Введение в дискретную оптимизацию. Осень 2010.

Лектор: Александр Дайняк

Конспект вел: Стас Смышляев

Этот конспект лекций редактировался мною минимально, поэтому воспринимать его следует «как есть». В частности, я не готов отвечать на вопросы по содержимому данного файла и исправлять в нём ошибки и/или опечатки. *А.Д.*

1 Лекция 1

1.1 Общая постановка задачи дискретной оптимизации

В самом общем виде задача оптимизации может быть поставлена в следующем виде: для данного множества S и определенной на S вещественной функции ϕ найти элемент $s \in S$, минимизирующий (или максимизирующий) функцию ϕ.

Часто (например, в задаче линейного программирования) множество $S \subseteq \mathbb{R}^n$ для некоторого натурального n.

Если $S \subseteq \mathbb{Z}^n$ (в общем случае — если S не более, чем счетно) то речь идет о задаче дискретной оптимизации.

Пример 1.1. **Задача о назначениях**

Пусть $m, n \in \mathbb{N}$, заданы неотрицательные вещественные числа $c_{i,j}$, $i = 1, \ldots, m$, $j = 1, \ldots, n$, где $c_{i,j}$ имеет смысл производительности рабочего i при решении задачи j.

Пусть

$$S = \{\mathrm{X} = (x_{i,j}) \in \{0,1\}^{m \cdot n} \mid \sum_{i=1}^{m} x_{i,j} \leqslant a_j, j = 1, \ldots, n; \sum_{j=1}^{n} x_{i,j} \leqslant b_i, i = 1, \ldots, m\},$$

$\phi(\mathrm{X}) = \sum\limits_i \sum\limits_j x_{i,j} c_{i,j}$. Здесь для всякой пары i, j значение булевой переменной $x_{i,j}$ определяет, назначен ли рабочий i на задачу j. Ограничения же имеют следующий смысл: на j-ю задачу рабочих должно быть назначено не более a_j, а также что всего i-й рабочий не должен быть назначен более, чем на b_i задач.

Требуется максимизировать ϕ, тем самым увеличив общую производительность труда.

Пример 1.2. **Задача коммивояжера**

Дан полный неориентированный граф $G = (V, E)$, все ребра которого помечены некоторыми неотрицательными вещественными числами или символом $+\infty$. Здесь множество V имеет смысл некоторого множества городов, которые требуется объехать коммивояжеру, причем не проезжая дважды по одному городу и вернувшись в итоге в исходный город (тем самым проделав путь, соответствующий некоторому гамильтонову циклу в графе G).

Требуется найти гамильтонов цикл в графе G с наименьшей суммой весов ребер.

Очевидно, здесь множество S — множество всех гамильтоновых циклов s в графе G, функционал $\phi(s)$ от гамильтонова цикла принимает значение суммы весов ребер цикла s (или $+\infty$, если хотя бы одно ребро s помечено как $+\infty$).

Задача заключается в минимизации ϕ на S.

1.2 Общие методы решения задач дискретной оптимизации

Принципиальное отличие задач дискретной оптимизации от "непрерывных" задач оптимизации состоит в том, что далеко не всегда удается действовать методом "локального поиска", т. е. рассматривая малые смещения в множестве S.

Формально метод локального поиска может быть записан следующим образом. Пусть (S, f) — задача оптимизации и пусть на множестве S задана некоторая окрестностная функция $N\colon S \mapsto 2^S$, ставящая в соответствие каждому элементу из S некоторое "близкое" элементу s подмножество S.

Например, в задаче о нахождении минимума непрерывной вещественной функции для всякого s логично взять $N(x) = (x - \varepsilon, x + \varepsilon)$.

Опишем формально метод локального поиска в общем виде.

1. выбираем s_{opt} произвольным образом из S;

2. если в окрестности $N(s_{opt})$ есть элемент s' такой, что $f(s') < f(s_{opt})$, положим $s_{opt} = s'$ и переходим к шагу 2, иначе переходим к шагу 3;

3. s_{opt} считаем приближенным значением.

Для задачи коммивояжера можно рассматривать функцию $N(s)$, соответствующую 2-окрестностям того или иного гамильтонова цикла.

2-окрестностью гамильтонова цикла s называется множество гамильтоновых циклов, которых можно получить следующим образом из цикла s:

1. удалить два ребра (u_1, u_2) и (v_1, v_2);

2. добавить два ребра (u_1, v_1) и (u_2, v_2).

Заметим, что 2-окрестность фиксированного гамильтонова цикла генерируется "быстро".

Аналогичным образом можно определить k-окрестность для любого $k \geqslant 2$.

Утверждение 1.3.
Если в качестве $N(s)$ в схеме локального поиска выбрать 2-окрестность, то существует такой граф и начальное приближение s_{opt}, что минимум не будет найден.

Доказательство.

Рассмотрим следующий граф (см. рис. 1, все неотмеченные ребра имеют вес 5).

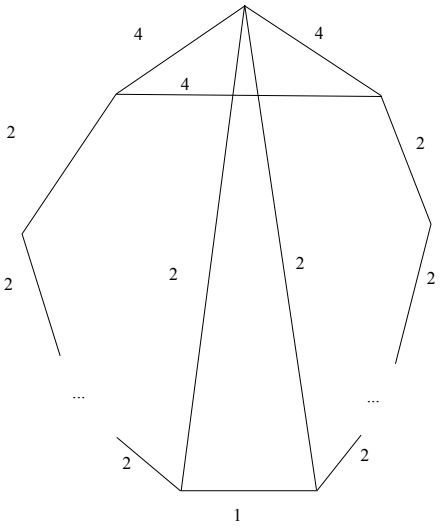

Рис. 1: Граф G

Если в качестве начального приближения взять внешний цикл, то для любого цикла из 2-окрестности начального приближения вес будет на единицу больше веса начального приближения, поэтому метод локального поиска выдаст начальное приближение как ответ. Тем не менее, цикл из ребер веса 2 — гамильтонов цикл с меньшим весом. □

Предложение 1.4. Если в качестве окрестности (для графа на n вершинах) выбрать $(n-3)$-окрестность, то существует такой граф и такое начальное приближение, что метод локального поиска не приведет к оптимальному решению задачи коммивояжера.

2 Лекция 2

Пусть дано множество S, для каждого элемента s которого определен вес $w(s)$, дано некоторое множество подмножеств $F \subseteq 2^S$ и требуется выбрать подмножество $S' \in F$ минимального (максимального) веса ($w(S') = \sum\limits_{s \in S'} w(s)$).

Опишем класс алгоритмов для решения задач такого типа, называемых жадными алгоритмами.

1. $S' = \varnothing$;

2. если существует элемент $s \notin S'$ такой, что $(S' \cup s) \in F$, выбрать из таких элементов элемент максимального веса и положить $S' = S' \cup s$, после чего перейти к шагу 2. Иначе вернуть текущее множество S' как решение.

Задача о минимальном остовном дереве формулируется следующим образом: пусть дан полный граф, ребра которого помечены неотрицательными числами. Требуется найти остовное дерево данного графа (подграф данного графа, являющийся деревом на всех вершинах исходного графа) минимального веса.

Можно показать (см. [В.Б.Алексеев. Введение в сложность алгоритмов.]), что в задаче о минимальном остовном дереве жадный алгоритм всегда приведет к оптимальному решению.

2.1 Матроиды

(в других материалах)

2.2 Лемма об изолировании

Лемма 2.1.
Пусть дано конечное множество S мощности n и некоторое множество его подмножеств $F \subseteq 2^S$.

Пусть веса элементов этого множества выбираются независимо случайным (с равномерным распределением) образом из множества $\{1, \ldots, N\}$ Тогда с вероятностью не меньшей $1 - \frac{n}{N}$ существует единственное подмножество S' такое, что $S' \in F$ и $w(S') \to \min$.

Доказательство.
Пусть $s \in S$. Рассмотрим величину $\alpha(s) = \min\limits_{A \subseteq S, A \in F, s \notin A} w(A) - \min\limits_{B \subseteq S, B \in F, s \in B} w(B \setminus \{s\})$.

Заметим, что для вычисления $\alpha(s)$ не нужно знать вес s, поэтому $\Pr(w(s) = \alpha(s)) = \frac{1}{N}$. Таким образом $\Pr(\forall s : w(s) \neq \alpha(s)) \geqslant 1 - \sum_s \Pr(w(s) = \alpha(s)) = 1 - \frac{n}{N}$.

Нетрудно видеть, что для завершения доказательства достаточно доказать следующий факт: если $A, B \in F$, $w(A) = w(B) = m = \min$, то для всякого $s \in B \setminus A$ выполняется $\alpha(s) = w(s)$.

Чтобы доказать этот факт, рассмотрим $s \in B \setminus A$. Так как $s \in B$, $s \notin A$, то $\alpha(s) = m - (m - w(s)) = w(s)$, что и завершает доказательство леммы. □

3 Лекция 3

3.1 Элементы теории сложности вычислений

Рандомизированный алгоритм отличается от детерминированного тем, что кроме входа получает бесконечную (вправо) последовательность случайных символов из $\{0, 1\}$, символы которой независимы в совокупности и равномерно распределены.

Через P обозначим класс полиномиальных алгоритмов.

Определение 3.1. Класс RP (randomized polynomial time) (класс $\mathrm{co-RP}$) — класс алгоритмов таких, что в случае «неправильного» входного слова выдают "нет" ("да") с вероятностью 1, а в случае «правильного» выдают ответ "да" ("нет") с вероятностью не менее $\frac{1}{2}$.

Определение 3.2. Класс ZPP (zero-error probabilistic polynomial time): класс полиномиальных в среднем рандомизированных алгоритмов, всегда выдающих правильный ответ.

Определение 3.3. Класс BPP (bounded-error probabilistic polynomial time): класс полиномиальных рандомизированных алгоритмов, выдающих ошибочный ответ с вероятностью не более $\frac{1}{2}$.

Лемма 3.4.
$\mathrm{ZPP} \subseteq \mathrm{RP} \cap \mathrm{co-RP}$.

Доказательство.
Покажем, что $\mathrm{ZPP} \subseteq \mathrm{RP}$ (второе вложение доказывается аналогично).

Пусть $L \in \mathrm{ZPP}$. Пусть математическое ожидание работы $t(n)$ алгоритма L на входе длины n не превосходит полинома $A(n)$.

Рассмотрим рандомизированный алгоритм L_2, работающий в течение $2t(n)$ шагов на любом входе длины n как L и выдающий ответ "да", если L успел выдать ответ "да" и выдает ответ "нет" иначе.

Очевидно, что L_2 работает полиномиальное время. Оценим вероятность ошибки L_2.

$$\Pr(L_2 \text{ на слове } x \text{ выдает "нет"} \mid x \in L) \leqslant \Pr(L \text{ на слове } x \text{ не успевает отработать за время } 2t(n)).$$

По неравенству Маркова:

$$t(n) = \sum_{i-\text{все возможные наборы на случайном входе}} t_i \cdot p_i \geqslant \sum_{\text{наборы на случайном входе, при которых время работы} \geqslant 2t(n)} =$$
$$= 2t(n) \cdot \Pr(\text{время работы} \geqslant 2t(n)).$$

Следовательно, вероятность ошибки L_2 не больше $\frac{1}{2}$. □

3.2 Коммуникационная сложность

Пусть Алиса и Боб хотят совместными усилиями вычислит значение некоторой функции f на данных (a, b), где a исходно находится у Алисы, а b — у Боба (будем считать, что ответ должен в итоге получиться у Боба).

Определение 3.5. Коммуникационной сложностью называется число символов, которые Алиса с Бобом суммарно передадут друг другу в процессе вычисления $f(a, b)$.

Лемма 3.6.
Количество простых чисел, которые делят некоторое натуральное N, не превосходит $\log_2 N$.

Доказательство.
$N = p_1^{\alpha_1} \dots p_s^{\alpha_s} \geqslant 2^{1^s} = 2^s$. □

Лемма 3.7. Количество простых чисел, не превосходящих N, не меньше $\frac{N}{\log_2 N}$.

Утверждение 3.8.
Алиса и Боб (обозначим их через A и B) могут с вероятностью $\frac{1}{2}$ проверить, что имеющиеся у них числа (a у Алисы и b у Боба) двоичной длины n равны с коммуникационной сложностью не более $O(\log n)$.

Доказательство.

1. A генерирует простое число $p \leqslant 2n^2$ и вычисляет $a \ mod p$; передает B пару $(p, a \ mod p)$.
2. если $a \equiv b(\ mod p)$, B выдает "да", иначе "нет".

Докажем корректность. Очевидно, что если $a = b$, то (A, B) выдаст "да". С другой стороны, по леммам 3.6 и 3.7,

$$\Pr(\text{ответ "да"} | a \neq b) = \Pr((a-b) \ mod p = 0 | a \neq b) \leqslant \frac{\log_2 2^n}{\text{количество простых чисел не превосходящих } 2n^2} \leqslant$$
$$\leqslant \frac{n}{2n^2/log_2 2n^2} < \frac{1}{2}.$$

□

Пусть ставится задача сравнить два заданных в некотором виде полинома P и Q на конечном множестве S степени не выше d. Очевидно, что данная задача эквивалентна задаче о сравнении заданного полинома Q степени не выше d с нулем.

Лемма 3.9.
Шварц, Зиппель
Пусть Q зависит от $x_1, x_2, \dots, x_n$. Выберем случайным образом константы $r_1, r_2, \dots, r_n \in S$. Тогда $\Pr(Q(r_1, r_2, \dots, r_n) = 0 \mid Q \neq 0) \leqslant \frac{d}{|S|}$.

Доказательство.
Докажем индукцией по n.

База. Если $n = 1$, то вероятность ошибки равна вероятности того, что r_1 — корень полинома Q, которая равна $\frac{\deg Q}{|S|}$.

Переход. Пусть x_1 входит в Q со степенью k. Тогда $Q(x_1, x_2, \dots, x_n) = x_1^k Q_1(x_2, x_3, \dots, x_n) + Q_2(x_1, x_2, \dots, x_n)$. $\Pr(Q(r_1, r_2, \dots, r_n) = 0 \mid Q \neq 0) = \Pr(A|B) + \Pr(A \mid \overline{B}) \leqslant \Pr(B) + \Pr(A \mid \overline{B})$, где $A = \{Q(r_1, \dots, r_n) = 0\}$, $B = \{Q_1(r_2, \dots, r_n) = 0\}$. Нетрудно видеть, что $\Pr(B) \leqslant \frac{d-k}{|S|}$, так как степень Q_1 не выше $d - k$. Кроме того, $\Pr(A \mid \overline{B}) \leqslant \frac{k}{|S|}$, что и завершает доказательство утверждения. □

3.3 Задача о паросочетаниях

Совершенное паросочетание в графе — разбиение двудольного графа на непересекающиеся пары соединенных ребрами вершин. Очевидно, что необходимым условием существования в графе совершенного паросочетания является равномощность долей. Однако легко показать, что достаточным данное условие не является.

Рассмотрим произвольный двудольный граф G с равномощными долями (мощности n) и рассмотрим его матрицу смежности M_G (без клеток, соответствующих заведомо несуществующим ребрам между вершин одной доли). Нетрудно видеть, что определитель матрицы M_G равен нулю тогда и только тогда, когда в G нет совершенного паросочетания.

Алгоритм определения существования совершенного паросочетания в двудольном графе:

1. по графу строим матрицу M_G;
2. выбираем случайным образом n чисел $(r_1, r_2, \ldots, r_{n^2})$ из множества $\{1, \ldots, 2n\}$;
3. подставляем $(r_1, r_2, \ldots, r_{n^2})$ в матрицу M_G, получаем M'_G. Если определитель M'_G не равен нулю, то выдаем, что совершенное паросочетание существует, иначе что не существует.

4 Лекция 4

4.1 Применение леммы об изолировании

Заметим, что каждое совершенное паросочетание в двудольном графе на $2n$ вершинах (по n в каждой половине) задает перестановку на множестве $\{1, 2, \ldots, n\}$. Перестановку, соответствующую совершенному паросочетанию, задает любое ненулевое слагаемое в сумме, задающей ненулевой определитель M_G.

Построим параллельный алгоритм для нахождения совершенного паросочетания.

Последовательный алгоритм мог бы выглядеть так. Пусть дан двудольный граф G, требуется найти в нем совершенное паросочетания. Выберем произвольное ребро графа. Если граф все еще содержит совершенное паросочетание, удалим выбранное ребро из графа и продолжим работать (искать в оставшейся части графа совершенное паросочетание).

Лемма 4.1.
Пусть в G веса ребер выбраны так, что существует единственное совершенное паросочетание минимального веса w_0. Пусть $\det M_G \mid_{x_{i,j} \to 2^{w((i,j))}} = D$. *Тогда максимальная степень двойки, которая делит число D равна 2^{w_0}.*

Доказательство.
Требуемое утверждение вытекает из равенства $\det M_G = \pm 2^{w_0} \pm 2^{w'} \pm 2^{w''} \pm \ldots$, в котором $w' > w_0$, $w'' > w_0$, ... □

Параллельный алгоритм (здесь и далее рассматриваем только алгоритмы с полиномиальным числом процессов и полилогарифмической глубиной).

1. Каждому ребру графа G приписываем случайный вес из множества мощности $2|E(G)|$, где $E(G)$ — множество всех ребер графа G. С вероятностью не меньшей, чем $\frac{1}{2}$, в графе существует единственное совершенное паросочетание минимального веса, если вообще есть паросочетание.
2. Возьмем $|E(G)|$ вычислительных единиц («процессоров») и каждому взаимно однозначным образом поставим в соответствие ребро. Каждый процессор должен выдавать свое ребро тогда и только тогда, когда оно входит в совершенное паросочетание минимального веса.
3. Вычислим $D = \det M_G \mid_{x_{i,j} \to 2^{w((i,j))}}$ и найдем w_0 как максимальную степень двойки, которая делит D.
4. Пусть процессор, отвечающий ребру (i, j), вычисляет $D'_{i,j}$ — определитель матрицы $M_{G \setminus \{(i,j)\}}$. Если максимальная степень двойки, которая делит $D'_{i,j}$, равна $w_0 - w(i, j)$, то процессор выдает свое рабро в ответ.
5. По доказанным леммам с вероятностью $\frac{1}{2}$ будет выдан набор ребер, составляющих совершенное паросочетание.

4.2 Применение к задаче коммивояжера

Рассмотрим еще раз задачу TSP (travelling salesperson problem) (задачу коммивояжера). Рассмотрим приближенные алгоритмы — алгоритмы, цель которых с некоторой точностью приблизить OPT — оптимальное решение — приближением $APPROX$. Если алгоритм умеет находить $APPROX$ такое, что $APPROX \leqslant \alpha \cdot OPT$, то он называется α-приближенным.

Часто рассматривают $(1+\varepsilon)$-приближенные алгоритмы. Тогда «полиномиальный» алгоритм — алгоритм, который тратит время $O(poly(\text{вход}, \frac{1}{\varepsilon}))$.

Рассмотрим метрическую задачу TSP (Metric TSP) — то есть, задачу TSP с подчиняющимися аксиомам метрики длинами ребер. Граф с весами на ребрах в этом случае будем называть метрическим.

Лемма 4.2.
Метрическая задача TSP эквивалентна задаче найти цикл в графе, проходящий через все вершины не менее одного раза, имеющий минимальный вес.

Доказательство.
Рассмотрим «квазигамильтонов цикл», не являющийся ГЦ. Тогда в этом цикле есть две вершины u, v, все вершины между которыми пройдены не менее двух раз. Легко заметить, что если мы заменим цепочку между u и v на ребро (u, v), то цикл останется квазигамильтоновым не большего веса. □

5 Лекция 5

Замечание 5.1. Очевидно, что любой квазигамильтонов цикл имеет длину не меньше суммарной длины ребер минимального остовного дерева.

Замечание 5.2. Нетрудно показать (порождая по произвольному гамильтонову циклу два паросочетания), что минимальный вес совершенного паросочетания не больше половины длины гамильтонова цикла.

Опишем $\frac{3}{2}$-приближенный полиномиальный алгоритм решения метрической задачи коммивояжера.
Алгоритм (Кристофидеса): пусть дан граф G с весами, удовлетворяющими аксиомам метрики.

1. Ищем T — остовное дерево G минимального веса.
2. Выбираем V' — множество всех вершин нечетной степени дерева T. Количество вершин нечетной степени в произвольном графе — четное число, поэтому $|V'|$ — четное.
3. Пусть G' — граф, полученный из G, удаляя все вершины, кроме вершин из V'.
4. В G' присутствует M — совершенное паросочетание минимального веса в G'. Суммарный вес ребер в M не больше половины веса мимимального гамильтонова цикла в G.
5. Рассмотрим G'': граф, полученный объединением множеств ребер M и T. Граф G'' заведомо связный; все вершины в G'' имеют четную степень. Следовательно, в G'' есть эйлеров цикл. Так как этот цикл в G'' — квазигамильтонов, то удалим повторяющиеся участки из него и предоставим как ответ.

5.1 Евклидова задача коммивояжера (Euclidean TSP)

Отличие — в качестве метрики выбрана евклидова метрика на плоскости (в прямоугольной системе координат); каждой точке сопоставлены координаты (x, y).

Утверждение 5.3.
Достаточно в ETSP искать гамильтонов цикл без самопересечений.

Доказательство.
Достаточно каждое самопересечение вида $(u_1, v_1) \cap (u_2, v_2)$ заменить на пару ребер (u_1, u_2),(v_1, v_2). □

Пусть мы ищем гамильтонов цикл для n точек плоскости. Будем считать, что эти точки находятся в квадрате со стороной n^2 и это минимально возможная длина стороны (иначе можно этого добиться масштабированием системы координат).

Будем считать, что координаты точек — целые числа от 0 до $L-1$.

Утверждение 5.4.
Если мы приблизим в полученной целочисленной задаче гамильтонов цикл с точностью ε, то этот цикл будет не больше, чем в $(1+\varepsilon+\frac{4}{n})$ длиннее, чем оптимальный гамильтонов цикл в исходной задаче.

Доказательство.

Расстояние, на которую переместится конкретная точка при округлении, не превосходит $\frac{\sqrt{2}}{2}$. Для каждой точки гамильтонова цикла в исходном графе сделаем следующее: добавим дважды ребро от точки до ее целочисленной аппроксимации. Если OPT_{int} — минимальный гамильтонов цикл в целочисленной задаче, то для OPT (минимальный гамильтонов цикл в исходной задаче) верно: $OPT \geqslant OPT_{int} - n\sqrt{2}$. Заметим, что OPT имеет длину не меньше n^2. Поэтому $n\sqrt{2} \leqslant \frac{OPT}{n\sqrt{2}}$ и $OPT_{int} \leqslant (1 + \frac{1}{n\sqrt{2}})$.

Цепочка неравенств $APPROX_{int} \leqslant (1+\varepsilon)OPT_{int} \leqslant (1+\varepsilon)(1+\frac{1}{n\sqrt{2}}) \cdot OPT$ завершает доказательство утверждения. □

Будем решать задачу методом «разделяй и властвуй». Разделим исходный квадрат на 4 равных квадрата, каждый из них — еще на 4. И так далее до того момента, когда дойдем до единичных клеток (очевидно, это произойдет через $2\log_2 n$ делений). На каждой стороне квадрата выберем m равноудаленных друг от друга точек, равномерно расположенных на стороне, которые будем называть порталами. Гамильтонов цикл теперь выбирается так, чтобы он пересекал стороны квадратов только в порталах.

Пусть $k = O(\log n)$ — высота дерева рекурсии. На i-м уровне рекурсии будем выбирать на каждой стороне квадрата ровно m порталов, где $m = 2^t$, $\frac{k}{\epsilon} \leqslant m \leqslant \frac{2k}{\varepsilon}$.

Заметим, что можно не увеличивая длину гамильтонова цикла добиться, чтобы этот гамильтонов цикл пересекал каждый портал не более двух раз (аналогичным избавлению гамильтонова цикла от самопересечений способом — перекидыванием ребер).

Часть гамильтонова цикла в каждом квадрате представляет собой набор непересекающихся линий, пересекающих стороны квадрата в порталах.

Для каждого портала есть три варианта взаимодействия с гамильтоновым циклом (не проходит через портал, входит и выходит через него же или проходит только один раз). Обходя квадрат по периметру, можно сопоставить каждому гамильтонову циклу правильную скобочную последовательность. Допустим, что квадрат разбит на четыре подквадрата и для каждого подквадрата и для всевозможных конфигураций (через какие порталы входит и какая скобочная последовательность ему соответствует) гамильтонова цикла в этом подквадрате мы знаем оптимальный маршрут, который гамильтонов цикл должен проходить по этому подквадрату, то по этому восстанавливается маршрут, проходимый гамильтоновым циклом по всему квадрату (перебором порядка $const^m$ конфигураций).

6 Лекция 6

Пусть задано расположение порталов на сторонах большого квадрата и пусть дана схема прохождения гамильтонова цикла по этим порталам. Перебором порядка не более $3^{4m} * 2^{4m} = O((6^4)^m) = O((6^4)^{\frac{const * \log n}{\varepsilon}}) = n^{O(1/\varepsilon)}$. Тогда если мы уже предрассчитали оптимальные фрагменты цикла в маленьких квадратах (для любых взаимодействий с порталами), то можно «склеить» из этих оптимальных фрагментов соответствующий фрагмент гамильтонова цикла в большом квадрате.

Утверждение 6.1.
Число пересечений отрезков оптимального гамильтонова цикла с линиями сетки не больше, чем удвоенная длина цикла (не больше $2 \cdot OPT_{int}$).

Доказательство.

Пусть есть две соединенные отрезком точки (x, y) и (x', y'). Очевидно, что число пересечений с вертикальными линиями сетки не превосходит $|x - x'|$, с горизонтальными — чем $|y - y'|$, поэтому общее число не больше $|x - x'| + |y - y'| \leqslant$ длина отрезка. Суммируя по всем отрезкам получим требуемое неравенство. □

Зафиксируем два числа a и b из отрезка $[0, \ldots, \frac{n^2-1}{2}]$ и рассмотрим разбиение квадрата на 4 прямоугольника, отсекая от нижнего левого угла отрезки a по горизонтали и b по вертикали.

Пусть $L = n^2$ — длина стороны квадрата. Линия сетки «попадает» в разбиение на i-м уровне рекурсии с вероятностью не более $\frac{2^i}{L/2}$. Среднее расстояние между порталами на линии разбиения на i-м шаге рекурсии равно $\frac{L}{2^i \cdot m}$ и одновременно этой же величине равна верхняя оценка удлинения гамильтонова цикла при пересечении линии разбиения на i-м уровне рекурсии. Таким образом, математическое ожидание удлинения гамильтонова цикла за счет пересечения конкретной линии сетки не превосходит $\sum\limits_{i=1}^{k} \frac{2^i}{L/2} \cdot \frac{L}{2^i \cdot m} = \frac{2k}{m} \leqslant 2\varepsilon$, следовательно общее удлинение (в среднем при случайном выборе разбиения) не превосходит $2\varepsilon \cdot 2OPT_{int} = 4\varepsilon OPT_{int}$.

6.1 Арифметические операции на «параллельных машинах»

Повторим, что при рассмотрении параллельных алгоритмах мы хотим, чтобы «глубина» алгоритма (соответствующая времени работы параллельного алгоритма) была не более, чем полиномиальной от логарифма длины входа.

Задача: вычислить определитель матрицы $A \in \mathbb{Q}^{n\times m}$ с глубиной не более $O((\log(n))^2)$.

Алгоритм (Czansky).

Утверждение 6.2.

Пусть $A = \begin{pmatrix} 1 & a_{12} & \dots & a_{1n} \\ 0 & 1 & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{pmatrix} = I - \Delta$ *— верхнетреугольная матрица. Тогда* $A^{-1} = I + \Delta + \Delta^2 + \dots \Delta^{n-1}$.

Доказательство.

$(I + \Delta + \Delta^2 + \dots \Delta^{n-1})(I - \Delta) = I - \Delta + \Delta - \Delta^2 + \Delta^2 \dots + \Delta^{n-1} - \Delta^n$. Очевидно, что $\Delta^n = 0$. Таким образом, утверждение доказано. □

Следствие 6.3. Для A такого вида обратная матрица вычисляется с глубиной $O((\log(n))^2)$. Сначала вычисляем $\Delta^2, \Delta^4, \dots, \Delta^{2^{\lfloor \log_2(n-1) \rfloor}}$, затем вычисляем Δ в остальных степенях, затем складываем все $\Delta, \dots, \Delta^{n-1}$.

Напомним, что для любой матрицы A существует некоторая обратимая матрица T такая, что $T^{-1}AT$ является верхнетреугольной, в которой на диагонали стоят собственные значения A (Жорданова форма). Тогда $\det(T^{-1}AT) = \det(A)$ равно произведению элементов на диагонали $T^{-1}AT$. Напомним также, что $tr(AB) = tr(BA)$ и $tr(T^{-1}AT) = tr(A)$ равен сумме собственных значений A. $tr(A^k) = tr(T^{-1}A^kT) = \lambda_1^k + \lambda_2^k + \dots \lambda_n^k$.

Симметрическими элементарными многочленами называются многочлены вида

$$p_k(x_1, x_2, \dots, x_n) = \sum_{1 \leqslant i_1 \leqslant \dots \leqslant i_k \leqslant n} x_{i_1} x_{i_2} \dots x_{i_k}.$$

Симметрическими ньютоновыми многочленами называются многочлены вида $s^m(x_1, \dots, x_n) = \sum\limits_{i=1}^{n} x_i^m$.

Замечание 6.4. $s^m(\lambda_1, \lambda_2, \dots, \lambda_n) = tr(A^m)$ — вычисляется «быстро». $p_n(\lambda_1, \dots, \lambda_m) = \lambda_1 \cdots \lambda_n = \det(A)$.

Введём ещё одно обозначение: $f_k^m(x_1, x_2, \dots, x_n) = \sum\limits_{1 \leqslant i_1 \leqslant \dots \leqslant i_k \leqslant n, j \neq i_1, \dots, i_k} x_{i_1} \cdots x_{i_k} \cdot (x_j)^m$.

Утверждение 6.5.

Справедливо соотношение: $p_k \cdot s^m = f_k^m + f_{k-1}^{m+1}$.

Доказательство.

$$\Big(\sum_{1 \leqslant i_1 \leqslant \dots \leqslant i_k \leqslant n} x_{i_1} \cdots x_{i_k}\Big)\Big(\sum_{j=1}^{n} x_j^m\Big) =$$

$$= \sum_{1 \leqslant i_1 \leqslant \dots \leqslant i_k \leqslant n, j \in \{i_1, \dots, i_k\}} x_{i_1} \cdots x_{i_k} x_j^m + \sum_{1 \leqslant i_1 \leqslant \dots \leqslant i_k \leqslant n, j \notin \{i_1, \dots, i_k\}} x_{i_1} \cdots x_{i_k} x_j^m.$$

□

7 Лекция 7

Рассмотрим следующую сумму:

$$p_k s^0 - p_{k-1} s^1 + p_{k-2} s^2 - \dots + (-1)^{k-1} p^1 s^{k-1} + (-1)^k s^k =$$
$$= f_k^0 + f_{k-1}^1 - (f_{k-1}^1 + f_{k-2}^2) + (f_{k-2}^2 + f_{k-3}^3) + \dots + (-1)^{k-1}(f_1^{k-1} + f_0^k) + (-1)^k s^k = f_k^0 + (-1)^{k-1} s^k + (-1)^k s^k = f_k^0 (s^0 - k) p_k.$$

Отсюда $p_k = \frac{1}{k}(p_{k-1}s^1 - p_{k-2}s^2 + \dots + (-1)^{k+1}s^k)$; $1p_k - \frac{s^1}{k}p_{k-1} + \dots + \frac{(-1)^k}{k}p_1 s^{k-1} = \frac{(-1)^{k+1}s^k}{k}$.

$$\begin{pmatrix} 1-\frac{s^1}{n} & \frac{s^2}{n} & \dots \\ 0 & 1 & \dots \\ \dots & \dots & \dots \end{pmatrix} \begin{pmatrix} p_n \\ \dots \\ p_1 \end{pmatrix} = \begin{pmatrix} \frac{(-1)^{n+1}s^n}{n} \\ \dots \\ s^1 \end{pmatrix}$$

Заметим, что матрица слева имеет верхнетреугольный вид и на диагонали стоят единицы, то есть мы (как было показано ранее) умеем ее быстро обращать.

Домножая слева и справа получившееся выражение на обратную матрицу, приходим к требуемому.

Заметим, что сравнивать n-битные числа можно с логарифмической глубиной (сравнивая независимо старшие и младшие половины чисел и затем по результатам двух сравнений вычисляя результат нужного сравнения). Кроме того, пользуясь тем, что неравенство $\mathbf{x} + \mathbf{y} > 2^n - 1$ выполнено тогда и только тогда, когда $\mathbf{x}$ больше инвертированного поразрядно $\mathbf{y}$, биты переноса при сложении также можно вычислять параллельно.

Построим параллельный алгоритм умножения чисел с полилогарифмической глубиной.

При умножении «в столбик» мы перемножаем первое число на все разряды второго числа.

Пусть $\mathbf{x} = (x_n, \dots, x_1), \mathbf{y} = (y_n, \dots, y_1)$ — два числа в двоичной системе счисления, имеющие длину n.

Задача сводится к быстрому сложению большого количества чисел: $(x_n \cdot y_1, \dots, x_1 \cdot y_1), (x_n \cdot y_2, \dots, x_1 \cdot y_2, 0), \dots$

Перейдем от сложения трех чисел к сложению пары чисел. Будем для трех сложений вычислять биты переноса и биты сложения, каждый раз вынося биты переноса в одно число $\mathbf{u}$, а биты сложения в число $\mathbf{v}$ — очевидно, это можно сделать параллельно.

Таким образом сложение каждой тройки чисел сведется к сложению пары чисел. Проводя аналогичную процедуру достаточное число раз, вычислим требуемую сумму.

8 Лекция 8

8.1 Метаэвристики

Метаэвристики — «эвристики для создания эвристик».

Одним из примеров метаэвристик является локальный поиск.

Рассмотрим популярный вид метаэвристик — GA (генетические алгоритмы), а также ACA («муравьиные колонии»).

Пусть f — целевая функция, которую мы хотим максимизировать на множестве S. Пусть также задана функция кроссовера $d(s_1, \dots, s_k)$, выдающая по набору элементов из множества S новый элемент S. Кроме того, пусть задана (случайная) функция мутации $m(s)$.

Генетический алгоритм.

1. Берем $S' \in S$ — начальная популяция.
2. Выбираем несколько наборов $s_{1,1}, \dots, s_{1,k}$, вычисляем $s'_1 = d(s_{1,1}, \dots, s_{1,k})$, …, $s'_n = d(s_{n,1}, \dots, s_{n,k})$.
3. Вычисляем $f(s'_1), f(s'_2), \dots, f(s'_n)$, выбираем максимизирующих целевую функцию потомков.
4. Вычисляем $f(m(s'_1)), f(m(s'_2)), \dots, f(m(s'_n))$, выбираем максимальные. Получаем множество S'''.
5. $S' = S'''$, переходим к шагу 2.

Замечание 8.1. Если отсутствует кроссовер, то алгоритм вырождается в локальный поиск. В качестве $m(s)$ можно брать функцию, возвращающую произвольный элемент из окрестности.

Пример 8.2. Вернемся к TSP — задаче коммивояжера. В качестве $m(s)$ можно выбирать случайный элемент из 2-окрестности s. В качестве $d(s_1, s_2)$ можно выбирать функцию, «скрещивающую» два гамильтонова цикла: берётся кусок первого гамильтонова цикла (длина этого куска может выбираться случайно), а затем к нему добавляются остальные вершины в том порядке, в котором они лежат на втором гамильтоновом цикле.

Алгоритм муравьиных колоний.

Пусть дан граф G, s — точка начала, t — точка назначения. Каждому ребру e ставится в соответствие «запах» $sc(e) \geqslant 0$. Параллельно из s в t выпускаются агенты. Каждый агент движется с большой вероятностью по тому ребру, на котором запах больше. Каждый муравей, проходя по некоторому пути, оставляет на всех ребрах свой запах — причем тем больше, чем больше значение целевой функции этого пути.

Пример 8.3. Фиксируем произвольную вершину v и запускаем из нее агентов. На каждом шаге агент движется по тому ребру, на котором либо больше запах, либо вес ребра меньше и которое ведет в еще не посещенную агентом вершину. Дойдя до v, агент оставляет на пути запах, обратно пропорциональный длине гамильтонова цикла.

Разумно также на каждой итерации слегка «развеивать» запах каждого ребра — с целью избежать ситуации, когда новые добавления запаха не будут существенно влиять на сложившееся ранее распределение запахов.

9 Лекция 9

9.1 Целочисленное линейное программирование

Пусть дана система уравнений $A\mathbf{x} = \mathbf{b}$, $\mathbf{x} \geqslant \mathbf{0}$ (жирным шрифтом здесь и далее будем выделять вектора) и требуется максимизировать (минимизировать) функционал $(\mathbf{c}, \mathbf{x})$ — дана задача линейного программирования. Если требуется, чтобы элементы вектора $\mathbf{x}$ являлись целыми числами, то говорят о задаче целочисленного линейного программирования (ЦЛП).

Замечание 9.1. Заметим, что задачу ЦЛП нельзя решать, решая сначала задачу линейного программирования, а затем округляя до ближайших целых.

Пример 9.2. Задача TSP может быть сведена к задаче ЦЛП следующим образом: пусть переменная $x_{i,j}$ соответствует прохождению гамильтонова цикла через ребро (i, j). Тогда выделим следующие необходимые условия на соответствие набора значений $x_{i,j}$ гамильтонову циклу:

$$\begin{cases} x_{i,j} \geqslant 0, \ \ i, j \in \{0, 1, \ldots, n\}, \\ \sum\limits_{i=0}^{n} x_{i,j} = 1, \ \ j \in \{0, 1, \ldots, n\}, \\ \sum\limits_{j=0}^{n} x_{i,j} = 1, \ \ i \in \{0, 1, \ldots, n\}, \\ \sum\limits_{i,j} c_{i,j} x_{i,j} \to \min, \end{cases} \tag{1}$$

где $c_{i,j}$ есть вес ребра (i, j).

Разобьем множество вершин на множества $\{i_0, \ldots, i_s\}$ и $\{i_{s+1}, \ldots, i_n\}$ и добавим следующее условие:

$$\sum_{i \in \{i_0, \ldots, i_s\}, j \in \{i_{s+1}, \ldots, i_n\}} x_{i,j} \geqslant 1.$$

Если выписать такие условия для всех возможных разбиений вершин и добавить к условиям (1) то, как нетрудно проверить, получится группа необходимых и достаточных условий того, что набор значений $x_{i,j}$ соответствует гамильтонову циклу. Таким образом, получим сведение задачи TSP к задаче ЦЛП.

Уменьшить количество условий можно, используя так называемые ограничения Таккера. Заметим, что группа условий (1) представляет собой критерий того, что значения переменных $x_{i,j}$ соответствуют разбиению графа на непересекающиеся циклы. Чтобы определить условие единственности цикла, введем n новых переменных $u_1, \ldots, u_n$; для всяких i, j, $i \neq j$ введем ограничения $u_i - u_j + n x_{i,j} \leqslant n - 1$. Содержательно это означает, что каждый из циклов проходит через вершину 0.

Действительно, пусть существует цикл, не проходящий через вершину 0. Возьмем те переменные u_i, что сопоставлены вершинам этого цикла $i_1, i_2, \ldots, i_k$ (проходимым именно в этом порядке). По ограничениям Таккера, $u_{i_1} - u_{i_2} + n x_{i_1, i_k} \leqslant n - 1$, …, $u_{i_{k-1}} - u_{i_k} + n x_{i_{k-1}, i_k} \leqslant n - 1$, $u_{i_k} - u_{i_1} + n x_{i_k, i_1} \leqslant n - 1$. Складывая их, получим: $n \sum x \leqslant (n-1)k$. Так как сумма в левой части равна k, придем к противоречию.

С другой стороны, пусть переменная u_{i_j} соответствует номеру вершины i_j в некотором гамильтоновом цикле. Если нет дуги из j' в j'', то должно быть выполнено $u_{j'} - u_{j''} \leqslant n - 1$. Если есть дуга, то для дуги из вершины i_l в вершину i_{l+1} неравенство принимает вид: $u_{i_l} - u_{i_{l+1}} + n \leqslant n - 1$.

Можно выделить следующие подходы к решению задачи ЦЛП. Очевидно, что если ввести новое ограничение, не удаляющее ни одной точки с целочисленными координатами, то соответствующий оптимум сохранится.

1. Метод отсекающей гиперплоскости — сужаем область поиска оптимума, добавляя гиперплоскость, отсекающую часть точек, среди которых нет целочисленных.

2. Метод ветвей и границ — разбиваем исходную задачу на две подзадачи (разбивая пространство на два подпространства). Если окажется, что оптимальное (нецелочисленное) решение в подзадаче B равно ξ_B и есть некоторая точка в задаче A, значение функционала в которой меньше ξ_B, то задачу B дорешивать не нужно.

Метод ветвей и границ может быть применен для задачи TSP следующим образом: разбиваем задачу поиска гамильтонова цикла на поиск среди циклов, содержащих ребро e и не содержащих. «Прогоняем» для задачи A алгоритм Кристофидеса — получаем ξ_A; для B находим нижнюю оценку (например, как вес минимального остовного дерева) — получаем ξ_B. Если $\xi_A < \xi_B$, можно отбросить задачу B.

Еще один подход к разбиению задачи TSP (при решении методом ЦЛП) — поиском семейства непересекающихся циклов. Если в решении более одного цикла, то берем один из полученных циклов $(i_1, \ldots, i_k)$ и разбиваем исходную задачу на подзадачи: A_1: $x_{i_1,i_2} = 0$, A_2: $x_{i_2,i_3} = 0$, ...

10 Лекция 10

10.1 Онлайновые алгоритмы

В некоторых задачах имеет смысл рассматривать модель, в которой исходные данные даны не сразу, а приходят по мере работы. Алгоритмы, работающие в таких моделях, называются онлайновыми (online) алгоритмами.

Очевидно, что онлайновый алгоритм в общем случае всегда сработает не лучше оффлайнового (которому все исходные данные даются сразу).

Задача: оценить, во сколько раз решение, предлагаемое онлайновым алгоритмом хуже (в некотором определенной конкретной задачей смысле) оптимума в оффлайновом случае.

Онлайновый алгоритм, который выдает решение не более, чем в α раз хуже оптимального, называется α-конкурентоспособным (α-competitive).

Рассмотрим задачу построения расписания (scheduling). Пусть есть m параллельно работающих процессоров (будем считать их одинаковыми), есть n задач, каждая из которых требует некоторого своего времени выполнения t_i (будем считать, что задачи прерывать и откладывать нельзя), требуется так распределять задачи по процессорам (сразу во время поступления), чтобы общее время выполнения было минимальным. При этом каждая задача может считаться одновременно не более, чем одним процессором, а один процессор может считать не более одной задачи в каждый момент времени.

Жадный алгоритм в этой задаче будет выглядеть так: вновь пришедшую задачу помещаем на процессор, меньше всего загруженный в расписании.

Теорема 10.1.
Жадный алгоритм всегда выдает решение не более, чем в два раза превосходящее оптимальное ($GREEDY \leqslant 2 \cdot OPT$).

Доказательство.

Введем параметр T_1 — момент времени, когда прекращает работу первый из процессоров (а до этого работали все), $T_2 = GREEDY - T_1$. Рассмотрим задачу, которая завершилась последней. Ее время выполнения t_j, очевидно, не меньше T_2. Имеем $OPT \geqslant \max\limits_{1 \leqslant i \leqslant n} t_i \geqslant t_j \geqslant T_2$. С другой стороны, $OPT \geqslant \frac{1}{n} \cdot \sum\limits_{i=1}^{n} t_i \geqslant T_1$. Складывая последние два неравенства, получаем требуемое. □

Замечание 10.2. Можно также доказать, что $GREEDY \leqslant (2 - \frac{1}{m}) \cdot OPT$.

Замечание 10.3. Существует онлайновый алгоритм, выдающий при достаточно больших m решение не больше, чем $1.945 \cdot OPT + const$. Такой алгоритм (называемый IMBALANCE) описывается следующим образом.

Назовем высотой h_i процессора i в данный момент времени — время, которое он проработал бы еще, если бы на него больше не приходили бы задачи. Очередную задачу алгоритм IMBALANCE назначает процессору,

для которого выполнено следующее: высота h_j этого процессора в сумме с временем t_k выполнения задачи не больше $1.945 \cdot A_j$, где A_j — средняя высота первых $j-1$ по загруженности процессоров (A_0 при этом считаем бесконечно большим).

Задача балансировки загрузки (LOAD BALANCING).

Пусть есть m параллельно работающих процессоров (будем считать их одинаковыми), есть n задач, каждая из которых требует некоторого своего неизвестного нам времени выполнения и имеет свой вес w_i. При этом каждая задача может считаться одновременно не более, чем одним процессором, а каждый процессор может считать сколь угодно много задач в каждый момент времени. Требуется минимизировать максимальную нагрузку на все процессоры в каждый момент времени, где под нагрузкой подразумевается сумма весов считаемых на процессоре задач.

Алгоритм ROBIN HOOD. Назовем "богатым" тот процессор, нагрузка которого в даннный момент времени больше или равна $\sqrt{m} \cdot L$, где L рассчитывается из соотношения: $L := \max\{L, w_k, \frac{1}{m} \cdot (w_k + \sum\limits_{i=1}^{m} l_i(t))\}$, где $l_i(t)$ — нагрузка на i-й процессор в данный момент времени t. Вновь пришедшая задача назначается любому бедному процессору (если он есть), а если таковых нет, то тому богатому процессору, который стал богатым позже всех.

Теорема 10.4. $ROBIN\ HOOD \leqslant OPT \cdot O(\sqrt{m})$.

Замечание 10.5. Доказано, что у любого онлайнового алгоритма выдаваемое в худшем случае решение больше или равно $\Omega(\sqrt{m}) \cdot OPT$.