САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра системного программирования

Мальковский Николай Владимирович

“Построение расписания занятий в автоматическом и полуавтоматическом режиме”

Дипломная работа

Допущена к защите.

Зав. кафедрой:

Терехов А.Н.

Научный руководитель:

ст. преп. Луцив Д.В

Рецензент:

Бондарев А.В.

Санкт-Петербург

2013

SAINT-PETERSBURG STATE UNIVERSITY

Mathematics&Mechanics Faculty

Software engineering Chair

Malkovskii Nikolai

Lesson scheduling in automated and semi-automated mode

Graduation Thesis

Admitted for defence.
Head of the chair:
Professor Terekhov A.N.

Scientific supervisor:
Senior Lect. Luciv D.V.

Reviewer:
Bondarev A.V.

Saint-Petersburg

2013

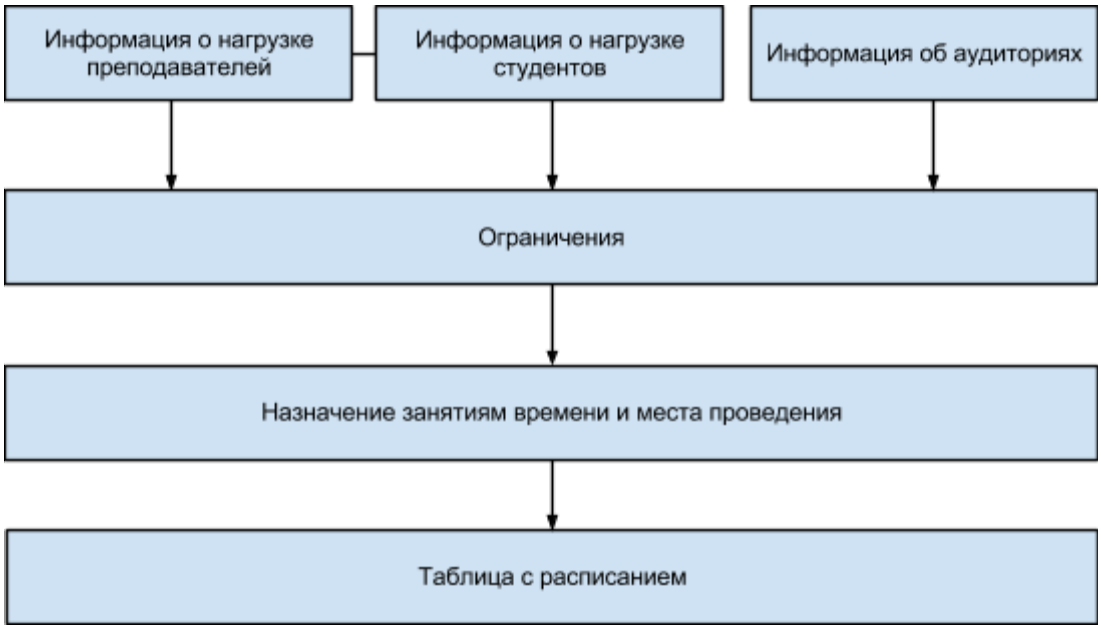
Оглавление

Введение

В современном мире все большее значение принимает автоматизация рутинных действий человека, будь то документооборот на предприятии или составление очереди в поликлинику. Данный процесс в принципе неизбежен из-за постоянно увеличивающегося количества обрабатываемой информации и свойства человека ошибаться при обработке рутинных операций.

Одним из важных процессов принятых в университете является составление расписания учебных занятий. Ручное составление расписания - занятие рутинное и трудоемкое.

Процесс составления рассписания для ВУЗа можно представить следующим образом:



На стадии распределения нагрузки студентов определяются программы курса, по которым учатся студенты. При распределении нагрузки преподавателей определяется, какие именно занятия и у кого будут проводить преподаватели. Физический сбор информации об аудиториях - выяснение различных свойств аудитория, их количества и т.п. Также при составлении рассписания, необходимо учитывать различные ограничения. Ограничения могут быть строгие и обязательно должны быть удовлетворены. Например проведение занятия по программированию в дисплейном классе, невозможность присутствия одного человека в двух местах одновременно, невозможность проведения нескольких различных занятий в одной аудитории одновременно. Некоторые ограничения могут быть нестрогие, такие как пожелания от преподавателей (например, при работе по совместительству преподаватель может себе позволить только какие-то определенные дни в неделе, или же преподавателю было бы проводить занятия в определенное время).

На следующей стадии происходит назначение занятиям времени и места проведения занятий. При этом часто возникают ситуации, когда расписания, удовлетворяющее всем ограничениям, не существует, или расписание, удовлетворяющее строгим ограничениям не является удобным на практике.

Автоматизация процесса назначения времени и аудиторий может существенно сократить затраты человеческого времени для составления расписания и исключить ошибки ручного ввода.

Моя дипломная работа посвящена изучению способов автоматизации процесса назначения времени и аудиторий занятиям, разработке эффективных структур данных для работы с расписанием.

Постановка задачи

В рамках дипломной работы ставятся следующие цели:

- Изучение существующих алгоритмов решения задачи составления расписания занятий ВУЗа
- Реализация этих алгоритмов и анализ их применимости
- Улучшение характеристик существующих алгоритмов

Обзор существующих решений и смежных задач

В научной литературе ([4], [5], [6]) рассматривались следующие подходы для решения задач связанных с составлением расписаниях:

- Эвристические подходы (Direct heuristics)
- Потоки в сетях (network flows)
- Генетические алгоритмы (Genetic algorithms)
- Метод отжига (Simulated annealing)
- Подход логического программирования (Logic programming approach)
- Метод ограничений (Constraint-based approach)
- Сведение к раскраска графа (Reduction to graph coloring)
- Линейное программирование (Linear programming)

Эвристические подходы обычно пытаются добавить занятия в расписания в специальном порядке, определяемой некоторой стратегией. Способы обработки конфликтов также определяется конкретным подходом.

Потоки в сетях часто используются как эффективные способы реализации каких-либо других задач, связанных с составлением расписания (нахождение максимального паросочетания, задача о назначениях и т. п.).

Генетические алгоритмы и метод отжига являются

оптимизационными подходами, которые обычно используются для нахождения оптимального значения некоторого функционала. К этой же категории можно отнести ршение задачи линейного программирования.

Методы ограничений и логического программирования рассматривают задачу как систему ограничений на некотором множестве, для которой нужно найти элемент, удовлетворяющий всем (или, например, как можно большему числу) ограничений.

Более подробный обзор можно найти в [4].

Далее следует более подробное описание нескольких подходов и задач, связанных с составлением расписаня.

Метод отжига

Метод отжига - рандомизированный алогоритм поиска минимума функционала. Подробный обзор различных вариантов метода отжига и сравнительный анализ можно найти в [10].

Идея метода отжига заключается в следующей модификации случайного поиска:

- Выбирается случайное начальное приближение и температура T (некоторый неотрицательный параметр алгоритма)
- На каждом шаге выбирается случайное соседнее к текущему состояние
- Если значение стало меньше, то переход от текущего состояния к новому происходит с вероятностью 1
- Если значение стало больше, то переход от текущего состояния к

новому происходит с некоторой вероятностью, уменьшающейся с уменьшением T

- Обычно вероятность перехода при в предыдущем пункте берется как $exp\{-\Delta F/T\}$
- Обычно температура уменьшается через каждые несколько шагов на некоторый фактор, т.е. $T \leftarrow T * a$

Данный подход часто применяют для в задачах оптимизации функционала в случаях, когда у функционала существует множество локальных минимумов. В этих случаях обычные градиентные методы с большой вероятностью найдут один из этих локальных минимумов.

Задача о максимальном паросочетании

Опр. Двудольным графом называется тройка $< V_1, V_2, E >$, V_1 - множество вершин первой доли, V_2 - множество вершин второй доли, $E \subseteq V_1 \times V_2$ - множество ребер.

Опр. Паросочетанием в двудольном графе называется множество ребер $M \subseteq E$, которое не содержит двух ребер, у которых есть общая вершина.

Таким образом сама задача формулируется достаточно просто: Дан двудольный граф, требуется найти в нем максимальное по размеру паросочетание.

Задача о максимальном паросочетании является одной из классической задачей теории графов. На практике(в том числе и в задачах планирования) встречаются множество вариаций этой задачи. О том, как предлагается использовать эту задачу в нашем случае разговор пойдет позже.

Дальше речь пойдет о алгоритме Куна - самом базовом алгоритме поиска паросочетание. Прежде, чем перейти к описанию самого алгоритма, важно отметить следующую вещь: если надо дополнить некоторое паросочетание до максимального, то такая задача сводится к поиску максимального паросочетания путем удаления из графа всем ребер, пересекающихся с ребрами выбранного паросочетания по вершинам.

Алгоритм Куна

Опр. Дополняющей цепью паросочетания M двудольного графа $G =< V_1,\ V_2,\ E >$ называется набор вершин $v_1,\ v_2,\ \dots,\ v_{2n},\ v_{2i} \in V_1,\ v_{2i+1} \in V_2$, $(v_i,\ v_{i+1}) \in E$, $i \neq j \Rightarrow v_i \neq v_j,\ (v_{2i},\ v_{2i+1}) \in M,\ (v_{2i-1},\ v_{2i}) \notin M$, а так же $\forall (u_1,\ u_2) \in M \quad u_1 \neq v_1, \quad u_2 \neq v_{2n}$.

Понятие дополняющей цепи является ключевым в алгоритме. У дополняющей цепи есть очевидное, но важное свойство: цепь есть путь нечетной длины, ребра с нечетными номерами(начиная с единицы) не содержатся в паросочетании, с четными - содержатся, так же ни первая, ни

последняя вершина не является концом ни одного ребра паросочетания, следовательно если заменить все четные ребра нечетными ребрами, то получится паросочетание размером на единицу больше. Фактически это означает, что паросочетания не является максимальным, если для него существует дополняющая цепь.

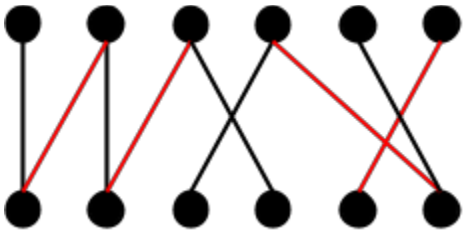


Рис. 1

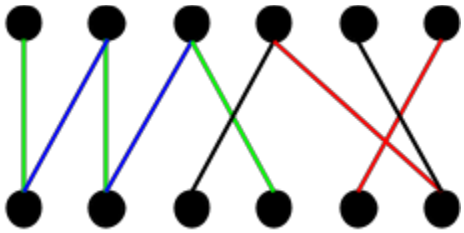


Рис. 2

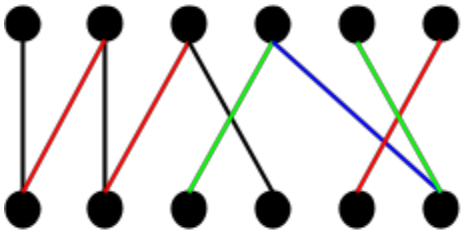


Рис. 3

На рис. 1 показано неполное паросочетание (ребра паросочетания обозначены красным), на рис. 2 и рис. 3 приведены позможные дополняющие цепи (зеленым обозначены ребро, которые включается в паросочетания, синим - ребра, которые исключаются из него).

Оказывается, что, более того, отсутствие дополняющей цепи является достаточным условием максимальности паросочетания.

теорема: Паросочетания является максимальным тогда и только тогда, когда для него не существует дополняющей цепи.

Как уже было ранее сказано, необходимость отстутствия

дополняющей цепи очевидна. Докажем достаточность.

Пусть существует паросочетание M, для которого нету дополняющей цепи. Пусть M' - максимальное паросочетание и $|M'| > |M|$. Рассмотрим $M \oplus M'$ - симетрическую разность этих паросочетаний (т.е. рассмотрим множество ребер, которые содержатся ровно в одном из паросочетаний). В силе того, что $M \oplus M'$ есть подмножество объединения двух паросочетаний, то степень вершин относительно рассматриваемых ребер не больше двух, следоветально это множество ребер разбивается на пути и циклы. Заметим, при обходе этих циклов/путей чередуются ребра из M и M', в силу того, что они являются паросочетаниями, а два подряд идущих в пути ребра не могут принадлежать одному паросочетанию. Любой цикл в двудольном графе имеет четную длину, следовательно и в $M \oplus M'$ все циклы четной длины. Если в $M \oplus M'$ есть пусть нечетной длины, то этот путь являлся бы дополняющей цепью для M или M', чего не может быть (для M по условию, для M' в силу того, что оно максимально). Таким образом полчается, что $M \oplus M'$ состоит из циклов и путей четной длины. В силу чередования ребер из M и M' мы получаем, что в каждом цикле/пути количество ребер из M и M' одинаково, следовательно, количество ребер, в которых они различаются равно, значит $|M| = |M'|$, получили противоречие.

Таким образом, напрашивается просто алгоритм построения паросочетания:

Начиная с пустого паросочетания, пока существует дополняющая цепь - обновить паросочетание.

Итак, мы практически подошли к алгоритму Куна, единственное отличие которого от описанного выше - это простое замечание: если мы не смогли в какой-то момент построить дополняющую цепь, начинающуюся в некоторой вершине, то из этой вершины мы и не найдем дополняющую цепь впоследствии. Таким образом достаточно перебрать все вершины одной доли и попытаться для каждой из них найти дополняющую цепь, начинающуюся в этой вершине. Поиск дополняющей цепи осуществляется обычным обходом в глубину или в ширину.

Пример реализации алгоритма Куна на C/C++. Первая доля имеет размер N, вторая - M.

```
int match[M];
bool visited[N];
bool c[N][M]; //Матрица смежности для двудольного графа

bool try_add(int v) {
	if(visited[v]) return false;
	visited[v] = true;
	for(int i = 0; i < M; ++i) {
		if(c[v][i] && (match[i] == -1 || try_add(match[i]) ) ) {
			match[i] = v;
			return true;
		}
	}
	return false;
```

```
}



void kuhn() {
        for(int i = 0; i < M; ++i) {
                match[i] = -1;
        }

        for(int i = 0; i < N; ++i) {
                for(int j = 0; j < N; ++j) visited[j] = false;
                try_add(i);
        }
}
```

Таким образом после вызова функции kuhn() ответ будет храниться в массиве match[M] - для каждой вершины второй доли. Если match[i] отлично от -1, значит ребро (match[i], i) содержится в паросочетании, иначе вершина i отсутствует в паросочетании.

Важно отметить следующее свойство свойство этого алгоритма: Если мы включили некоторую вершину в паросочетание, то она в обазятельно в нем останется.

Раскраска графа

Под задачей раскраски графа обычно подразумевается сразу две различные задачи:

- Покрасить вершина графа так, чтобы вершины, между которыми есть ребро, были разных цветов
- Покрасить ребра графа так, чтобы ребра, смежные по вершине, были разных цветов

Вторая задача стандартным образом сводится к первой следующим образом: строится граф, вершины которого являются ребрами исходного графа, а ребра проводятся между всеми конфликтующими парами вершин (для которых ребра в исходном графе смежны по вершине). Если исходной граф $G =< V, E >$, то получившийся граф $G' =< E, E' >$, $E' = \{(e_1, e_2) \in E | e_1 = (v, v_1), e_2 = (v, v_2), v, v_1, v_2 \in V\}$.

$$|E| = \sum_{v \in V} deg(v)$$

$$|E'| = \sum_{v \in V} \frac{deg(v)(dev(v)-1)}{2} \leq |E| \frac{maxdeg(V)-1}{2} = O(|E|^2)$$

То есть такое сведение полиномиально, но количество ребер может значительно увеличиться.

Таким образом, задаче вершинной раскраски рассматривают чаще, чем задачу реберной расскраски. Примеры использования данных задач для составления расписания рассмотрены в [4], [5], [6].

Обычно, к задачам раскраски графа сводятся задачи распределения

времени проведения некоторых событий, которые могут конфликтовать в случае их проведения в одно и тоже время (в частности, в случае построения расписания, один преподаватель не может вести два различных занятия одновременно).

Далее речь пойдет о задаче реберной раскраски двудольного графа. Рассматривается следующая задача: требуется раскрасить ребра двудольного графа $G =< V_1, V_2, E >$ в минимальное число цветов так, чтобы ребра, пересекающиеся по вершинам были раскрашены в разные цвета.

Если максимальная степень вершины в графе - K, то для раскраски понадобится как минимум K цветов. Оказывается, что K цветов будет достаточно.

Это легко показать для регулярного графа - графа, у которого степень каждой вершины одинакова. Общий случай сводится к случаю регулярного графа.

Пусть G есть K-регулярный граф, тогда по Холла [3] в нем существует совершенное паросочетание. Так как ребра паросочетания не пересекаются по вершинам, то все паросочетание можно покрасить в один цвет. Удалив это паросочетание из графа останется (K-1)-регулярный граф. Повторив операцию еще K-1 раз получим желаемый результат.

Сведем общий случай к случаю регулярного графа. Произвольный граф достраивается до K-регулярного (K - максимальная степень вершин графа) следующим образом: Добавляются вершины, чтобы в обоих долях их было одинаковое число, далее, пока существует пара вершин из разных долей со степенью меньше, чем K - добавить ребро между ними. Нетрудно

понять, что такой цикл остановится только на регулярном графе в силу того, что, при условии равного количества вершин в долях, при существовании вершины с меньшей степенью чем K в одной доле, обязательно существует аналогичная вершина в другой доле.

Теперь, решив задача для регулярного графа и отбросив фиктивные вершины и ребра, мы получаем решение для произвольного графа.

Таким образом, написанное выше можно подытожить следующей леммой:

Лемма: $G =< V_1, V_2, E >, \Delta = max\ deg(i)\ \{ i \in V_1 \cup V_2 \}$, тогда существует реберная раскраска графа G в Δ цветов.

Целочисленное линейное программирование

Под задачей линейного программирования подразумевается следующее:

$$f(x) = \sum_{i=1}^{n} c_i x_i \rightarrow min$$

$$\sum_{i=1}^{n} a_{ji} x_i \geq b_j \text{ или } \sum_{i=1}^{n} a_{ji} x_i = b_j$$

$$x_i \geq 0$$

Задача целочисленного линейного программирования является частным случаем задачи ЛП, для которой все переменные могут принимать только целые значения.

Далее приведен пример сведения задачи составления расписания к задачи ЦЛП.

Пусть $D = \{l_i\}_{i=1}^{n}$ - занятия, которые нужно провести $l_i \in D \subset P \times G \times S$

Мы хотим провести все занятия так, чтобы они не пересекались по времени и аудитории относительно преподавателя и студента. Тогда целевой функцией будет максимизация количества распределенных занятий, отсутствие конфликтов будут гарантировать ограничения.

Заведем для каждого предмета $l = (p,\ g,\ s)$ переменную $0 \le x_{pgs} \le 1$, 0 - занятие не проводится, 1 - проводится . Количество занятий, которые нужно провести известно заранее, отсюда получаем ограничение:

$$\sum_{i=1}^{n} x_i = const$$

Аудиторий может быть много, но чаще всего можно выделить несколько основных типов(Обычная аудитория, лекционная, компьютерный класс, спортзал ...). Можно считать, что каждая каждая пара должна проводиться в определенном типе аудитории.

Для каждого возможного времени проведения и типа аудитории заведем переменную $0 \le y_{tr} \le 1$, 0 - в это время аудитория свободна, 1 - занята.

Наконец можно добавить для каждого возможного занятия x_{pgs} и времени t (тут можно учитывать можелания преподавателей/студентов) переменную $0 \le z_{pgst} \le 1$. Оставшиеся ограничения накладываются естестенным образом

Для каждого типа аудитории известно кол-во таких аудиторий:

$$\sum_t y_{tr} \le M_r$$

Зависимость между переменными x, y, z:

$$x_{pgs} = \sum_t z_{pgst}$$

$$y_{tr} = \sum_{pgs:r} z_{pgst}$$

С помощью целевой функции можно минимизировать пожелания преподавателей/студентов по времени/аудитории проведения занятий. К примеру, функционал, минимизирующий кол-во окон в данной задаче можно сделать следующим образом:

Для каждого преподавателя/группу студентов и день: $max_t(tz_{pgst})$ - время, когда последнего занятия, аналогично, время первого занятия: $min_t(tz_{pgst})$. Просуммировав по каждому дню разность max-min, результат будет равено числу проведенных занятий + число окон. Таким образом, в силу ограничений, число проведенных занятий есть константа, значит минимизация такого функионала даст минимизацию окон.

Так же, ручное изменение расписание равносильно присвоению определенных значений переменным z, при подстановке которых задача все еще остается задачей ЦЛП.

Основная часть

Формальная модель

Для составления расписания нужна информация о нагрузке преподавателей (или эквивалентная ей) и доступных аудиториях. После получения этой информации в каком-либо виде, нужно распределить занятия по времени и аудиториям. Таким образом, мы получаем на вход в каком-либо виде информацию о нагрузке и информацию о аудиториях. Далее, распределяются уже конкретные пары, которые нужно провести (пара есть преподаватель, группа студентов и наименование дисциплины и требуемый тип аудитории). нужно распределить занятия по неделе (или какому-либо другому циклу) так, чтобы ни одному преподавателю или студенту не пришлось находиться одновременно на двух занятиях. Это распределение и предлагается автоматизировать. Далее более формально описана задача.

P - множество преподавателей
G - множество групп студентов
S - множество изучаемых предметов
T - множество рабочих сессий(пар) в течении одного цикла(недели)
R - множество аудиторий

Задача заключается в реализации следующей системы:

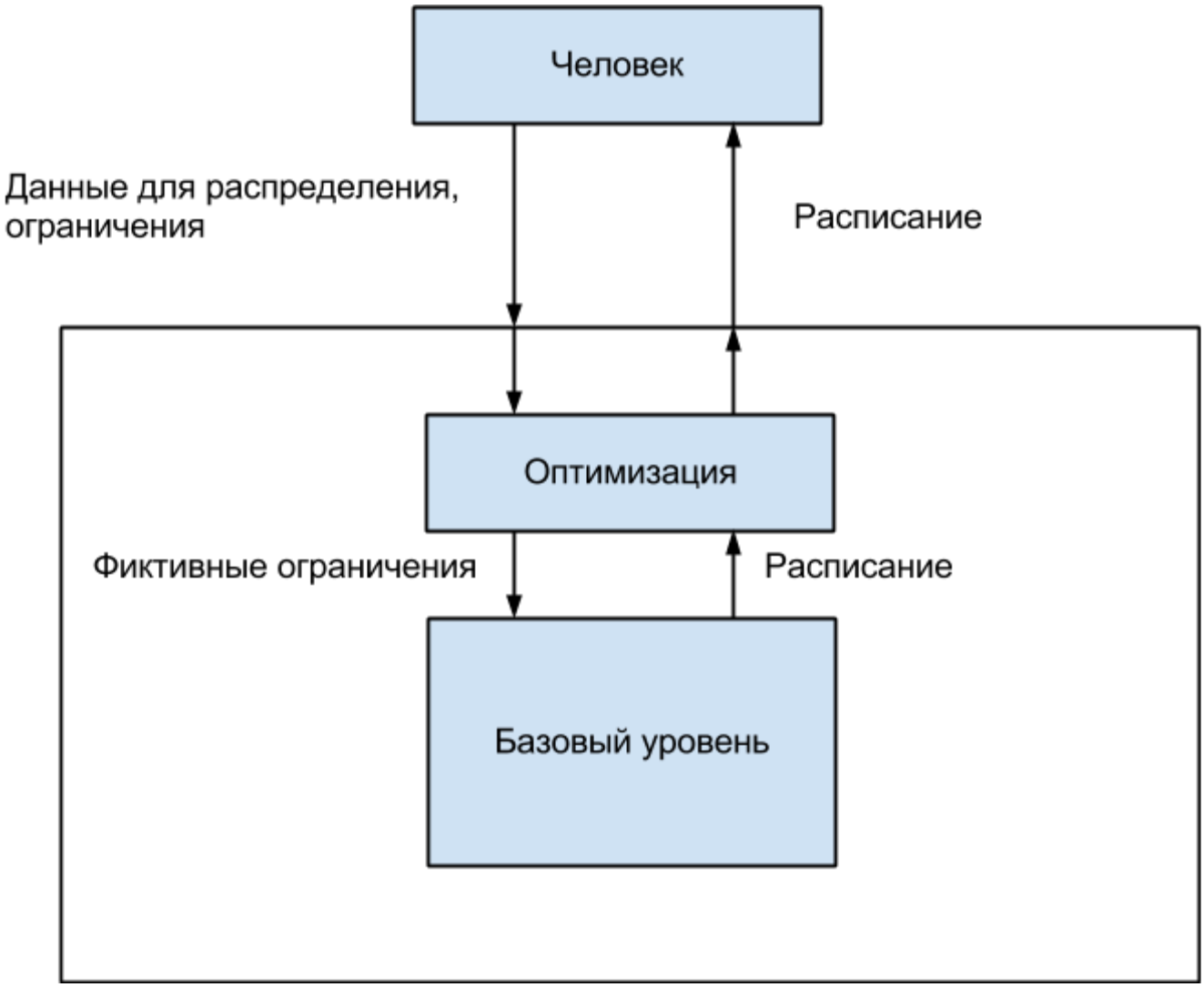
- Входные данные
    - Список всех занятий для распределения - множество $D \subseteq P \times G \times S$
    - Ограничения
- Выходные данные
    - Отображение $f : D \to T \times R$, удовлетворяющее следующим свойствам
        - f инъективно
        - $f_T(\cdot, g, s)$ инъективно
        - $f_T(p, \cdot, s)$ инъективно

Получение такого отображения и есть процесс, который нужно автоматизировать. Обычно получение какого-то расписания не является достаточным результатом, в силу того, что такое расписание может быть достаточно неудобным, хоть и удовлетворяет формальным требованиям. Таким образом вводиться некоторые функционал на множестве расписаний, а задача ставится, как нахождение расписания, минимизирующего этот функционал.

Все рассматриваемые подходы можно разделить на две группы:

- Базовые алгоритмы
    - Паросочетания и потоки
    - Раскраска ребер двудольного графа
- Оптимизирующие эвристические алгоритмы
    - Рандомизированные алгоритмы
    - Генетические методы

Саму структуру можно описать следующим образом:



Базовый уровень

В случае, если все аудитории одинаковые и их достаточно много, чтобы уместились одновременно все классы, а любое занятие проводится ровно одним преподавателем и одним классом, то составление какого-то расписания заключается в нахождении раскраски следующего двудольного графа:

- Вершины первой доли - преподаватели
- Вершины второй доли - группы учащихся
- Ребра - занятия, которые надо провести
- Каждый цвет соответствует определенному времени

Таким образом, найдя реберную раскраску этого графа мы гарантируем, что любое время у каждого преподавателя и каждой группы студентов будет не более одной пары(урока).

Известно, что задача нахождения минимальной реберной раскраски двудольного графа полиномиально разрешима, более того, точно известно, что минимальное число цветов, за которое можно раскрасить двудольный граф есть максимальная степень вершины. Однако стоит отметить, что данное утверждение верно только при условии, что все ребра можно красить в любой цвет(в соответствующей главе этот вопрос будет описан более подробно). В нашем же случае действия, совершенные человеком являются начальными данными для этой задачи, т.е. цвет некоторых ребер заранее известен. Так же стоит отметить, могут быть ограничения вида: “Ребро E нельзя красить в цвет C”, что интерпретируется как “Занятие нельзя ставить в это время”. Обычно такие ограничения возникают из пожеланий преподавателей или же невозможностью провести занятие в

данное время. Важно, что, в случае если такие ограничения исходят от преподавателей(вершин), то они распространяются на все смежные занятия(ребра). Так же, нельзя забывать, здесь еще не учитывается ограниченность аудиторий. Если считать, что для каждого занятия нужен какой-то определенный тип аудитории, то все занятия разбиваются на классы эквивалентности по типу аудитории, и добавляются условия вида “Из данного подмножество ребер нельзя красить больше N в один цвет”. В [5], [7] описано использование такого подхода для построения расписания, в [8] доказано, что с такими ограничениями задача становится NP-полной, тем не менее на практике применимы различные эвристические алгоритмы. Чуть дальше описаны предложены алгоритмы для построения раскраски графа с данными ограничениями.

Далее следует описание предложенного способа распределения в случае наличия пар для нескольких групп или при наличии разделения по группам. Данных подход является эвристическим и основан на некоторых особенностях используемых алгоритмов раскраски графа. Предполагается, что эвристика “красить ребро в цвет, в котором меньше всего заинтересованы другие вершины” дает хорошую гарантию для следующего подхода получить положительный результат.

В большинстве случаев можно считать, что потоковые лекции/разбиения на подгруппы задают некоторое подмножество подмножеств групп, в котором любые два множества либо не пересекается, либо одно содержит другое. Более формально:

$D \subseteq 2^V$, где V - множество групп учащихся. D - множество разбиений групп.

$\forall A, B \in D \;\; A \subseteq B \; B \in A$ или же $A \cap B = \varnothing$

Тогда для произвольного множества $A \subseteq D$, в котором все попарные пересечения пусты, можно составить двудольный аналогичный двудольный граф, где левая доля - преподавателя, а правая - элементы A, ребра, как и раньше, - пары, которые нужно провести. Опять же, нахождение без-конфликтного расписания для данного множества занятий эквивалентно нахождению реберной раскраски в получившемся графе.

Теперь, рассмотрим следующий орграф:

- вершины - элементы D
- Есть ребро $A \rightarrow B \Leftrightarrow A \supseteq B$

Полученный орграф является корневым деревом. Легко понять, что любое множество вершин этого графа, которое не содержит одновременно вершину и ее предка, не содержит пересекающихся множеств. Тогда расписание можно попытаться составить следующим алгоритмом:

q – очередь

$q \leftarrow root$

Пока q не пусто {

$q' \Leftarrow q$

$q \leftarrow \varnothing$

Найти расписание для занятий на разбиениях в q, не конфликтующее с уже найденным

Для всех $A \leftarrow q'$ {

Для всех $B : \exists A \rightarrow B$ {

$q \leftarrow B$

}

}

}

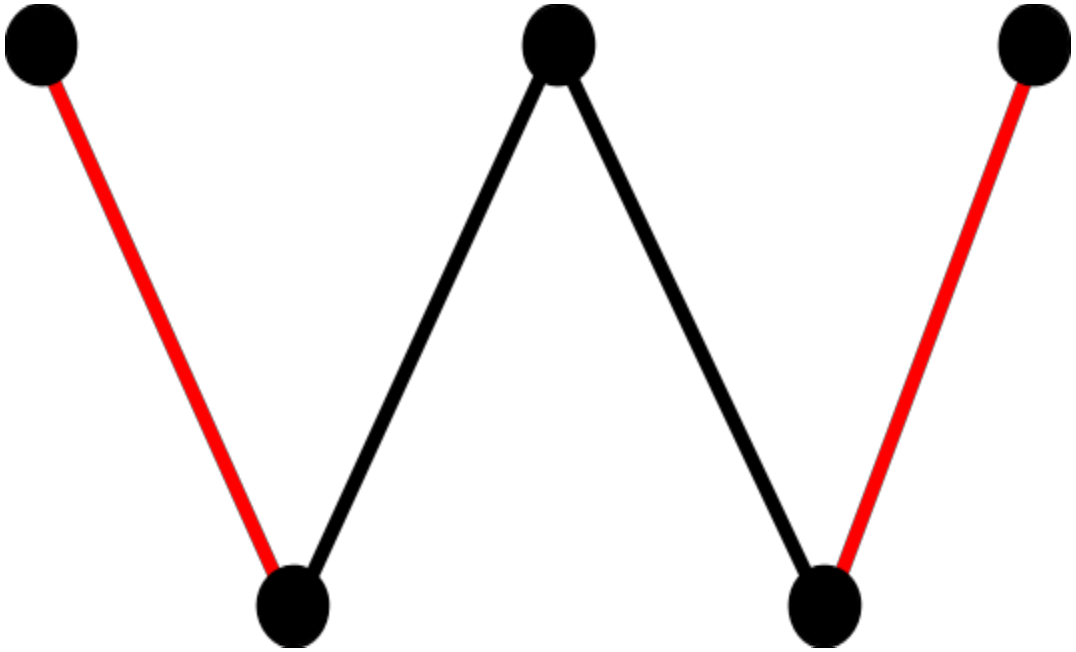
По сути этот алгоритм просто проходит по слоям дерева и достраивает расписание используя занятия для групп на этом уровне. На практике получается, что высота такого дерева не больше 3. Проблема здесь заключается в том, что нет никакой гарантии, что при переходе на следующий уровень мы все еще сможем составить полное расписание, учитывая то, что занятия на верхних уровнях мы уже зафиксировали. Как уже было сказано ранее, использование различных эвристик дает хорошую гарантию нахождения раскраски.

Алгоритмы реберной раскраски двудольного графа

На данном этапе использовались в основном алгоритмы на графах с некоторыми эвристическими подходами. Как уже было описано, в простейших случаях задача получения безконфликтного расписания сводится к реберной раскраске двудольного графа, что служит хорошой базой для построения расписания в более общих случаях. Основной проблемой при реализации каких-либо алгоритмов реберной раскраски было то, что большинство таких алгоритмов решают не совсем ту задачу, а именно: дан двудольный граф - нужно раскрасить ребра графа за минимальное число цветов так, чтобы ребра, смежные по вершине были

разных цветов. Действительно, в нашей задаче тоже нужно такое же ограничение на цвета, однако так же важно то, что некоторые ребра уже могут быть покрашены(что соответсвтует какому-то внешнему влиянию на программу). Большинство известных алгоритмов реберной раскраски нацелены на эффективность решения исходной задачи, что, к сожаления, слабо приближает к нахождению раскраски при наличии уже покрашенных ребер.

Известно, что минимальное число цветов, за которое можно раскрасить ребра произвольного графа это либо K, либо K+1, где K - максимальная степень вершины (т. Визинга [9]), для двудольных графов это всегда K. Однако, при наличии уже покрашенных ребер, уже может быть невозможным покраска за максимальную степень, как к примеру на следующем рисунке.



При наличии ограничений вида “Ребро E нельзя красить в цвет C” и “Ребро E должно быть цвета C”, задача раскраски за минимальное число цветов (или даже за заданное число цветов) становится NP-полной (доказательство можно найти в [8]). Тем не менее модификации алгоритмов, работающих без этих ограничений могут дать хорошие приближения. Стоит отметить, что обычно ограничения вида “Ребро E нельзя красить в цвет C” заменяется на “Вершина V не может иметь смежных ребер цвета C”, в предположении, что если пару нельзя ставить в определенное время, то это связано либо с группой, либо с преподавателем, значит нельзя в это время ставить никакие пары этого преподавателя/группы.

Далее описаны два основных алгоритма, которые были использованы для построения раскраски.

Оба алгоритма являются конструктивными, то есть по очереди добавляют ребра в раскраску, возможно, перекрашивая некоторые ребра. Добавление заключается в нахождении дополняющей цепи. В первом случае, мы добавляем новое ребро, крася в его в цвет, которые еще не использован для соответствующей вершины первой доли. В таком случае может произойти конфликт, если для соответствующей вершины второй доли уже есть ребро покрашенное в этот цвет. В этом случае это это ребро снова становится неокрашенным, и мы пытаемся покрасить его в другой цвет. Это реализуется с помощью процедуры, подобной поиску дополняющей цепи в алгоритме Куна.

```
bool try_add(int v, int color) {
        if(visited[v][color]) return false;
```

```
        visited[v][color] = true;
        for(int c = 0; c < COLORS; ++c) {
            for(int i = 0; i < M; ++i) {
                if(c[v][i] && (match[i][color] == -1 ||
                        try_add(match[i], c) ) ) {
                    match[i][color] = v;
                    return true;
                }
            }
        }
        return false;
    }
```

В данном случае в массиве match[i][color] записан номер ребра инцидентного к вершине i, покрашенного в цвет color. Этот алгоритм похож на алгоритм покраски, описанный в [1]. В [1] показано, что при отсутствии ограничений достаточно перекраски ребер каких-то двух цветов, в то время как ребра остальных цветов останутся нетронутыми.

Второй предлагаемый алгоритм немного слабее, чем предыдущем в том смысле, что, если, второй алгоритм найдет расписание, то первый тоже должен его найти. Отличие второго алгоритма заключается в том, что при попытке покрасить ребро в определенный цвет, он пытается добавить это ребро в соответствующее паросочетания, используя только непокрашенные ребра, в то время, как у предыдущего алгоритма остается возможность отправить ребро в другое паросочетание (соответствующее другому цвету).

Данное ограничение дает алгоритму следующее свойство: если в какой-то момент времени алгоритм назначил ребру, смежному к некоторой вершине v, определенный цвет, то в конце работы алгоритма, у этой вершины всегда будет смежное ребро этого цвета. Важно это свойство из следующего соображения: Пусть мы как-то покрасили граф, а теперь мы хотим его как-то изменить. После этих изменений могли произойти какие-то конфликты, которые мы уладили удалением ребра из раскраски. Теперь нужно достроить раскраску до полной, что делается с помозью последнего алгоритма. Тогда полученная раскраска будет мало отличаться от той, которая была в том смысле, что набор цветов ребер, смежных к вершинам изменится на столько же, сколько произошло изменений в самом расписании. Перестановка по времени проведения в данном случае остается на уровень оптимизации, тем не менее, данный подход нецелесообразно использовать без каких-либо эвристик. Описанный выше рисунок показывает, как такой алгоритм может повести себя глупо в простой ситуации.

Эвристики для построение реберной раскраски

В конструктивных алгоритмах построения расписания, описанных выше есть два основных подхода применения эвристик:

- Порядок добавления вершин/ребер в раскраску
- Приоритет цветов

По сути, оба предложенных ранее алгоритма по очереди пытаются

расширить кол-во покаршенных ребер, пытаясь покрасить новое ребро в какой-то цвет, разрешив возможные конфликты. Порядок, в котором происходит попытка добавления ребер в раскраску, и способок, которым выбирается цвет могут сильно повлиять на то, будет ли граф полностью раскрашен.

Мною были предложены следующие стратегии:

- Первыми обрабатываются вершины с большей плотностью
- В первую очередь назначается цвет, в котором меньше заинтересованы другие вершины

Теперь более подробно про эти стратегии. Под плотностью в данном случае подразумевается следующая величина: пусть всего доступно C цветов. Для вершины v с учетом ограничений есть C_V цветов, в которые можно красить смежные с вершиной v ребра, а так же, учитывая уже покрашенные ребра, у этой же вершины остается E_v смежных ей вершин, которые нужно раскрасить. Тогда плотность есть величина $P_v = \frac{E_V}{C_v}$. Чем больше эта величина, тем меньше свободы покраски ребер смежных с этой вершиной. Если плотность больше единицы, то покрасить все ребра уже невозможно.

Далее, рассмотрим величины C_E - количество цветов доступных ребру E. Будем считать, что ребро будет покрашено равновероятно в любой доступный ей цвет. Тогда вероятность того, что ребро E будет покрашено в цвет C есть $p_{CE} = \frac{1}{C_E}$, тогда, просуммировав по всем ребрам, пересекающимся по вершине с ребром E мы получим интересующую нас величину, которую можно интерпретировать следующим образом:

насколько станет хуже остальным, если мы покрасим ребро в этот цвет. Понятно, что желательно выбирать цвет, в который пытаемся покрасить это ребро, то желательно взять такой, для которого это величина поменьше.

Оптимизация и улучшение расписания

На данном этапе предлагается оптимизация полученного на базовом уровне расписания, путем изменения начальных данных. По сути, раз в качестве начальных данных мы учитываем уже распределенные занятия, то можно эти условия дополнять и пытаться решить задачу с этими фиктивными ограничениями. Важно отметить, что здесь используется факт того, что при небольших изменениях в построенном расписание, перестроить расписание можно намного быстрее, чем составлять его с нуля. К примеру, если мы составили расписание, а теперь хотим поставить какой-то занятие в другой время, то после такой замены, могут возникнуть конечное числе конфликтов(обычно не больше 2х - для преподавателя и группы, но групп может быть много), тогда, если просто убрать конфликтующие занятия, то их надо будет куда-нибудь поставить, что делается за несколько поисков дополняющей цепи.

Формально, данный этап можно характеризовать, как поиск минимума функционала качества расписания, получаемого с помощью базовых алгоритмов, на множестве различных начальных данных, вычисляемого на базовом уровне. Понятно, что начальные данные вообще говоря являются

частью расписания, более того начальные данный могут дать готовое расписание. Обычно, для функционала можно оценить оптимальное значение(все пожелания удовлетворены, нету окон и т.п.), но далеко не всегда существует такое “идеальное” расписание, а так же, “оптимальное” расписание для данных начальных данных вычисляется неточно(в качестве приближения берется расписания, которое вычисляется на базовом уровне).

Для поиска минимума в данных условиях были использованы различные рандомизированные подходы (в основном метод отжига), что давало хорошее улучшение расписания, полученного на базовом уровне.

Особенности реализации метода отжига

Общий вид метода отжига был описан в обзоре. В качестве целевой функции использовались различные варианты оценки негодности расписания, которые составлялись из нестрогих ограничений. Обычно такая функция являлась линейной комбинацией таких факторов как количество окон, количество нераспределенных занятий, количество дней, в которых у преподавателя/студента только одна пара и т.п. Выбор соседнего расписания происходило путем назначения случайного времени одному случайному занятию. При возникновении конфликтов после такого назначения, конфликтующие занятия удаляются из расписания.

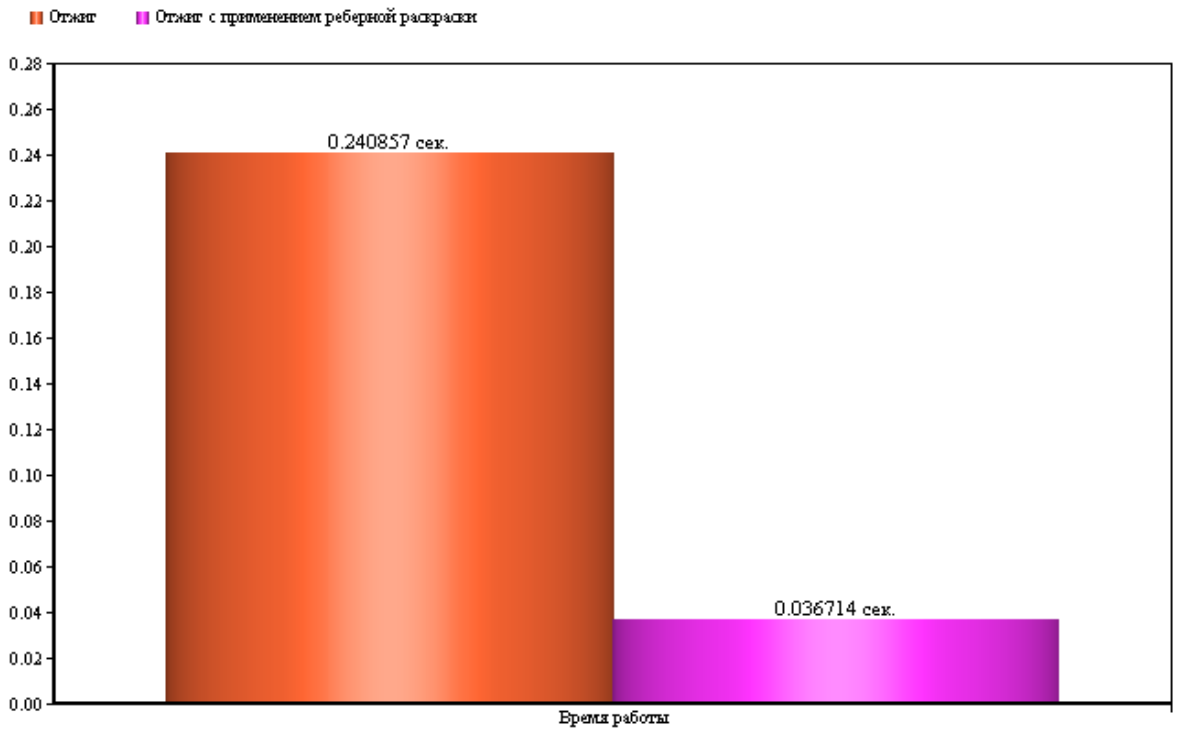
Тестирование

В основном тестирование проводилось для случайно

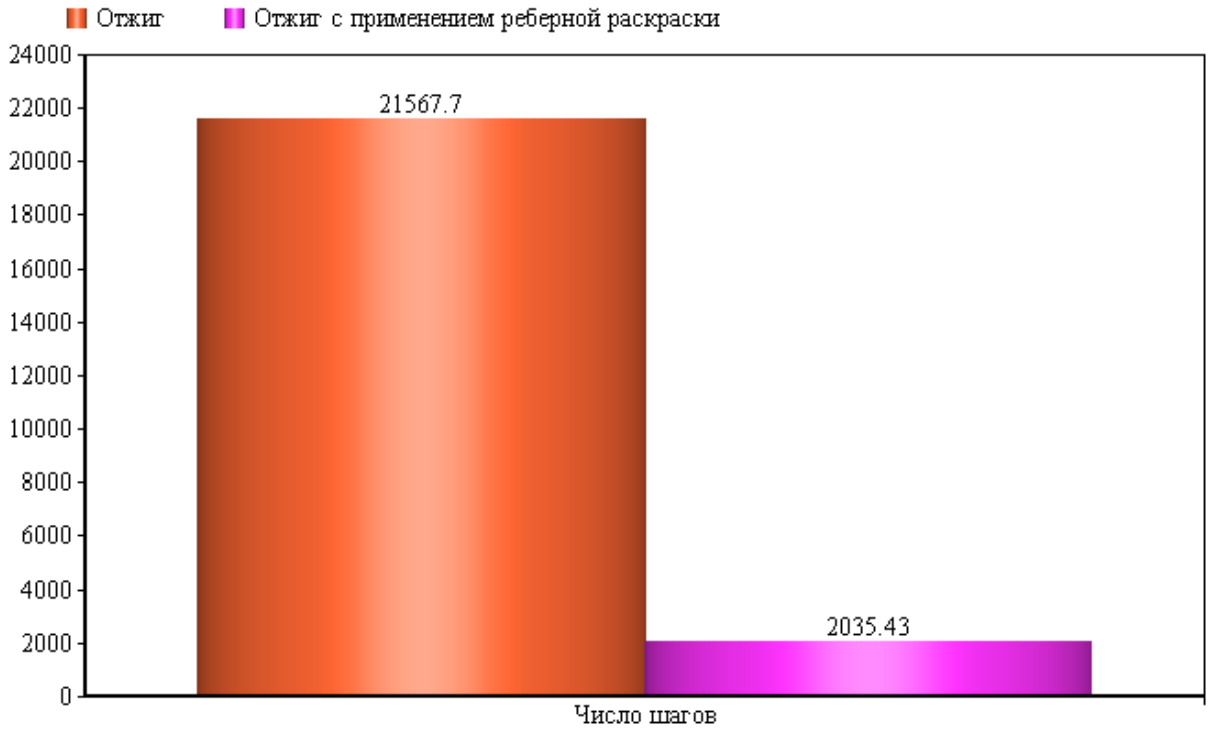
сгенерированных нагрузок по размеру данных эквивалентых нагрузке среднестатистической старшей школы/одному или нескольким курсам одного факультета СПбГУ. Тестирование проводилось для собственных реализаций методов случайного поиска, метода отжига и метода отжига с применением алгоритмов реберной раскраски двудольного графа. Реализация этих подходов могут быть далеко не самыми удачными, тем не менее все реализации были схожи: Реализация метода отжига заключалась добавлении возможности принять худший результат в метод случайного поиска. Опять же, реализация метода отжига в случае с использованием с алгоритмами реберной раскраски в точности совпадала с реализацией тестируемова метода отжига, только с применением алгоритма реберной раскраски на каждом шаге при поиске соседнего расписания. Тестирование выявило, что использование метода отжига вместо обычного случайного поиска оправдано (т.е. рассматриваемые функционалы имели множество локальных минимумов) и, что самое главное, предложенные алгоритмы решения задачи реберной раскраски двудольного графа с ограничениями на цвета и их реализация значительно ускоряет сходимость метода отжига как по количеству итераций, так и по времени работы.

Далее представлены результаты работы программы на трех примерах. Во всех примерах производилось распределение занятий 24 возможным временам проведения занятий - 6 дней в неделю, 4 пары в день, каждое занятие занимало ровно пару. Средняя нагрузка для групп студентов ~16 пар в неделю, для преподавателей ~8. В первом примере проводилось распределение 266 занятий у 32 преподавателей для 17 групп, функция минимизации - количество нераспределенных занятий + 10 * (количество окон). На следующих диаграммах показано среднее время для получения

оптимального значения 0 этой функции.

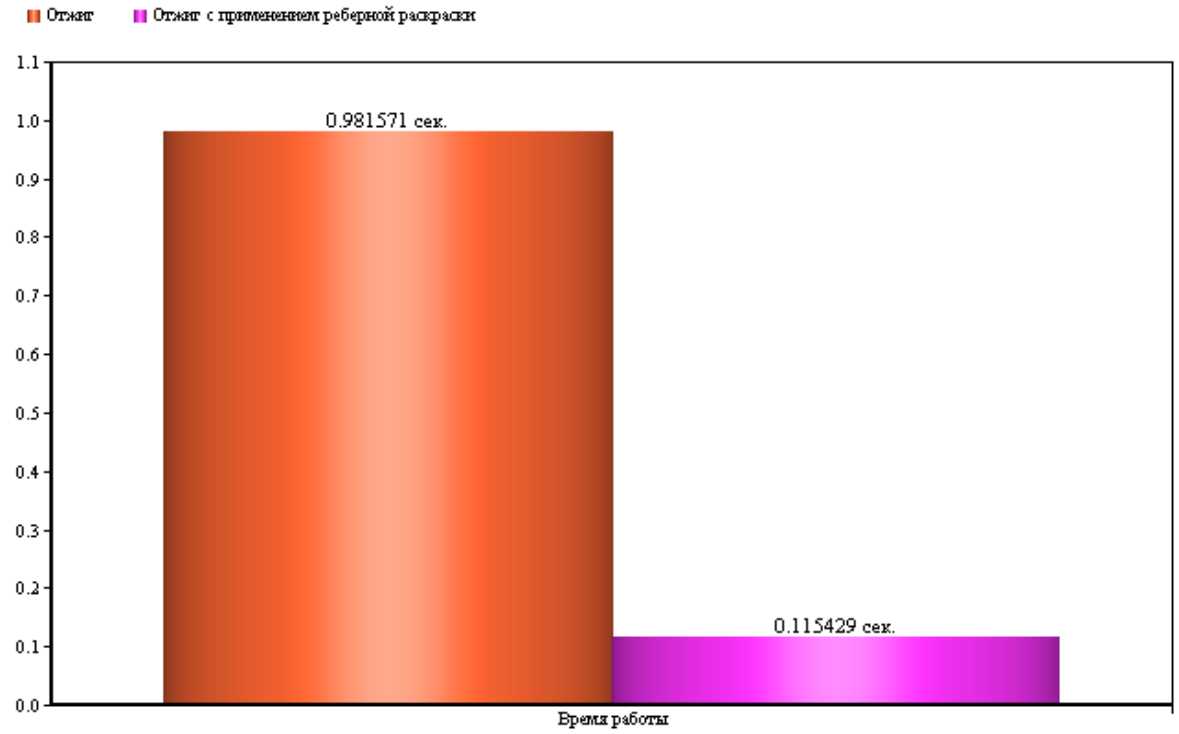


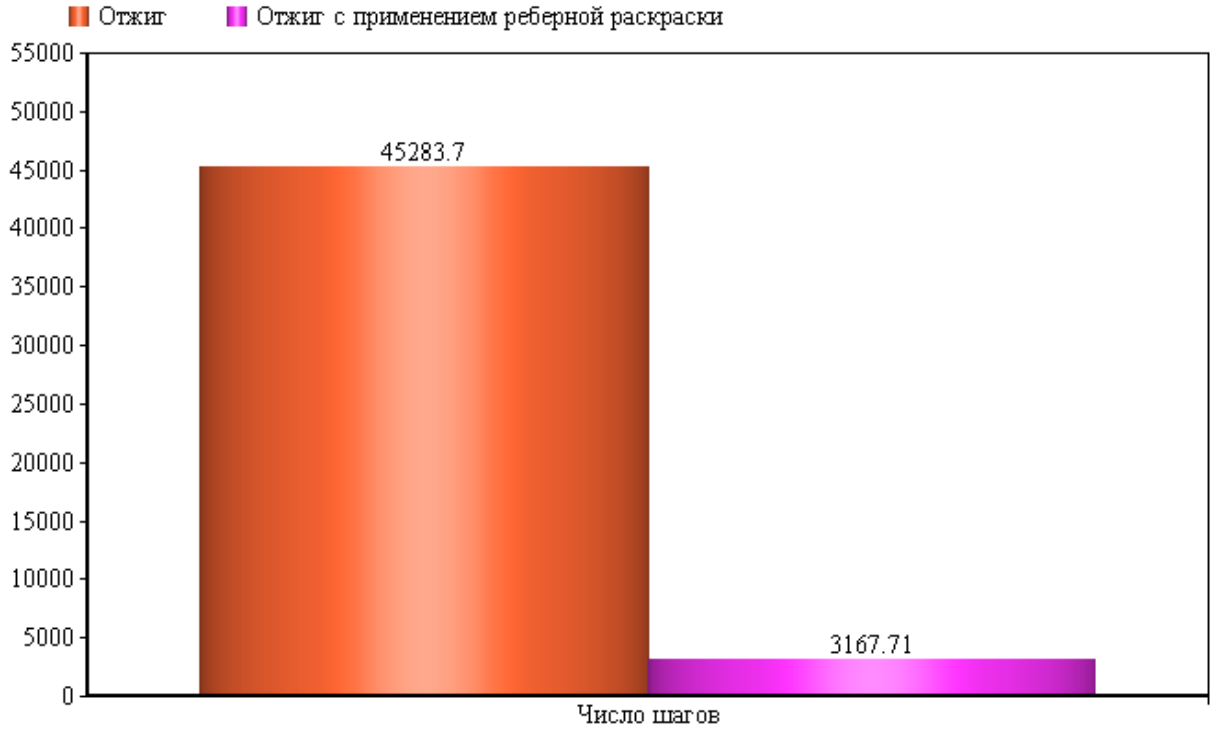
Метод случайного поиска обычно не находил оптимального результата в 20-30% случаев и застревал в локальном минимуме. Тем не менне, когда сходился, то делал это немного быстрее метода отжига. На следующей диаграмме показано среднее число шагов метода отжига до получения оптимального результата.



Уменьшение числа шагов оказывается более существенным, чем улучшение производительности в целом, что довольно естественно, так как в случае использования алгоритмов реберной раскраски итерация метода отжига работает дольше.

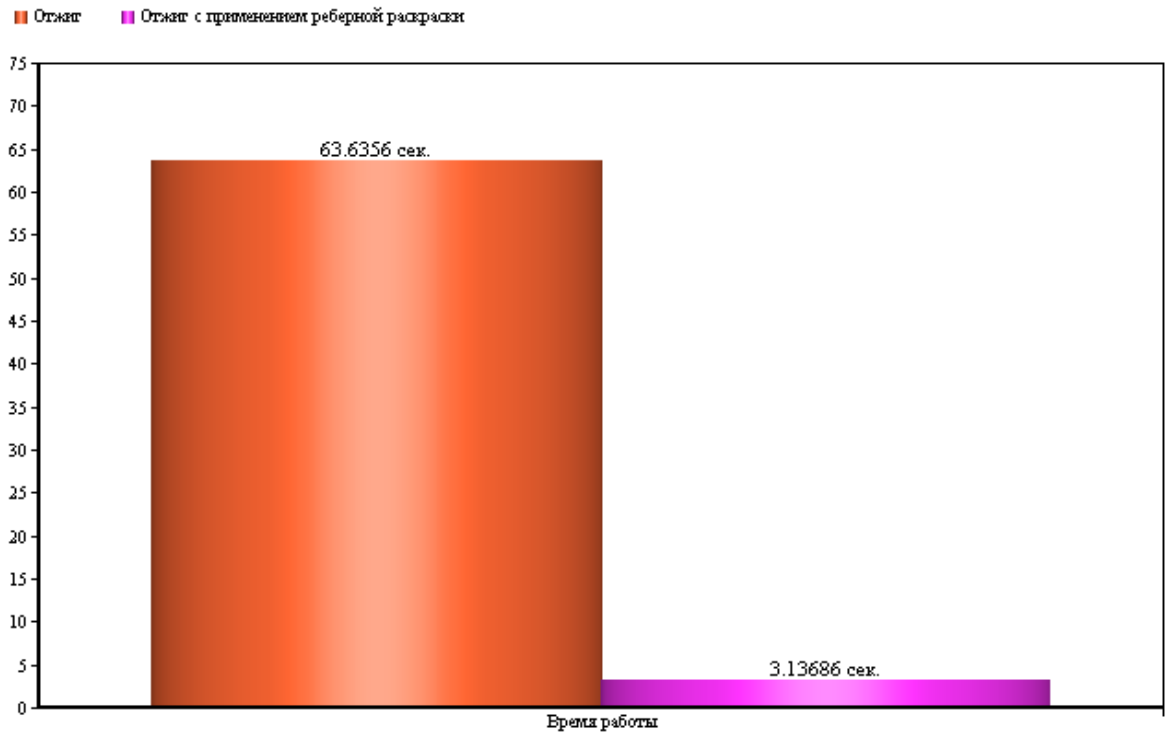
На следующих двух диаграммах показаны аналогичные результаты для второго примера входных данных. На этот раз распределяется 577 занятий, количество групп и преподавателей удваивается по сравнению с предыдущим примером, средняя нагрузка немного возрастает. Функция минимизации остается прежней.

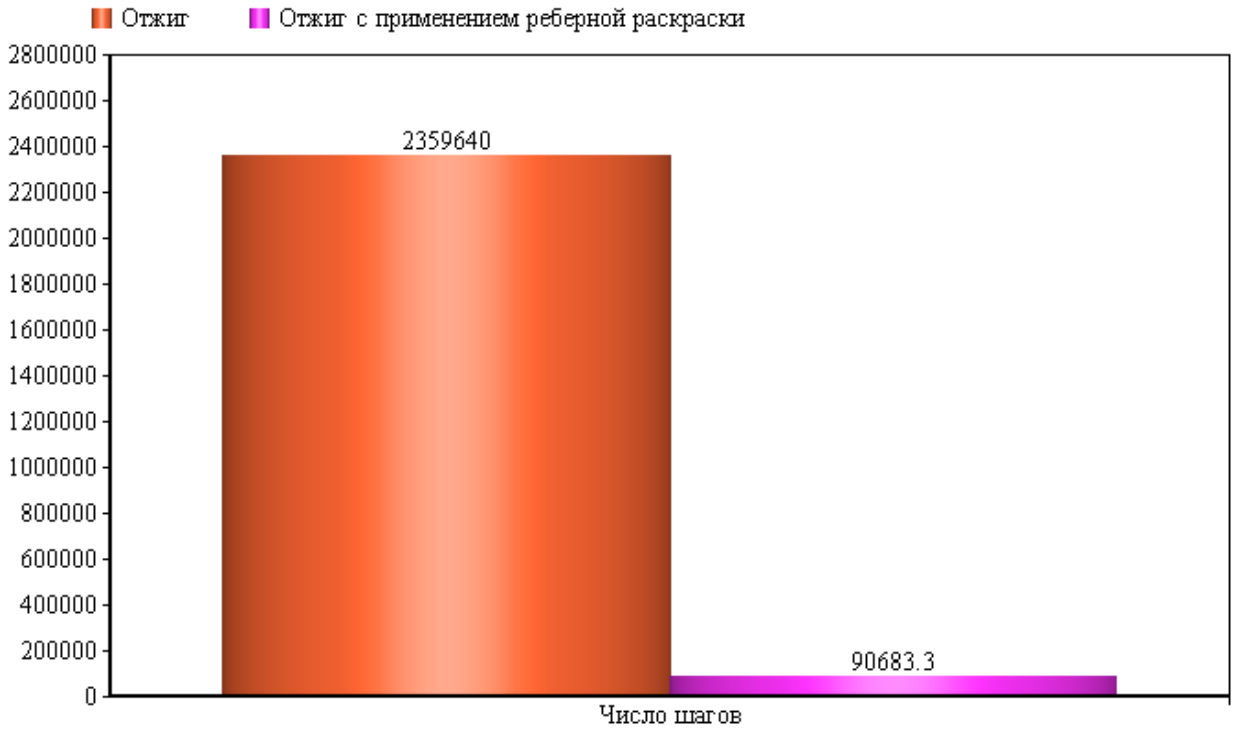

Отжиг
Отжиг с применением реберной раскраски
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.981571 сек.
0.115429 сек.
Время работы


Отжиг
Отжиг с применением реберной раскраски
55000
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
45283.7
3167.71
Число шагов

В сравнении с предыдущем примером, улучшение производительности оказалось несколько лучше, из чего можно предположить, что, в том числе, имеет место улучшение асимптотического характера. Если в предыдущем примере метод случайного поиска во многих случаях не натыкался на локальный минимум, то в данном случае он не находил оптимума почти никогда.

В следующем примере входные данные были такие же, как и в этом, но в функцию минимизации добавилось слагаемое “количество дней, в которых только одно занятие”. Грубой оценкой минимума для такой функции было количество групп + количество преподавателей, у которых одно занятие в неделю, и такая оценка достигалась. Опять же метод случайного поиска оптимума не находил, застревая на локальных минимумах, в которых часто некоторые занятия оставались нераспределенными.

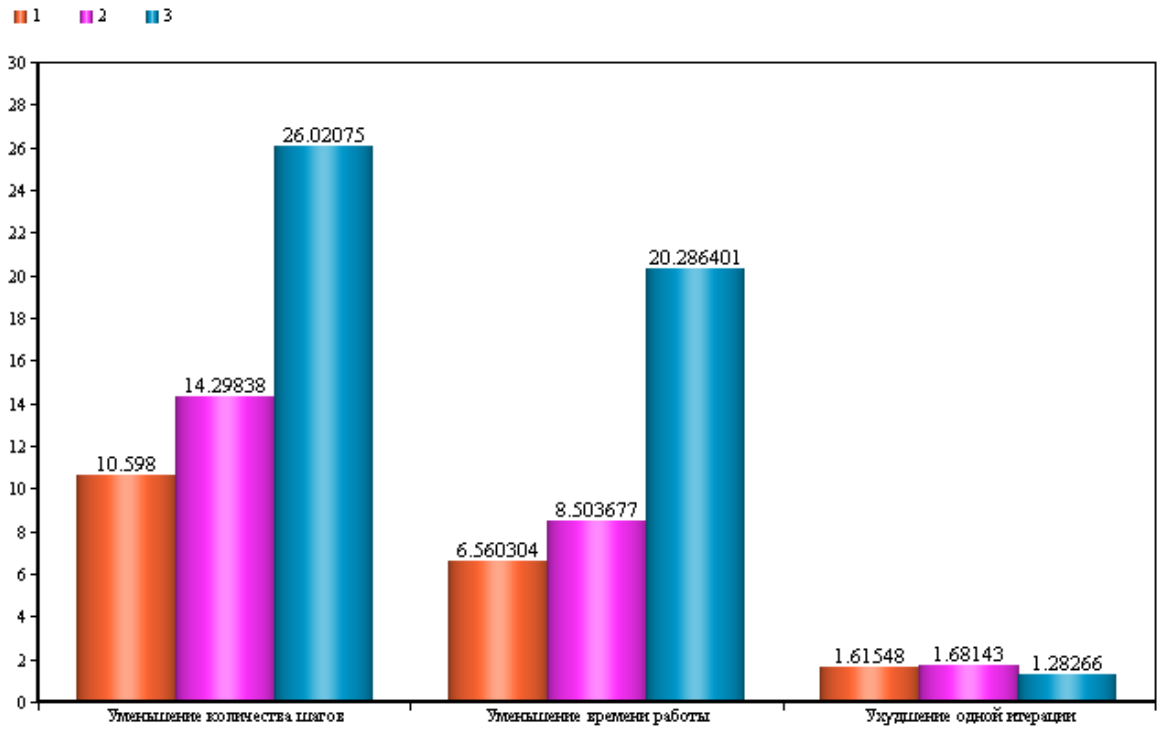




Как видно из результатов, время работы и число шагов заметно в обоих случаях, тем не менее приращение производительности, напротив, улучшилось.

Важно отметить следующее: на каждой итерации метод отжига проводит две операции, затраты времени на которые линейны относительно размера расписания - копирование и подсчет функционала. Таким образом на одной итерации метода отжига всегда будут как минимум линейные по сравнению с размером расписания затраты, которых вообще говоря достаточно. Тем не менее раскраска не слишком ухудшает время выполнения одной итерации, несмотря на то, что используемый алгоритм далеко не линейный. Достигается это за счет использования результата

работы алгоритма раскраски на предыдущем шаге. На каждой итерации отжиг делает небольшое локальное изменение. В результате конфликтовать с результатом раскраски, полученным на предыдущем шаге могут только 1-2 занятия, добавление которых в расписание с помощью алгоритмов раскраски происходит уже за линейное относительно размера расписания время. На следующей диаграмме показано улучшение производительности по времени и числу шагов на всех трех примерах, а так же среднее ухудшение одной итерации.



Коэффициент ухудшения одной итерации варировался в пределах от 1 до 2 на различных входных данных.

Результаты

Изучены различные подходы решения задач, связанных с составлением расписания: генетические алгоритмы, метод отжига, сведение к раскраске графа, потоки в сетях.

Проведенный анализ выявил, что применение задачи раскраски двудольного графа является наиболее перспективным для нахождения расписания в случае, когда рассматриваются только необходимые ограничения распределения занятий по времени.

Применительно к задаче составления расписания предложен новый способ применения задачи реберной раскраски двудольного графа (применение совместно с оптимизирующими алгоритмами). На примере метода отжига показана эффективность предложенного подхода.

Список литературы

[1] Brucker P. Scheduling algorithms. – Springer, 2007.

[2] Cole R., Hopcroft J. On edge coloring bipartite graphs //SIAM Journal on Computing. – 1982. – Т. 11. – №. 3. – С. 540-546.

[3] Plummer M. D., Lovász L. Matching theory. – North Holland, 1986. – Т. 121.

[4] Schaerf A. A survey of automated timetabling //Artificial intelligence review. – 1999. – Т. 13. – №. 2. – С. 87-127.

[5] Cauvery N. K., "Timetable Scheduling using Graph Coloring", *International Journal of P2P Network Trends and Technology*, Volume 1, Issue 2, 2011

[6] Burke E. K., Petrovic S. Recent research directions in automated timetabling //European Journal of Operational Research. – 2002. – Т. 140. – №. 2. – С. 266-280.

[7] de Werra D. An introduction to timetabling //European Journal of Operational Research. – 1985. – Т. 19. – №. 2. – С. 151-162.

[8] Even S., Itai A., Shamir A. On the complexity of time table and multi-commodity flow problems //Foundations of Computer Science, 1975., 16th Annual Symposium on. – IEEE, 1975. – С. 184-193.

[9] Vizing V. G. The chromatic class of a multigraph //Cybernetics and Systems Analysis. – 1965. – Т. 1. – №. 3. – С. 32-41.

[10] Лопатин А.С. "Метод отжига в задачах оптимизации", дипломная работа СПбГУ мат-мех кафедра системного программирования, 2004