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

Разработка и исследование алгоритмов построения экстремальных цепей в вершинно-взвешенных орграфах Общий алгоритм

Допущен к защите

_____

«__»_____ 2014 г.

Квалификационная работа
на степень бакалавра наук
по направлению «Математика,
прикладная математика»
студента группы МТ-401
Березина Антона Александровича

Научный руководитель
Вакула Игорь Александрович,
кандидат физико-математических
наук, заведующий сектором
адаптивных систем управления
ИММ УрО РАН

Екатеринбург
2014

Эта страница появилась здесь, потому что я не умею нормально нумеровать страницы в TeX

РЕФЕРАТ

Березин А.А. РАЗРАБОТКА И ИССЛЕДОВАНИЕ АЛГОРИТМОВ ПОСТРОЕНИЯ ЭКСТРЕМАЛЬНЫХ ЦЕПЕЙ В ВЕРШИННО-ВЗЕШЕННЫХ ОРГРАФАХ. ОБЩИЙ АЛГОРИТМ,

выпускная квалификационная работа на степень бакалавра наук: стр. 21, рис. 4, табл. 1, форм. 14, библ. 10 назв.

Ключевые слова: АЛГОРИТМЫ, ОРИЕНТИРОВАННЫЕ ГРАФЫ, ПРОСТЫЕ ЦЕПИ, ГОРЯЧАЯ ПРОКАТКА

Объект исследования — ориентированные графы специального вида.

Цель работы — построение простых цепей максимального веса в вершинно-взвешенных ориентированных графах специального вида.

В процессе работы исследуется структура графов, возникающих при моделировании задачи планирования графиков горячей прокатки. Разрабатываются алгоритмы поиска простых цепей максимального веса. Оценивается их сложность. Полученные результаты применяются для решения более общих задач в области планирования металлургического производства.

Оглавление

Глава 1

Введение

В настоящей работе рассматриваются ориентированные графы специального вида. Эти графы возникают при математической постановке задачи построения графиков прокатки для металлургических предприятий. Ресурсом для прокатки металла являются так называемые *партии* — металлические заготовки обладающие определеными геометрическими параметрами. *График прокатки* представляет собой упорядоченный набор партий, прокатываемых на стане горячей прокатки. Для получения качественного металла необходимо соблюдать ряд ограничений на порядок следования партий друг за другом. Таким образом, график прокатки должен удовлетворять всем технологическим ограничениям и при этом обладать "хорошим"сочетанием показателей по ряду критериев оценки. В настоящей работе в качестве критерия выбрана максимальная суммарная длина прокатанных рулонов.
В ряде работ (например, [2], [3]) 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 энергосбережния 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, [7]). Существуют модели PCTSP (Prize Collecting TSP), предложенная Балашом ([4], [5]) и PCVRP (Prize Collecting VRP, [6]), 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 возможнен 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.

Учитывая специфичный способ описания ограничений предшествования партий в графике прокатки, принятый на отечественных предприятиях (например, ОАО "ММК"), ограничения удобно представлять в виде ориентированного графа. Предлагается удалить из целевой функции составляющую штрафов за переход, а сами

переходы отнести в состав ограничений. Таким образом, задача построения графиков прокатки в своих базовых ограничениях сводится к задаче поиска простой цепи достаточно большого веса в вершинно-взвешенном ориентированном графе. Смежность узлов в этом графе задается соотношением между ширинами и толщинами последовательно прокатываемых партий.
Основным результатом является то, что благодаря такому способу задания смежности удается построить алгоритм для нахождения простой цепи (графика прокатки) максимального веса, сложность которого ограничена кубом числа узлов (партий). Хотя в общем случае задача поиска простой цепи максимального веса труднорешаема.
Созданный алгоритм используется для оперативного планирования на станах прокатки в некоторых цехах ОАО "ММК".

Поясним причину выбора критерия оптимизации. Оценки графиков прокатки связаны с особенностями организации производства на конкретном предприятии. Вместе с тем общим является то, что график прокатки должен содержать достаточно таких партий, чтобы суммарный вес/длина партий были достаточно велики. Более того, наличие "большого"технологически допустимого графика прокатки позволяет на следующем этапе в автоматическом или ручном режиме его отредактировать, чтобы получить желаемый результат по всей совокупности практических требований. Также в качестве веса партии может выступать величина, характеризующая не только ее физические характеристики, но и предпочтительность по срокам, видам продукции и т.п. По указанным причинам для получения оптимизационной постановки задачи естественно потребовать максимальности суммарного веса партий, входящих в график прокатки.

Глава 2

Постановка задачи

Обозначим через P множество партий $\{p_1, p_2, \ldots, p_k\}$, где $k \in \mathbb{N}$.
Пусть

- $w : P \to \mathbb{R}^+$ — ширина партии;
- $t : P \to \mathbb{R}^+$ — толщина партии;
- $l : P \to \mathbb{R}^+$ — длина партии; Для удобства в дальнейшем будем называть её весом, таким образом, l — весовая функция.
- $r : \mathbb{R}^+ \to \mathbb{R}^+$ — монотонно неубывающая функция, определяющая максимальную величину допустимой разности значений толщины между двумя соседними партиями;
- $\mathbb{W} \in \mathbb{R}^+$ — величина, определяющая максимальную величину допустимой разности значений ширины между двумя соседними партиями.

Геометрические параметры партий являются основным источником накладываемых ограничений на порядок следования партий друг за другом в графике прокатки. Партия q может непосредственно следовать за партией p в графике прокатки в том и только в том случае, когда выполнены:

1. ограничение "для перехода по толщине":
$$|t_p - t_q| \leq \min\{r(t_p), r(t_q)\} \tag{2.1}$$
2. ограничение "для перехода по ширине":
$$0 \leq w_p - w_q \leq \mathbb{W} \tag{2.2}$$

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.

Множество всех допустимых графиков прокатки совпадает со множеством $\mathbb{B}(G)$ всех простых цепей в G.

Для простой цепи z через $l(z)$ обозначим её вес:

$$l(z) \triangleq \sum_{p \in P(z)} l(p) \tag{2.3}$$

Сформулируем **задачу** оптимизации — найти в $\mathbb{B}(G)$ цепь максимального веса:

$$l(z) \to \max, \text{ где } z \in \mathbb{B}(G) \tag{2.4}$$

Пусть p - произвольный узел в G. Обозначим через:

- $In(p)$ — множество узлов q таких, что дуга $qp \in E$
  $|In(p)|$ — входящая валентность узла p
- $In^+(p) = \{q \in In(p) | w_q > w_p\}$
- $Out(p)$ — множество узлов q таких, что дуга $pq \in E$
  $|Out(p)|$ — исходящая валентность узла p
- $Succ(p)$ — множество всех узлов q, для которых существует ориентированный путь pq в графе G. Из общей теории графов известно, что в этом случае также существует и простая цепь из p в q.
- $Pred(p)$ — множество всех узлов q, для которых существует ориентированный путь из q в p в графе G. Отметим, что в этом случае также существует и простая цепь из q в p.
- $W = \{w_p | p \in P\}$ — всевозможные значения ширин узлов
- $P_w = \{p | w_p = w\}$, где $w \in W$, — множество узлов, имеющих данную ширину w.
- G_w — граф, индуцированный множеством P_w.

Заметим, что любая наибольшая по весу простая цепь является максимальной по включению узлов. Поэтому для решения задачи 2.4 достаточно искать наибольшую по весу цепь среди максимальных по включению узлов цепей.

Глава 3

Метод решения

Для эффективного решения задачи предложен следующий подход:

1. Разрабатывается алгоритм, состоящий из двух частей - алгоритм, решающий задачу 2.4 в подграфах G_w и общий алгоритм, решающий задачу 2.4 в графе G. В настоящей работе рассматривается общий алгоритм, использующий частный алгоритм для подграфов (частный алгоритм описан в [1])в качестве подпрограммы.

2. Описывается способ оптимизации работы алгоритма. Предлагается подавать на вход алгоритма подграф G' графа G с тем же множеством узлов, что и у G, имеющий то же множество максимальных по включению цепей, что и в исходном графе. Нетрудно видеть, что в частности можно взять $G' = G$. Предлагается эффективная конструкция графа G', позволяющая существенно снизить время работы алгоритма.

3. Проводится анализ сложности алгоритма.

4. Приводятся результаты тестов и их анализ.

Глава 4

Общий алгоритм поиска цепи максимального веса

4.1 Анализ структуры графа

Для лучшего понимания структуры графа и удобства разработки алгоритма придуман удобный и естественный способ укладки графа на плоскости. Геометрические характеристики узлов выступают в роли координат. Для визуализации разработан плагин укладки для открытой программы Gephi [10].

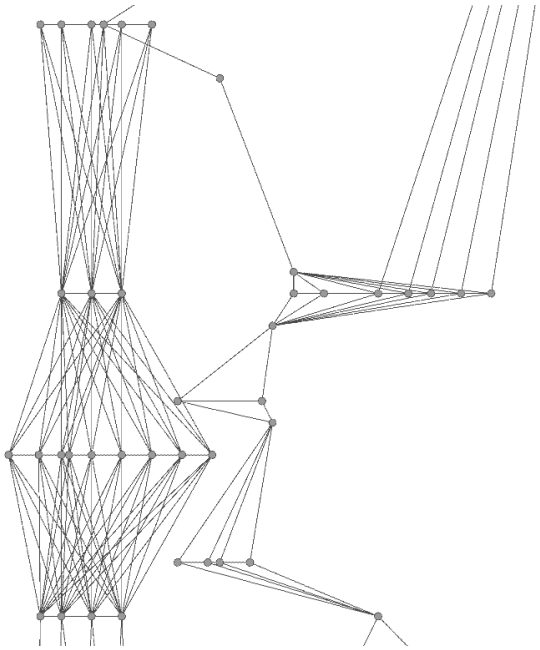

Рис. 4.1: Структура графа

На рис. (4.1) изображен фрагмент графа.

По оси абсцисс откладывается толщина узлов, по оси ординат откладывается ширина узлов, начало координат находится в левом нижнем углу. Видно, что граф складывается из подграфов G_w таким образом, что дуги между подграфами направлены только вниз. Это обеспечивается условием предшествования партий по ширине. Внутри любого подграфа G_w для каждой пары вершин $p, q \in P_w$ из существования дуги pq следует существование обратной дуги qp. Эта симметрия обеспечивается модулем в условии предшествования по толщине. В данной работе мы рассмотрим алгоритм для решения задачи в G с использованием готового алгоритма для решения задачи в подграфе G_w.

4.2 Неформальное описание алгоритма

Идея общего алгоритма основывается на принципе динамического программирования и заключается в постепенном решении задачи "сверху вниз". Для каждого узла в каждом подграфе последовательно строятся максимальные по весу цепи, заканчивающиеся в данном узле, вместе с тем из них выбирается максимум.
Частный алгоритм для каждой пары узлов p и q в подграфе G_w предъявляет простую цепь максимального веса начинающуюся в p и заканчивающуюся в q (либо указывает, что не существует маршрута соединяющего p и q).
Обозначим такую цепь H_{pq}. Тогда очевидно, что

$$\max_{p,q \in P_w} \{H | H = H_{pq}\} \tag{4.1}$$

это решение задачи 2.4 в G_w.
Рассмотрим случай, когда G состоит из двух подграфов G_{w_0} и G_{w_1}, что $w_0 > w_1$. Заметим, что $\forall p \in P(G_{w_0})\ In^+(p) = \varnothing$.
Таким образом для простой цепи $H = p \to v_1 \to \ldots \to v_n \to q$, являющейся решением задачи возможны следующие варианты.

1. $p, q \in P(G_{w_0})$ и $\forall i = \overline{1,n}\ v_i \in P(G_{w_0})$
2. $p, q \in P(G_{w_1})$ и $\forall i = \overline{1,n}\ v_i \in P(G_{w_1})$
3. $p \in P(G_{w_0}), q \in P(G_{w_1})$ и $\exists k\ 0 \le k \le n$ такое, что:

$$\begin{cases} v_i \in P(G_{w_0}), & \text{если } 1 \le i \le k; \\ v_i \in P(G_{w_1}), & \text{если } k < i \le n. \end{cases}$$

Варианты 1 и 2 означают, что цепь целиком лежит в каком-то одном подграфе. Третий вариант означает, что часть узлов цепи находится в одном подграфе, а часть в другом. Здесь удобно ввести вспомогательное понятие.

Определение. *Для двух произвольных простых цепей $H_1 = p_1 \to p_2 \to \ldots \to p_n$ и $H_2 = q_1 \to q_2 \to \ldots \to q_m$, не имеющих общих узлов и таких, что $p_n q_1 \in E$,*
суммой *этих цепей назовем цепь $H = p_1 \to \ldots \to p_n \to q_1 \to \ldots \to q_m$*
Обозначим $H = H_1 + H_2$.

Таким образом вариант 3 означает, что решение является суммой двух цепей, каждая из которых лежит в своем подграфе.

Будем действовать следующим образом:
Для каждого узла $p \in P(G_{w_0})$ переберем все узлы $s \in P(G_{w_0})$, каждый раз определяя H_{sp} простую цепь максимального веса, соединяющую эти узлы, лежащую в G_{w_0}.
Обозначим через $H[p] = \max\limits_s \{H | H = H_{sp}\}$ — максимальную по весу простую цепь, оканчивающуюся узлом p.
На нулевом шаге максимальной простой цепью в G является

$$H_{max} = \max_p \{H | H = H[p]\} \tag{4.2}$$

Выберем и зафиксируем в G_{w_1} узел t. Выберем в G_{w_1} еще один узел q, возможно тот же самый.
H_{qt} – простая цепь максимального веса, соединяющая эти узлы, лежащая в G_1.
Если $In^+(q) = \varnothing$, то полагаем $H[t] = \max\{H_{qt}, H[t]\}$.

В противном случае просмотрим узлы $p \in In^+(q)$. Для каждого из них на предыдущем шаге уже рассчитаны максимальные по весу цепи $H[p]$.
Выберем такой узел p, у которого $l(H[p])$ максимальна.
Сконструируем сумму $H_{ptq} = H[p] + H_{qt}$. Положим $H[t] = \max\{H_{ptq}, H[t]\}$.
Решением задачи в исходном графе G будет

$$H_{max} = \max\{\max_t\{H[t]\}, H_{max}\} \tag{4.3}$$

Понятно, что описанный подход работает и в случае, когда исходный граф состоит из многих подграфов. Нужно обрабатывать подграфы в порядке убывания ширин входящих в них узлов. После n-ого шага максимальные цепи уже рассчитаны в первых n подграфах, выбирается следующий подграф, в нём строятся цепи, соединяющие всевозможные пары узлов, они комбинируются с рассчитанными ранее цепями и результат передаётся на следующий шаг.

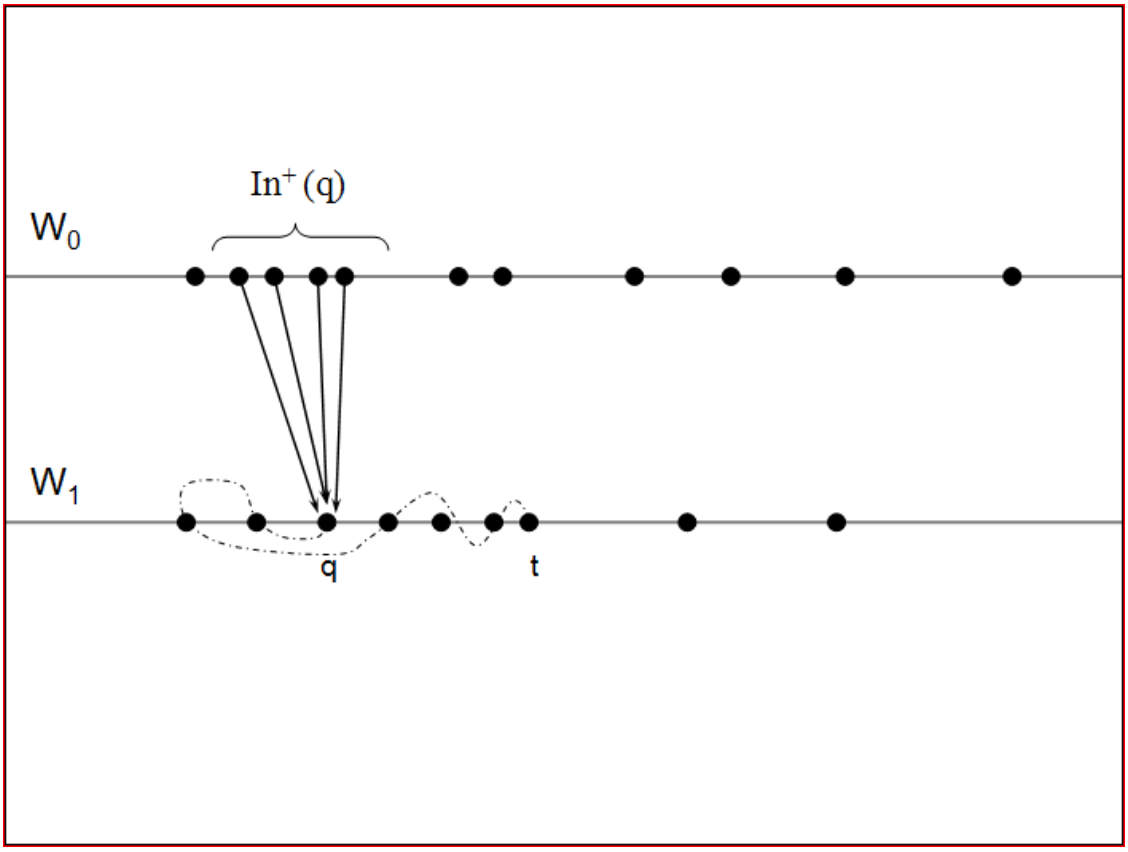

Рис. 4.2: Работа алгоритма

На рис. (4.2) изображены узлы, из которых состоят подграфы G_{w_0} и G_{w_1}.
Пунктиром изображена простая цепь H_{qt}.

4.3 Формальное описание алгоритма

Обозначим алгоритм $\mathbb{A}$ – частный алгоритм, решает задачу 2.4 в подграфе G_w. Алгоритм $\mathbb{A}$ для заданной пары узлов $s,\ t$ возвращает упорядоченный набор $H = (p_1, p_2, \ldots, p_n)$, что $s = p_1 \to p_2 \to \ldots \to p_n = t$ - максимальная по весу простая (s,t) - цепь для заданной ширины w, или сообщение, что такую цепь построить нельзя.
Обозначим:

- $w_0, w_1, \ldots, w_z$ - всевозможные значения ширины, упорядоченные по убыванию.
- $n_0, n_1, \ldots, n_z$ - количество узлов в соответствующих подграфах.

Общий алгоритм:

1. На нулевом шаге
   - $P_{w_0} := \{p_1^0, p_2^0, \ldots, p_{n_0}^0\}$.
   - $\forall r = \overline{1, n_0}, \forall k = \overline{1, n_0} : H_{rk}^0$ - максимальная по весу простая цепь, соединяющая p_r^0 и p_k^0, построенная алгоритмом $\mathbb{A}$.
   - $\forall m = \overline{1, n_0} : H[p_m^0] := \arg \max\limits_{r=\overline{1,n_0}} l(H_{rm}^0)$ - максимальная по весу простая цепь заканчивающаяся узлом p_m^0.
   - $H^0 := \arg \max\limits_{m=\overline{1,n_0}} l(H[p_m^0])$.
2. На i-ом шаге
   - $P_{w_i} := \{p_1^i, p_2^i, \ldots, p_{n_i}^i\}$.
   - $\forall r = \overline{1, n_i}, \forall k = \overline{1, n_i} : H_{rk}^i$ - максимальная по весу простая цепь из p_r^i в p_k^i, построенная алгоритмом $\mathbb{A}$.
   - $\forall r = \overline{1, n_i}$: $H_r^i := \arg \max\limits_{p \in In^+(p_r^i)} l(H[p])$ - максимальная по весу простая цепь, заканчивающаяся одним из узлов-предшественников узла p_r^i.
   - $\forall m = \overline{1, n_i}$: $H[p_m^i] := \arg \max\limits_{r=\overline{1,n_i}} l(H_r^i + H_{rm}^i)$
   - $H^i := \arg\max\{l(H^{i-1}), l(H[p_1^i]), \ldots, l(H[p_{n_i}^i])\}$.
3. $H = H^z$ - максимальная по весу простая цепь в графе G.

Теорема (1). Общий алгоритм *строит максимальную по весу простую цепь.*

Доказательство. Докажем, что для любого узла p построенная алгоритмом простая цепь $H[p]$ является максимальной по весу. Проведем доказательство методом математической индукции по номеру подграфа, в который входит узел p.

База индукции. Пусть p такой, что $w_p = w_0$. На нулевом шаге алгоритм строит максимальные по весу простые цепи в G_{w_0} в силу теоремы 1 работы [1].

Шаг индукции. Пусть $\forall i \le k$, $\forall p \in P_{w_i}$: $H[p]$ - максимальная по весу простая цепь, оканчивающаяся узлом p.
Докажем, что $\forall q \in P_{w_{k+1}} : H[q]$ - максимальная по весу простая цепь с концом в q.

Пусть $q \in P_{w_{k+1}}$, $H[q]$ - цепь, построенная по общему алгоритму.

Пусть также M - максимальная по весу простая цепь с концом в q.

Предположим, что $l(M) > l(H[q])$.

$$H[q] = p_1 \to \ldots \to p_l \to p_{l+1} \to \ldots p_s \to q \tag{4.4}$$

где $w_{p_i} > w_{p_j} = w_q$ по всем $i \in \overline{1, l}, j \in \overline{l+1, s}$.

$$M = q_1 \to \ldots \to q_m \to q_{m+1} \to \ldots \to q_t \to q \tag{4.5}$$

где $w_{q_i} > w_{q_j} = w_q$ по всем $i \in \overline{1, m}, j \in \overline{m+1, t}$.

Тогда

$$l(H[q]) = l(p_1 \ldots p_l) + l(p_{l+1} \ldots p_s) + l(q) \tag{4.6}$$

и

$$l(M) = l(q_1 \ldots q_m) + l(q_{m+1} \ldots q_t) + l(q) \tag{4.7}$$

Рассмотрим цепь M. По предположению индукции $l(H[q_m]) \geq l(q_1 \ldots q_m)$.

Пусть h - максимальная по весу простая цепь, соединяющая q_{m+1} и q, построенная алгоритмом $\mathbb{A}$.

Тогда

$$l(H[q]) \geq l(H[q_m]) + l(h) \geq \tag{4.8}$$

Неравенство 4.8 выполняется по построению $H[q]$ (т.к. перебираются всевозможные такие суммы цепей, лежащих в подграфах G_{w_i} и подграфе $G_{w_{k+1}}$).

А т.к. h - максимальна по весу, то выполняется

$$\geq l(H[q_m]) + l(q_{m+1} \ldots q_t) + l(q) \geq l(q_1 \ldots q_m) + l(q_{m+1} \ldots q_t) + l(q) = l(M) \tag{4.9}$$

Получается, что предположение не верно, и на самом деле $L(M) \leq l(H[q])$,
а т.к. M максимальная по весу, то таким образом и $H[q]$ максимальная по весу простая цепь, заканчивающаяся в q.

□

Теорема (2). *Общий алгоритм имеет сложность $O(k^3)$, где $k = |P|$.*

Доказательство. Пусть $n_i = |P_{w_i}|$, $m = |W|$. Алгоритм производит m итераций по количеству различных значений ширины узлов. При построении максимальной цепи для двух фиксированных узлов, алгоритм в худшем случае просматривает один раз отсортированный по толщине список узлов в данной ширине. То есть, на каждом шаге список узлов сортируется за $n_i \log n_i$, после чего перебираются попарно узлы (n_i^2) и для них строится максимальный путь (n) [1], и для каждого узла перебирается список его предшественников.

Тогда сложность общего алгоритма не превосходит

$$C(\sum_{i=1}^{m}(\sum_{p \in P_{w_i}} |In^+(p)| + n_i \log n_i + n_i^3)) \tag{4.10}$$

где C — некоторая константа, что в худшем случае, когда все узлы находятся в одной ширине есть $O(k^3)$. □

4.4 Оптимизация работы алгоритма. Построение графа G'

Обозначим $\bar{G}$ - граф, полученный из исходного графа G удалением дуг pq, для которых существует путь $H = p \to p' \to \ldots \to p'' \to q$ такой, что $w_p > w_{p'}$ и $w_{p''} > w_q$.

Теорема (3). *Граф $\bar{G}$ содержит все максимальные по включению узлов цепи графа G.*

Доказательство. Пусть H_m — некоторая максимальная по включению цепь в графе G. Предположим, что H_m не содержится в $\bar{G}$. То есть существует хотя бы одна дуга $pq \in H_m$, которая не содержится в $E(\bar{G})$.
Раз эта дуга была удалена из графа G, следовательно, для этой дуги существует путь $H = p \to p' \to \ldots \to p'' \to q$ в графе G,
где $w_p > w_{p'}$ и $w_{p''} > w_q$. Это условие противоречит максимальности цепи H_m. Следовательно, любая максимальная цепь графа G содержится в графе $\bar{G}$. □

Таким образом, в качестве G' можно взять граф $\bar{G}$.

Иллюстрация "чистки"графа

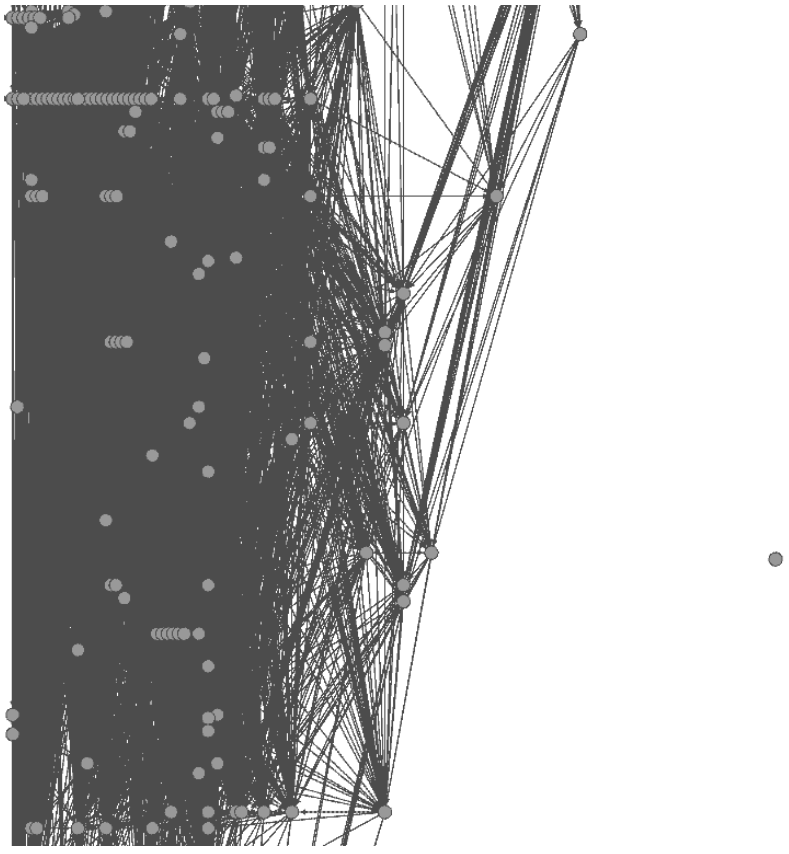

Рис. 4.3: Исходный граф G

На рис. (4.3) изображен фрагмент исходного графа, в котором порядка 340 узлов и 17000 дуг.

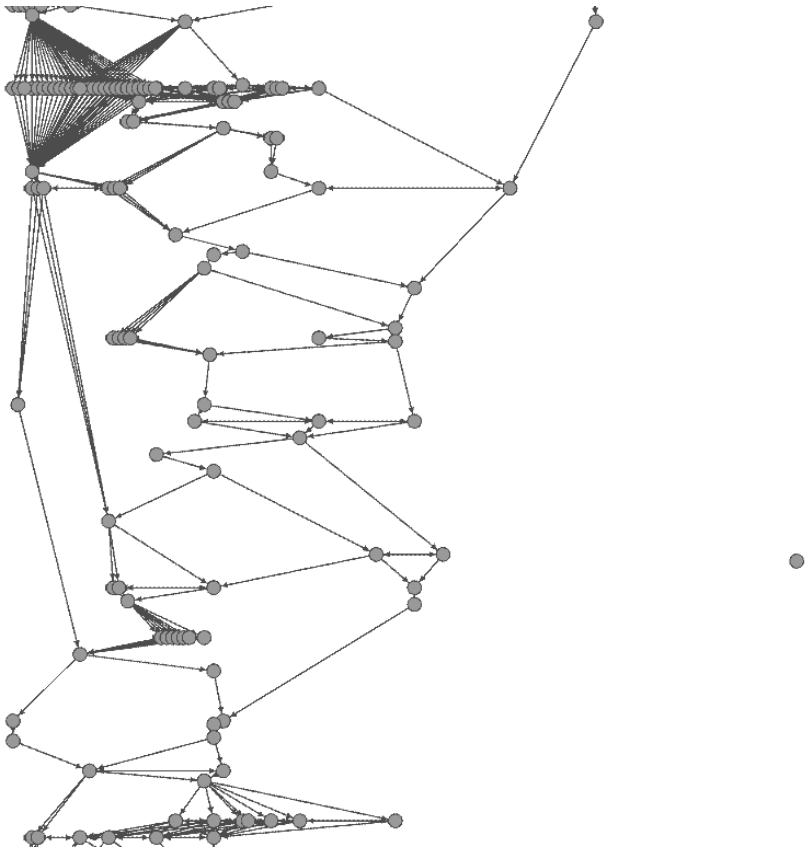

Рис. 4.4: "Чищенный"граф G'

На рис. (4.4) изображен фрагмент соответствующего "чищенного"графа.
В нем то же количество узлов, но после "чистки"остаётся порядка 3900 дуг.
Обратим внимание, что удаляются только лишь некоторые дуги, соединяющие узлы, имеющие разную ширину.
Поэтому легко придумать пример, когда построение такого графа не даст выигрыша в работе алгоритма, но на реальных данных процедура чистки оказывается эффективной.

Глава 5

Программная реализация

В работе использован язык программирования Java.

5.1 Описание кода программы

5.1.1 Типы данных

- Вводим класс для представления узлов графа:
  class Vertex { ...
  double width;
  double thickness;
  double length;
  ... }
  где width, thick - геометрические параметры партии, а length - длина (вес).

- Класс для представления ограничений:
  class Crit { ...
  double w;
  double r(double thickness) {...}
  ... }
  где число w соответствует ограничению 2.1, а метод r(double) соответствует ограничению 2.2 из главы 2.

- Класс для представления дуг графа:
  class Edge { ...
  Vertex from;
  Vertex to;
  ...}

- Создаём класс для представления графа:
  class Graph { ...
  ArrayList<Vertex> vertex;
  ArrayList<Edge> edges;
  ArrayList<ArrayList<Vertex>> pred;
  ArrayList<ArrayList<Vertex>> succ;
  ArrayList<ArrayList<Vertex>> in;

ArrayList<ArrayList<Vertex>> out;
... }
где списки in, out, pred, succ служат для представления множеств описанных в главе 2 (in соответствует множеству In^+), a length - суммарная длина (вес) цепи.

- Класс для представления результата работы алгоритма - простой цепи:
class Path { ...

  Vertex start;

  Vertex end;

  ArrayList<Vertex> path;

  double length;
... }
где в списке path хранится последовательность узлов цепи, начиная со start и заканчивая end.

5.1.2 Методы

- Основной метод решающий поставленную задачу(принимает на вход граф, возвращает простую цепь максимального веса):
Path maxPath(Graph graph, Crit crit) {

  ArrayList<Path> paths = new ArrayList<Path>(); */* объявляем массив, в котором будут строится максимальные цепи */*
  ...

  for (double w : widthList) {

    ArrayList<Vertex> layer = layer(graph, w);

    merge(graph, layer, paths, crit);

  }

  return max(paths);
}
где ArrayList<double> widthList — отсортированный по убыванию список всевозможных ширин узлов,
a ArrayList<Vertex> layer — соответствует подграфу G_w.

- Метод для конструирования подграфа G_w (принимает на вход граф и значение ширины, возвращает список узлов имеющих заданную ширину):
ArrayList<Vertex> layer(Graph, double) {...}

- Метод, производящий на каждом шаге слияние уже построенных цепей с цепями в следующем подграфе (вызывается в цикле в основном методе, результат накапливается в переменной paths):
void merge(Graph graph, ArrayList<Vertex> layer,

ArrayList<Path> paths, Crit crit) { ...

  for (Vertex fixed : layer) { */*выбираем в подграфе "фиксированную"вершину*/*

    Path fixedPath = new Path();

    for (Vertex var : layer) { */*перебираем в подграфе "свободные"вершины*/*

      */*в следующей строке используем частный алгоритм*/*

```
                Path path = Path.sortedLayerPath(layer, fixed, var, crit);
            /*выбираем подходящую цепь-продолжение для "свободной"вершины*/
                Path maxTail = chooseTail(var, paths);
                /*сравниваем с текущим претендентом на максимальную цепь,
                заканчивающуюся в узле fixed*/
                maximize(path, maxTail, fixedPath);
            }
            /*записываем результат в список уже рассчитанных цепей*/
            paths.set(fixed.id, fixedPath);
... }
```

- Path sortedLayerPath(ArrayList<Vertex>, Vertex, Vertex, Crit) {...} — реализация частного алгоритма.
- Path maximize(Path, Path, Path) {...} — конструирует сумму первых двух аргументов, возвращает максимум из суммы и третьего аргумента.

5.2 Результаты экспериментов

Данные для тестов берутся из корпоративной сети ОАО "ММК". Обычно в распоряжении имеются выборки порядка 200-300 партий.

№ Выборки	Общее количество узлов	Количество узлов в цепи	Суммарный вес выборки	Суммарный вес цепи	Среднее время работы программы (мс)		
					Чистка	После чистки	Без чистки
1	10	4	18,004	12,177	1	1	1
2	20	16	81,214	65,869	3	3	3
3	50	47	249,486	246,268	10	5	8
4	100	77	295,160	264,895	10	10	16
5	200	170	822	746,41	80	30	120
6	200	154	507,255	437,181	25	10	90
7	300	242	1007,75	899,395	130	40	320
8	344	278	1067,52	947,617	140	50	400

Табл. 5.1: Результаты тестов

Реальные данные таковы, что обычно при больших выборках от 75% процентов узлов попадают в построенную цепь. Это хороший с практической точки зрения результат, т.к. на производстве такие большие графики прокатки избыточны и построенную цепь можно дополнительно редактировать, добиваясь выполнения остальных технических ограничений, не упоминаемых в этой работе.

Так же отметим, что программа работает достаточно быстро, что важно, потому что в реальной жизни требуется многократный запуск алгоритма. Видно, что скорость работы программы довольно сильно зависит от структуры конкретных начальных данных. Имеет значение соотношение между количеством подграфов в исходном графе и количеством входящих в них узлов. Например в случае, когда исходный граф состоит из малого числа подграфов, то время работы уменьшается за счет сокращения перебора дуг между подграфами(выборка 6).

(Тесты проводились на персональном компьютере).

5.3 Инструкция пользователю

- **Входной файл** должен иметь следующий формат:
  Список партий, заданных тремя числами — шириной, толщиной и длиной (весом) партии.
  Пример:

  1635 2,1 3,288
  1630 2,1 1,099
  1630 3,9 1,237
  1630 2,1 2,199
  1620 2,1 5,531
  1615 4,5 1,151
  1610 3,9 0,626
  1610 3,9 1,252
  1610 3,9 1,252
  1600 4,0 0,369

- **Выходной файл** имеет следующий формат:
  Общий вес поданной выборки, вес построенной цепи, количество входящих в нее узлов, последовательность узлов, время работы в милисекундах.
  Пример:
  Total Weight: 18.004
  Path Weight: 12.117
  Path Size: 4
  $[[0(1635.0; 2.1; 3.288)], [1(1630.0; 2.1; 1.099)], [3(1630.0; 2.1; 2, 199)], [4(1620.0; 2.1; 5, 531)]]$
  Time: 1

Заключение

Автоматическое планирование металлургического производства является сложной задачей, которая до сих пор не реализована полностью ни на одном предприятии в мире. Сопутствующие ей математические постановки зачастую являются труднорешаемыми. За счет исследования технологических ограничений на конкретном предприятии удалось сформулировать математическую задачу планирования графика прокатки. Удалось разработать полиномиальный алгоритм её решения. Результат уже частично внедрён и продолжается внедрятся на ОАО "ММК". В реальной жизни существуют более общие и сложные постановки задачи планирования графиков прокатки, решение которых основывается на описанных в настоящей работе алгоритмах.

Литература

[1] Леонова С.И. Разработка и исследование алгоритмов построения экстремальных цепей в вершинно-взвешенных орграфах. Частный алгоритм. 2014

[2] 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.

[3] 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.

[4] E. Balas, The prize collecting travelling salesman problem, Networks 19 (1989) 621-636.

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

[6] 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.

[7] G. B. Dantzig and J. H. Ramser. Management Science, Vol. 6, No. 1 (Oct., 1959), pp. 80-91.

[8] 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.

[9] K. Ioannidou, S. D. Nikolopoulos. The Longest Path Problem is Polynomial on Cocomparability Graphs. Algorithmica. January 2013, Volume 65, Issue 1, pp 177-205.

[10] The Open Graph Viz Platform [Офиц. сайт]. URL: http://gephi.org/ (дата обращения: 01.06.2014)

Приложение

Пример исходного кода

```
public abstract class MaxPathSearch {
    public Graph graph;
    public ArrayList<Double> widthList;
    double eps = 1e-5;

    public MaxPathSearch(Graph graph) {
        this.graph = graph;
        widthList = new ArrayList<Double>();
        for (Vertex v : this.graph.vertex) {
            if (!widthList.contains(v.width)) {
                widthList.add(v.width);
            }
        }
    }

    public ArrayList<Vertex> layer(Graph graph, double width) {
        ArrayList<Vertex> result = new ArrayList<Vertex>();
        for (Vertex v : graph.vertex) {
            if (Math.abs(v.width - width) < eps) {
                result.add(v);
            }
        }
        return result;
    }

    // ищем самый длинный хвост среди тех, которые можно присоединить к var вершине
    public abstract Path chooseTail(Vertex var, ArrayList<Path> paths);
```

```
    // соединяем получившиеся части и сравниваем с текущим максимальным для fixed вершины
    public void maximize(Path path, Path maxTail, Path fixedPath) {
        // к построенной цепи присоединяем максимальный хвост
        path.join(maxTail);
        // увеличиваем максимальный путь для fixed вершины
        fixedPath.maximize(path);
    }

    public void merge(Graph graph, ArrayList<Vertex> layer, ArrayList<Path> paths, Crit crit) {
//      Path.sort(layer);
        for (Vertex fixed : layer) {
            Path fixedPath = new Path();
            for (Vertex var : layer) {
                Path path = Path.sortedLayerPath(layer, fixed, var, crit);
                Path maxTail = chooseTail(var, paths);
                maximize(path, maxTail, fixedPath);
            };
            paths.set(fixed.id, fixedPath);
        }
    }
```

```
    // общая процедура
    public Path maxPath(Graph graph, Crit crit) {
        // paths.get(i) – максимальный по длине путь, построенный из вершины с номером i
        ArrayList<Path> paths = new ArrayList<Path>();
        for (Vertex v : graph.vertex) {
            paths.add(new Path());
        }
        for (double w : widthList) {
            ArrayList<Vertex> layer = layer(graph, w);
            merge(graph, layer, paths, crit);
        }
        System.out.println("Total Weight:" + graph.totalWeight());
        return max(paths);
    }
    // выбираем максимальный по длине путь
    public Path max(ArrayList<Path> paths) {
        Path result = new Path();
        for (Path p : paths) {
            if (p.length >= result.length) {
                result = p;
            }
        }
        return result;
    }
}
```