{ij_0}}} (s_{ij} + e_{ij})) \\ 2b_{ij0} \le \sum\limits_{j' \in \overset{o}{J_i}^{<,p_{ij_0}}} b_{ij'} + (2 - \sum\limits_{j \in J_i^{<,p_{ij_0}}} (s_{ij} + e_{ij})) \end{cases} \tag{3.13}$$

Далее опишем ограничения, которые обеспечат возможность связать части цепи-решения соответствующие отдельным подграфам в одну простую цепь.

Если подграф G_{w_i} не включен в цепь-решение, то ни один его узел не представлен в решении.

$$\forall i \in \overline{1,m}\ \forall j \in \overline{1,n_i} \quad b_{ij} \le q_i \tag{3.14}$$

Если хотя бы один узел подграфа G_{w_i} включен в цепь-решение, то подграф включен в решение.

$$\forall i \in \overline{1,m} \quad q_i \le \sum_{j=1}^{n_i} b_{ij} \tag{3.15}$$

Если в подграфе есть вход и есть выход, то подграф включен в решение.

$$\forall i \in \overline{1,m} \quad q_i = \sum_{j=1}^{n_i} e_{ij}, \qquad \forall i \in \overline{1,m} \quad q_i = \sum_{j=1}^{n_i} s_{ij} \tag{3.16}$$

Входы и выходы в подграф должны быть включены в цепь-решение.

$$\forall i \in \overline{1,m}\ \forall j \in \overline{1,n_i} \quad b_{ij} \ge s_{ij}, \qquad \forall i \in \overline{1,m}\ \forall j \in \overline{1,n_i} \quad b_{ij} \ge e_{ij} \tag{3.17}$$

Осталось только предложить ограничение, которое позволит обеспечить согласованность входов и выходов в различных подграфах.

Среди вершин подграфа G_{w_k} зафиксируем множество вершин, являющихся непосредственными предшественниками вершины q, принадлежащей подграфу, отличному от G_{w_k}. Обозначим его $In_{P_k}(q) = \{p_k \in P_{w_k} \mid p_k q \in E\}$. Зафиксируем также множество индексов этих вершин $J_k^{in}(q) = \{j \mid p_{kj} \in In_{P_k}(q)\}$.

Итак, для того, чтобы обеспечить согласованность входов и входов достаточно, чтобы выполнялось следующее условие. Вершина включенного в решение подграфа может являться входом в этот подграф, если среди вершин-предшественников этой вершины, относящихся к предыдущему включенному в решение подграфу, какая-то из вершин является выходом.

$$\forall i \in \overline{1,m}\ \forall k < i \quad s_{ij} \le \sum_{k<l<i} q_l + \sum_{z \in J_k^{in}(p_{ij})} e_{kz} + (1 - q_k) \tag{3.18}$$

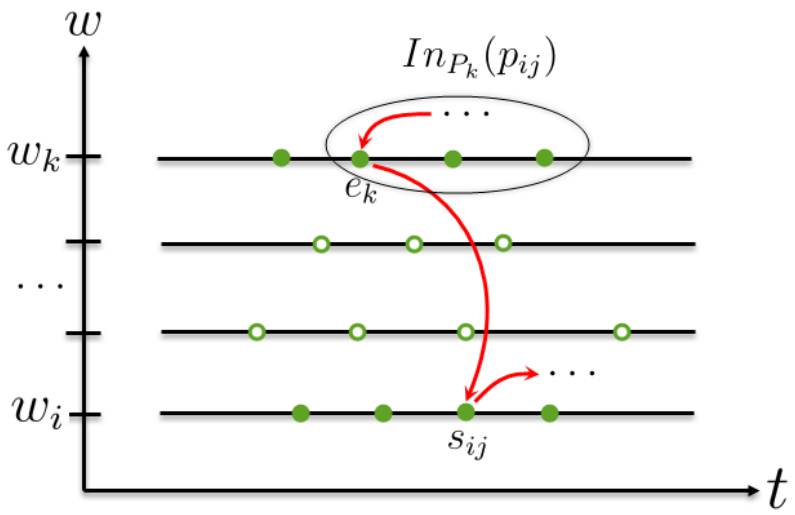

Рис. 3.2: Связь произвольных подграфов G_{w_i} и G_{w_k}

Покажем, что для любого подграфа G_{w_i} и для любого предшествующего ему подграфа G_{w_k} ограничение 3.18 обеспечивает согласованность входов и выходов в этих подграфах.

Если подграф G_{w_i} не включен в цепь-решение, или если вершина p_{ij} не является входом в G_{w_i}, то ограничение 3.18 тривиально выполнено, поскольку 0 не превосходит суммы неотрицательных чисел.

Пусть далее подграф G_{w_i} включен в цепь-решение, и вершина p_{ij} является входом в G_{w_i}. Если при этом G_{w_k} не включен в цепь-решение, то $q_k = 0$, последнее слагаемое правой части ограничения 3.18 равно единице, все остальные слагаемые неотрицательные, и следовательно, ограничение 3.18 выполнено, поскольку его левая часть не превосходит единицы.

Остается рассмотреть случай, когда G_{w_k} включен в цепь-решение. Если при этом G_{w_k} не является непосредственно предыдущим для G_{w_i} подграфом, включенным в цепь решение, то ограничение 3.18 выполнено поскольку в этом случае $\sum_{k<l<i} q_l > 1$.

Если же G_{w_k} это непосредственно предыдущий для G_{w_i} подграф, включенный в цепь-решение, то $\sum_{k<l<i} q_l = 0$ и $1 - q_k = 0$, и поэтому ограничение 3.18 будет выполнено только в том случае, если среди вершин множества $In_{P_k}(p_i j)$ найдется вершина-выход.

3.3 Сравнение способов формализации

Пусть n — число вершин в графе G, m — число подграфов G_{w_i}.

- Локальная задача существования с фиксированными входами и выходами

  Переменные: n

  Ограничения: $n + 2m$

- Локальная задача существования со свободными входами и выходами

  Переменные: $3n$

  Ограничения: $6n + 2m$

- Исходная задача существования

  Переменные: $3n + m$

  Ограничения: $5n + 2m + (m - 1)(n - \frac{m+1}{2})$

- Классический подход к формализации маршрутной задачи ([7])

  Переменные: n^2

  Ограничения: n^2

Как видно из приведенного сравнения, использование иерархической структуры графа предшествования партий G позволяет существенно снизить количество используемых переменных и ограничений по сравнению с известным способом сведения маршрутной задачи к задаче математического программирования в общем случае. Кроме того это сравнение демонстрирует то, насколько локальные задачи существования компактнее, чем исходная.

Глава 4

Программная реализация

В данной главе рассматривается программная реализация алгоритма решения задачи построения графика прокатки, сбалансированного по видам продукции. В рамках данной работы реализованы следующие этапы.

- Формирование структуры, хранящей данные для расчета
- Формирование структуры, содержащей переменные и ограничения соответствующей задачи математического программирования
- Трансляция задачи математического программирования для решения готовой библиотекой
- Построение простой цепи на основе решения, полученного с помощью готовой библиотеки

Для программной реализации использован язык Java. Для решения задач математического программирования используется система IBM ILOG CPLEX. В реализованной программе пользователь должен указать источник данных для расчета и выбрать режим расчета (указать вид задачи — локальная или исходная).

Входные данные должны содержать список партий прокатки (вершин графа G), целевое значение суммарного веса партий в графике прокатки и целевые значения суммарного веса по каждому виду продукции.

Vertex p представляет вершину p в графе G.

- *double* $p.width$ — ширина соответствующей партии p.
- *double* $p.thick$ — толщина соответствующей партии p.
- *double* $p.measure$ — значение функции l, то есть полезность соответствующей партии p.
- *ProdType* $p.prodType$ — вид готовой продукции партии p.

Таким образом, задание для расчета должно содержать:

- *ArrayList<Vertex> vertices* — набор партий. Каждая партия представляет собой вершину $Vertex\ v$.
- *ArrayList<ProdType> prods* — набор видов готовой продукции, которые должны получиться после прокатки. Для каждой вершины v из $vertices$ в $prods$ найдется $v.prodType$.
- *ArrayList<double> prodTotalLength* — целевое значение суммарной длины партий по каждому виду продукции.

- *double Lmin* — минимальное значение суммарной длины партий в графике прокатки
- *double Lmax* — максимальное значение суммарной длины партий в графике прокатки

4.1 Результаты экспериментов

Проведена серия вычислительных экспериментов с использованием реальных данных полученных из корпоративной сети ОАО «ММК». Произведены расчеты более чем на 500 наборах исходных данных, каждый из которых включает порядка 200-300 партий прокатки. Вычисления осуществлялись на персональном компьютере.

Эксперименты показали, что время на поиск решения исходной задачи существования или установления того, что задача несовместна укладывается в 5 минут.

Эксперименты показали, что локальная задача существования может быть использована для получения решений в более компактной схеме. В предыдущей работе [4] предложен алгоритм кубической временной трудоемкости, решающий задачу поиска простой цепи максимального веса. По такой цепи естественным образом строится набор подграфов с входами и выходами. Используя эти данные в качестве начальных, зачастую можно коротким путём получить решение локальной задачи существования и, соответственно, решение исходной задачи.

Также проведен эксперимент по решению задачи построения сбалансированного графика прокатки, формализованной классическим способом (квадратичное число переменных и ограничений). Эксперимент показал, что на небольших выборках (до 50 партий), возможно получить решение за несколько минут, однако при больших выборках дожидаться времени окончания работы алгоритма практически нецелесообразно.

В таблице 4.1 представлено сравнение производительности алгоритма решения задачи построения сбалансированного графика прокатки для различных способов формализации задачи.

В целом эксперименты показали пригодность разработанного подхода для решения практикой задачи построения графика прокатки, сбалансированного по видам продукции.

Таблица 4.1: Результаты расчетов

Задача	**Среднее время решения, сек**
ЛЗС со свободными входами и выходами	1-2
ЛЗС с фиксированными входами и выходами	3-5
Исходная задача существования	30-50
Классический подход	—

Заключение

В настоящей работе предложена математическая модель задачи построения графика прокатки, сбалансированного по видам продукции. На множестве партий был построен ориентированный вершинно-взвешенный раскрашенный граф предшествования партий G. Практическая задача построения сбалансированного графика прокатки сведена к задаче поиска сбалансированной простой цепи в графе предшествования партий G.

Задача поиска сбалансированной простой цепи была формализована в виде задачи целочисленного программирования. Формализация опирается на исследование структуры графа предшествования партий G. Это позволяет получить более компактную по числу неравенств задачу математического программирования, чем при использовании классических техник (например, для решения задачи коммивояжера). Также было предложено несколько упрощенных и приближенных вариантов формализации задачи поиска сбалансированной простой цепи, которые позволяют в некоторых случаях получить решение за более короткий промежуток времени. Было показано, что предложенная формализация задачи поиска сбалансированной простой цепи является корректной, то есть по решению задачи математического программирования можно построить решение задачи поиска сбалансированной простой цепи. Отметим тот факт, что если полученная задача математического программирования является несовместной, то исходная задача поиска сбалансированной простой цепи не имеет решения. Таким образом, предложенный подход позволяет за приемлемое с практической точки зрения время найти решение задачи поиска сбалансированной простой цепи, либо установить, что его нет.

Литература

[1] *Асанов М.О., Баранский В.А., Расин В.В.* Дискретная математика: Графы, Матроиды, Алгоритмы. - Ижевск: НИЦ «РХД», 2001, 288 стр.

[2] *Леонова С.И.* Задача построения графика горячей прокатки, сбалансированного по видам продукции. Исследование свойств графика.//Магистерская диссертация, ИМКН УрФУ, Екатеринбург, 2016

[3] *Леонова С.И.* Разработка и исследование алгоритмов построениея экстремальных цепей в вершинно-взвешенных ориентированных графах. Частный алгоритм.//Бакалаврская диссертация, ИМКН УрФУ, Екатеринбург, 2014

[4] *Березин А.А.* Разработка и исследование алгоритмов построениея экстремальных цепей в вершинно-взвешенных ориентированных графах. Общий алгоритм.//Бакалаврская диссертация, ИМКН УрФУ, Екатеринбург, 2014

[5] 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

[6] Березин А.А., Вакула И.А. Задача построения сбалансированного графика прокатки // Труды 47-й Международной молодежной школы-конференции, ИММ УрО РАН, Екатеринбург, 2016

[7] *Dantzig G.B., Fulkerson D.R., Johnson S.* On a linear programming combinatorial approach to the traveling salesman problem, Operations Research, 7 (1959), pp. 58–66

[8] *Balas E.* The prize collecting travelling salesman problem // Networks, October 1989. — Vol. 19. — P. 621–636.

[9] *Balas E., Clarence H.M.* Combinatorial optimization in steel rolling. Workshop on Combinatorial Optimization in Science and Technology, April, 1991.

[10] *Lixin Tang, Jiyin Liu, Aiying 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.

[11] *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.

Приложение

Получение данных для расчета

```
public class RollingSchedulingTask {

   public ArrayList<Vertex> vertex;
   public double totalLength;
   public ArrayList<Flow> flows;
   public ArrayList<Double> flowsTotalLength;
   public ArrayList<Block> levels;
   public int levelAmount;
   public double Lmin;
   public double Lmax;
   public int optFlow =- 1;
   public String name;
   public int calcID;
   public long startTime;

   public RollingSchedulingTask() {}

   /** Загружает задание, используя переданный connection */
   public RollingSchedulingTask(int calcID, double lmin, double lmax,
       Connection cn) {
      this.vertex = RollingSchedulingLoader.loadVertexes(calcID, cn);
      this.flows = RollingSchedulingLoader.loadFlows(calcID, cn);
      this.startTime = RollingSchedulingLoader.loadStartTime(calcID, cn);
      prepareTask(calcID, lmin, lmax);
   }

   private void printTimeStat() {
      ArrayList<Vertex> sortedByReadyTime = new ArrayList<Vertex>();
      sortedByReadyTime.addAll(vertex);
      Collections.sort(sortedByReadyTime, new SortByReadyTime());
      Map<Integer, Double> hoursStat = new HashMap<Integer, Double>();
      ArrayList<Integer> hoursAfterStartTime = new ArrayList<Integer>();
      double acc = 0;
      for (Vertex v : sortedByReadyTime) {
         int h = (int) (v.readyTime - this.startTime) / 3600;
         if (!hoursAfterStartTime.contains(h)) {
            hoursAfterStartTime.add(h);
            System.out.println(h);
         }
         acc += v.measure;
         hoursStat.put(h, acc);
      }
      DecimalFormat formatter = new DecimalFormat("#.#", new
```

```
            DecimalFormatSymbols(Locale.GERMAN));
        for (int i : hoursAfterStartTime) {
            System.out.println(formatter.format(hoursStat.get(i)));
        }
    }

    private void prepareTask(int calcID, double lmin, double lmax) {
        this.Lmin = lmin;
        this.Lmax = lmax;
        this.calcID = calcID;

        //выставляем свободные границы для потоков, для которых нет ограничений
        for (Vertex v : this.vertex) {
            if (!Flow.contains(this.flows, v.flowType)) {
                this.flows.add(new Flow(v.flowType, 0, Lmax));
            }
        }

        this.flowsTotalLength = new ArrayList<Double>();
        for (int i = 0; i < this.flows.size(); i++) {
            this.flowsTotalLength.add(0.0);
        }

        for (Vertex v : this.vertex) {
            for (int i = 0; i < this.flows.size(); i++) {
                if (v.flowType.equals(this.flows.get(i).name)) {
                    this.flowsTotalLength.set(i, this.flowsTotalLength.get(i) +
                        v.measure);
                    this.totalLength += v.measure;
                    break;
                }
            }
        }

        this.organizeLimits();

        this.levels = BlockStructure.getLevels(this.vertex);
        this.levelAmount = this.levels.size();
    }

    private void organizeLimits() {
        //выставляем реальные границы для потоков, для которых имеется меньше, чем
            минимальный объем
        int i = 0;
        for (Flow f : flows) {
            if (f.min > flowsTotalLength.get(i)) {
                f.min = 0.99 * flowsTotalLength.get(i);
            }
            i++;
        }
    }

    /** Загружает задание, используя переданный connection, и генерирует
        последовательность уровней */
```

```
public RollingSchedulingTask(int calcID, Crit crit, double lmin, double
    lmax, Connection cn) {
   this(calcID, lmin, lmax, cn);


   for (Block b : this.levels) {
      b.isUsed = true;
   }
}

public void setLmin(double Lmin) {
   this.Lmin = Lmin;
}

public void setLmax(double Lmax) {
   this.Lmax = Lmax;
}

public void refineFlowTask() {
   int i = 0;
   for (Flow f : flows) {
      if (f.min > flowsTotalLength.get(i)) {
         f.min = 0.1 * flowsTotalLength.get(i);
      }
      if(f.max < 0.10 * Lmax) {
         f.max = 0.10 * Lmax;
      }
   }
}

public String showFlowTaskStat() {
   String s="";
   s += "/-----------------------------------/\n";
   s += "Целевое распределение\n";
   s += "Min: " + Lmin + "; Max: " + Lmax + "\n";
   for (Flow f : flows) {
      s += f.name + ": min " + f.min + "; max " + f.max + "\n";
   }
   s += "/-----------------------------------/" + "\n";
   s += "Фактическое распределение в задании" + "\n";
   s += "Exists: " + totalLength + "\n";
   for (int i = 0; i < flows.size(); i++) {
      s += flows.get(i).name + ": " + flowsTotalLength.get(i) + "\n";
   }
   return s;
}

public void printFlowTaskStat() {
   System.out.println(showFlowTaskStat());
}

public void setLowLimsTo(double v) {
   for (int i = 0; i < flows.size(); i++) {
      flows.get(i).min=v;
```

```
    }
  }

  public void setUpperLimsTo(double v) {
    for (int i = 0; i < flows.size(); i++) {
      flows.get(i).max=v;
    }
  }

  public void setStartTime(long startTime) {
    this.startTime = startTime;
  }
}
```

Перевод задания для построения графика в задачу математического программирования

```
public class RollingSchedulingProblem {

  public static Problem createProblem(RollingSchedulingTask t,
      RollingMathModel model, Crit crit, RollingMathOptFunc func) {
    Problem p = new Problem("HotRolling");
    if (model != RollingMathModel.NO_LEVELS_STRUCTURE) {
      for (int i = 1; i <= t.levelAmount; i++) {
        int size = t.levels.get(i-1).size;
        VarArray B = new VarArray(String.format("B[%d]", i), 1, new int[]
            {size}, VarType.BV, BoundType.DB, 0, 1);
        p.addVars(B);
      }
      if (model == RollingMathModel.FREE_ENTERS_EXITS || model ==
          RollingMathModel.FREE_LEVELS_SEQUENCE) {
        for (int i = 1; i <= t.levelAmount; i++) {
          int size = t.levels.get(i-1).size;
          VarArray S = new VarArray(String.format("S[%d]", i), 1, new int[]
              {size}, VarType.BV, BoundType.DB, 0, 1);
          p.addVars(S);
        }
        for (int i = 1; i <= t.levelAmount; i++) {
          int size = t.levels.get(i-1).size;
          VarArray E = new VarArray(String.format("E[%d]", i), 1, new int[]
              {size}, VarType.BV, BoundType.DB, 0, 1);
          p.addVars(E);
        }
      }
      if (model == RollingMathModel.FREE_LEVELS_SEQUENCE) {
        int size = t.levelAmount;
        VarArray Q = new VarArray("Q", 1, new int[] {size}, VarType.BV,
            BoundType.DB, 0, 1);
        p.addVars(Q);
      }
      Constraint c = new Constraint("weightConstr");
      // Ограничение на вес
      for (int i = 1; i <= t.levelAmount; i++) {
        for (int j = 1; j <= t.levels.get(i-1).size; j++) {
          double lij = t.levels.get(i-1).get(j-1).measure;
```

```
            c.addVarEntry(String.format("B[%d]", i), new int[] {j}, lij);
        }
    }
    c.setLowBound(t.Lmin);
    c.setUpBound(t.Lmax);
    c.setType(BoundType.DB);
    p.addConstraint(c);

    // Ограничение на виды продукции
    for (int k = 1; k <= t.flows.size(); k++) {
        c = new Constraint(String.format("prodFlowConstr[%d]", k));
        for (int i = 1; i <= t.levelAmount; i++) {
            for (int j = 1; j <= t.levels.get(i-1).size; j++) {
                if
                    (t.levels.get(i-1).get(j-1).flowType.equals(t.flows.get(k-1).name))
                    {
                    double lij = t.levels.get(i-1).get(j-1).measure;
                    c.addVarEntry(String.format("B[%d]", i), new int[] {j}, lij);
                }
            }
        }
        c.setLowBound(t.flows.get(k-1).min);
        c.setUpBound(t.flows.get(k-1).max);
        c.setType(BoundType.DB);
        p.addConstraint(c);
    }

    if (model == RollingMathModel.FIXED_ENTERS_EXITS) {
        addPrecConstraintsWithFixedLevelsSequenceAndFixedEntersAndExits(p,
            t, crit);
    } else if (model == RollingMathModel.FREE_ENTERS_EXITS) {
        addPrecConstraintsWithFixedLevelsSequenceAndFreeEntersAndExits(p, t,
            crit);
    } else if (model == RollingMathModel.FREE_LEVELS_SEQUENCE) {
        addPrecConstraintsWithFreeLevelsSequence(p, t, crit);
        addReadyTimeConstraints(p, t);
    }

    ObjectiveFunction f = new ObjectiveFunction("Objective");
    if (func == RollingMathOptFunc.TOTAL_VALUE_MAX || func ==
        RollingMathOptFunc.TOTAL_VALUE_MIN) {
        for (int i = 1; i <= t.levelAmount; i++) {
            for (int j = 1; j <= t.levels.get(i-1).size; j++) {
                double lij = t.levels.get(i-1).get(j-1).measure;
                f.addVarEntry(String.format("B[%d]", i), new int[] {j}, lij);
            }
        }
    } else if (func == RollingMathOptFunc.FLOW_LOWER_AMOUNT || func ==
        RollingMathOptFunc.FLOW_UPPER_AMOUNT) {
        if (t.optFlow >= 0) {
            String sopt = t.flows.get(t.optFlow).name;
            for (int i = 1; i <= t.levelAmount; i++) {
                for (int j = 1; j <= t.levels.get(i-1).size; j++) {
                    double lij = t.levels.get(i-1).get(j-1).measure;
```

```
                String s = t.levels.get(i-1).get(j-1).flowType;
                if(s.equals(sopt))
                   f.addVarEntry(String.format("B[%d]", i), new int[] {j},
                       lij);
             }
          }
       }
    }

    if (func == RollingMathOptFunc.TOTAL_VALUE_MAX || func ==
        RollingMathOptFunc.FLOW_LOWER_AMOUNT) {
       f.setType(ObjectiveType.MAX);
    } else if (func == RollingMathOptFunc.TOTAL_VALUE_MIN || func ==
        RollingMathOptFunc.FLOW_UPPER_AMOUNT) {
       f.setType(ObjectiveType.MIN);
    }
    p.setObjectiveFunction(f);
 }
 if (model == RollingMathModel.NO_LEVELS_STRUCTURE) {
    int vSize = t.vertex.size();
    for (int i = 1; i <= vSize; i++) {
       VarArray P = new VarArray(String.format("P[%d]", i), 1, new int[]
           {vSize}, VarType.BV, BoundType.DB, 0, 1);
       p.addVars(P);
    }
    Constraint c = null;
    Constraint c1 = null;
    Constraint c2 = null;
    for (int i = 1; i <= vSize; i++) {
       c1 = new
           Constraint(String.format("OneVertexToOnePositionConstr[%d]", i));
       c2 = new
           Constraint(String.format("EachVertexUsedOnlyOnceConstr[%d]", i));
       for (int j = 1; j <= vSize; j++) {
          c1.addVarEntry(String.format("P[%d]", i), new int[] {j}, 1.0);
          c2.addVarEntry(String.format("P[%d]", j), new int[] {i}, 1.0);
       }
       c1.setLowBound(1.0);
       c1.setUpBound(1.0);
       c1.setType(BoundType.UP);
       p.addConstraint(c1);
       c2.setLowBound(1.0);
       c2.setUpBound(1.0);
       c2.setType(BoundType.UP);
       p.addConstraint(c2);
    }

    for (int i = 1; i < vSize; i++) {
       c = new Constraint(String.format("PositionOrderConstr[%d]", i));
       for (int j = 1; j <= vSize; j++) {
          c.addVarEntry(String.format("P[%d]", i), new int[] {j}, -1.0);
          c.addVarEntry(String.format("P[%d]", i+1), new int[] {j}, 1.0);
       }
       c.setLowBound(0.0);
```

```
   c.setUpBound(0.0);
   c.setType(BoundType.UP);
   p.addConstraint(c);
}

long start = System.currentTimeMillis();
Graph g = new Graph(t.vertex, crit);
for (int i = 2; i <= vSize; i++) {
   for (int j = 1; j <= vSize; j++) {
      c = new Constraint(String.format("ConnectivityConstr[%d,%d]", i,
          j));
      c.addVarEntry(String.format("P[%d]", i), new int[] {j}, 1.0);
      Vertex v_j = t.vertex.get(j-1);
      ArrayList<Integer> In = getInIndexes(g, v_j, crit);
      for (int k : In) {
         c.addVarEntry(String.format("P[%d]", i-1), new int[] {k}, -1.0);
      }
      c.setLowBound(0.0);
      c.setUpBound(0.0);
      c.setType(BoundType.UP);
      p.addConstraint(c);
   }
}
long finish = System.currentTimeMillis();
System.out.println(String.format("Connectivity constraint time: %d",
    finish - start));
c = new Constraint("WeightConstr");
// Ограничение на вес
for (int i = 1; i <= vSize; i++) {
   for (int j = 1; j <= vSize; j++) {
      double lj = t.vertex.get(j-1).measure;
      c.addVarEntry(String.format("P[%d]", i), new int[] {j}, lj);
   }
}
c.setLowBound(t.Lmin);
c.setUpBound(t.Lmax);
c.setType(BoundType.DB);
p.addConstraint(c);
/**/
// Ограничение на виды продукции
for (int k = 1; k <= t.flows.size(); k++) {
   c = new Constraint(String.format("FlowConstr[%d]", k));
   for (int i = 1; i <= vSize; i++) {
      for (int j = 1; j <= vSize; j++) {
         if (t.vertex.get(j-1).flowType.equals(t.flows.get(k-1).name)) {
            double lj = t.vertex.get(j-1).measure;
            c.addVarEntry(String.format("P[%d]", i), new int[] {j}, lj);
         }
      }
   }
   c.setLowBound(t.flows.get(k-1).min);
   c.setUpBound(t.flows.get(k-1).max);
   c.setType(BoundType.DB);
   p.addConstraint(c);
```

```
        }
        /**/
        ObjectiveFunction f = new ObjectiveFunction("Objective");
        if (func == RollingMathOptFunc.TOTAL_VALUE_MAX || func ==
            RollingMathOptFunc.TOTAL_VALUE_MIN) {
           for (int i = 1; i <= vSize; i++) {
              for (int j = 1; j <= vSize; j++) {
                 double lj = t.vertex.get(j-1).measure;
                 f.addVarEntry(String.format("P[%d]", i), new int[] {j}, lj);
              }
           }
        } else if (func == RollingMathOptFunc.FLOW_LOWER_AMOUNT || func ==
            RollingMathOptFunc.FLOW_UPPER_AMOUNT) {
           if (t.optFlow >= 0) {
              String sopt = t.flows.get(t.optFlow).name;
              for (int i = 1; i <= vSize; i++) {
                 for (int j = 1; j <= vSize; j++) {
                    double lj = t.vertex.get(j-1).measure;
                    String s = t.vertex.get(j-1).flowType;
                    if (s.equals(sopt)) {
                       f.addVarEntry(String.format("P[%d]", i), new int[] {j},
                           lj);
                    }
                 }
              }
           }
        }

        if (func == RollingMathOptFunc.TOTAL_VALUE_MAX || func ==
            RollingMathOptFunc.FLOW_LOWER_AMOUNT) {
           f.setType(ObjectiveType.MAX);
        } else if (func == RollingMathOptFunc.TOTAL_VALUE_MIN || func ==
            RollingMathOptFunc.FLOW_UPPER_AMOUNT) {
           f.setType(ObjectiveType.MIN);
        }
        p.setObjectiveFunction(f);

     }
     return p;
  }
```

Взаимодействие с системой IBM ILOG CPLEX

```
public class Translator {

   public static IloCplex toCPLEXProblem(IloCplex cplex, Problem p,
       IntegerCounter c2, boolean printSolverLog) throws IloException {
      cplex.clearModel();
      IloNumVar[] allVars = null;
      String[] names = new String[p.getVarAmount()];
      double[] lb = new double[p.getVarAmount()];
      double[] ub = new double[p.getVarAmount()];
      IloNumVarType[] types = new IloNumVarType[p.getVarAmount()];
      Map<String,Integer> varOffsets = new HashMap<String, Integer>();
      int i = 0;
```

```
for (VarArray v : p.getVarArrays()) {
   ArrayList<String> vNames = new ArrayList<String>();
   int dims = v.getDim();
   int[] dimSize = v.getDimSize();
   int[] c = new int[v.getDimSize().length];
   String varName = v.name;
   cplexNameLoops(vNames, varName, dims, dimSize, 0, c);
   varOffsets.put(v.name, i);
   for (int j = i; j < i + v.size; j++) {
      names[j] = vNames.get(j-i);
      lb[j] = v.lowBound;
      ub[j] = v.upBound;
      types[j] = cplexType(v.varType);
   }
   i += v.size;
}
allVars = cplex.numVarArray(p.getVarAmount(), lb, ub, types, names);

for (Constraint c : p.getConstraints()) {
   IloLinearNumExpr expr = cplex.linearNumExpr();
   for (VarEntry v : c.getVarEntries()) {
      expr.addTerm(v.coef, allVars[getColsIndex(p, v, varOffsets)-1]);
   }
   switch (c.type) {
      case UP : cplex.addLe(expr, c.upBound, c.name); break;
      case LO : cplex.addGe(expr, c.lowBound, c.name); break;
      case FX : cplex.addEq(expr, c.lowBound, c.name); break;
      case DB : cplex.addGe(expr, c.lowBound, c.name); cplex.addLe(expr,
          c.upBound, c.name); break;
   default:
      break;
   }
}

ObjectiveFunction f = p.getObjectiveFunction();
if (f.getType() != ObjectiveType.NO) {
   IloLinearNumExpr objExpr = cplex.linearNumExpr();
   for (VarEntry v : f.getVarEntries()) {
      objExpr.addTerm(v.coef, allVars[getColsIndex(p, v, varOffsets) - 1]);
   }

   if (f.getType().equals(ObjectiveType.MAX)) {
      cplex.addMaximize(objExpr);
   } else {
      cplex.addMinimize(objExpr);
   }
}
if(!printSolverLog)
   cplex.setOut(null);
if (cplex.solve()) {
   c2.set(1);
   if(printSolverLog) {
        cplex.output().println("Solution status = " + cplex.getStatus());
        cplex.output().println("Solution value = " + cplex.getObjValue());
```

```
        }

          double[] val = cplex.getValues(allVars);

          int ncols = cplex.getNcols();
          int varArrayCounter = 0;
          int localArrayCounter = 0;
          VarArray v = p.getVarArrays().get(varArrayCounter);
          for (int j = 0; j < ncols;) {
//          cplex.output().println(names[j] + " = " + val[j]);
           if (localArrayCounter < v.size) {
              v.getVars().add(val[j]);
              localArrayCounter++;
              j++;
           } else {
              varArrayCounter++;
              v = p.getVarArrays().get(varArrayCounter);
              localArrayCounter = 0;
           }
          }
       }
      return cplex;
   }

   private static IloNumVarType cplexType(VarType varType) {
      switch (varType) {
         case IV : return IloNumVarType.Int;
         case BV : return IloNumVarType.Bool;
         default : return IloNumVarType.Float;
      }
   }


   private static int getColsIndex(Problem p, VarEntry varEntry, Map<String,
       Integer> varOffsets) {
      // смещение до нужного блока с переменными + индекс внутри этого блока
      return varOffsets.get(varEntry.varArrayName) +
          p.get(varEntry.varArrayName).lineIndex(varEntry.index);
   }


   public static String indexString(int[] a) {
      String s = "[";
      for (int i : a) {
         s += i + ",";
      }
      return s.substring(0, s.length()-1) + "]";
   }

   public static void cplexNameLoops(ArrayList<String> names, String varName,
       int dims, int[] dimSize, int n, int[] c) {
       if (n == dims) {
         names.add(varName + indexString(c));
         return;
```

```
      }
      c[n] = 0;
      for (int i = 1; i <= dimSize[n]; i++) {
        c[n]++;
          cplexNameLoops(names, varName, dims, dimSize, n+1, c);
      }
   }
}
```