Лекция 14

Приближенные алгоритмы (I)

(Конспект: Я. М. Подольский)

[Стиль автора [конспектировавшего] (в основном) сохранен. Орфография и пунктуация (в основном) исправлены. –Э.А.]

14.1 Приближенные алгоритмы

Как известно, достопочтенный мой читатель, не все задачи имеют алгоритмическое решение или имеют, но использовать такие не стоит нашего времени. Поэтому умные люди собрались и придумали искать не оптимальные решения, а те, которые почти оптимальные. Как мы будем искать такие решения? Все очень просто! Для начала, неплохо бы придумать некую оценку для решения, а потом придумывать алгоритмы, оценка которых нас удовлетворит!

Определение 14.1. *Максимизационная задача* — это массовая задача, снабженная целевой функцией f : решения $\to \mathbb{R}_{\oplus}$, определяющей “качество решений”. Для условия u требуется найти решение x_o, на котором f достигает наибольшего значения. *Минимизационная задача* определяется аналогично.

Определение 14.2. Алгоритм для максимизационной задачи называется *α-приближенным*, если он выдает решение x, для которого выполняется $f(x) \geqslant \alpha f(x_o)$. Для минимизационной задачи условие трансформируется в $f(x) \leqslant \alpha f(x_o)$.

14.2 Задача о рюкзаке

Ну, чтобы не голословить, давайте решим приближенно всеми любимую задачу о рюкзаке. Напомним условие:
Дан объем рюкзака V, количество предметов в нашем распоряжении n, ценность каждого из предметов $p[i]$ $(1 \leqslant i \leqslant n)$, и объемы предметов $w[i]$.

Как решали ее наши российские студенты в первом семестре: конечно же, динамическим программированием! Выявили подзадачу: какой минимальный объем рюкзака можно занять, для того, чтоб положить туда вещей данной суммарной ценности?

Для этого мы, конечно, заведем таблицу $W[,]$. И будем заполнять ее по правилу: в $W[k,p]$ будет решение подзадачи, какой достаточен объем, если предметы — с 1 по k, а нужная ценность — p.

$$W[k,p] = \min(w[k] + W[k-1, p-p[k]], W[k-1,p]),$$

т.е. мы разобрали два случая, взяли ли мы k-ый предмет в набор или нет. K пробегает $1..n$, p пробегает $1..\sum_{i=1}^{n} p[i]$. Вместо элементов, находящихся за пределами таблицы, используем

$$W[\ldots, \leqslant 0] = 0; \quad W[0, > 0] = +\infty.$$

После всего этого давайте оглянемся и ужаснемся!!! Табличка-то — размера $n \times \sum_{i=1}^{n} p[i]$!!![1] Такими темпами мы решение найдем не скоро.

Давайте найдем приближенный алгоритм для нашей "рюкзаковой" задачи! Мы будем строить $(1-\varepsilon)$-приближенный алгоритм. Для начала уменьшим имеющиеся ценности предметов:

$$\varepsilon' := \frac{\varepsilon}{1-\varepsilon};$$
$$K_\varepsilon := \frac{\max_i p[i]}{n \cdot (1 + 1/\varepsilon')};$$
$$p'[i] := \left\lceil \frac{p[i]}{K_\varepsilon} \right\rceil;$$

тогда

$$\max_i p'[i] = \left\lceil \frac{\max_i p[i]}{K_\varepsilon} \right\rceil = \lceil n \cdot (1 + 1/\varepsilon') \rceil = \left\lceil \frac{n}{\varepsilon} \right\rceil$$

[1] Это три восклицательных знака, а не факториала. И то хорошо...

теперь матрица алгоритма имеет размер $O(n^2)$. (Заметим, что ϵ влияет на константу в этом $O(\ldots)$.) Продолжим наши изыски.

Пусть оптимальное решение — $\{p[i]\}_{i\in I}$, его стоимость — P_o. Тогда после преобразования мы получим $\{p'[i]\}_{i\in I}$, для которого $\sum_{i\in I} p'[i] \geqslant P_o/K_\varepsilon$; наш алгоритм найдет набор не меньшей “преобразованной” стоимости. Однако, после того, как мы перейдем к прежним ценностям, мы получим стоимость P_ε, возможно, меньше оптимальной (так уж мы округляли ценности), но все же $P_\varepsilon \geqslant P_o - K_\varepsilon n$. Наконец,

$$\frac{P_\varepsilon}{P_o} \geqslant \frac{P_o - K_\varepsilon n}{P_o} = 1 - \frac{\max_i p[i]\cdot n}{P_o n(1+1/\varepsilon')} \geqslant 1 - \frac{1}{1+1/\varepsilon'} = 1 - \varepsilon.$$

Таким образом, мы получили решение, стоимость которого отличается от оптимальной не больше чем в $1-\varepsilon$ раз.

14.3 Задача о коммивояжере

Студенты мат-меха знают, что задача о коммивояжере, скорее всего[2], не имеет алгоритма, работающего за полиномиальное время. Поэтому будем решать ее приближенно!

Условие: дан неориентированый граф с весами. *Гамильтонов цикл* — это цикл, проходящий по всем вершинам, но не проходящий через какую-либо вершину более одного раза. Надо найти гамильтонов цикл минимального суммарного веса.

Будем решать задачу о коммивояжере *в метрическом пространстве* (нам важно выполнение правила треугольника). Она трудна даже в такой постановке. Ясно, что можно считать, что граф — полный (отсутствующие ребра просто имеют вес $+\infty$).

14.3.1 Простой алгоритм

Когда-то давным-давно в первом семестре мы искали минимальное остовное дерево. Вот нам этот алгоритм и пригодился. Найдем его (дерево).

Продублируем все ребра в этом дереве. Получим некий цикл, почти удовлетворяющий нашим требованиям. Докажем: если вес нашего цикла w, то $w < 2w'$ где w' — вес оптимального пути. В самом деле, если выкинуть из оптимального цикла одно ребро, то получится остовное дерево; а мы использовали минимальное остовное дерево.

Сделаем наш цикл гамильтоновым. Пойдем по нашему циклу; если мы пришли в вершину в которой мы уже были, то пойдем в в следующую.

[2]Если P≠NP.

В конце концов мы пройдем все вершины. Из неравенства треугольника следует, что мы такими срезками только укорачивали путь.

Итак, мы получили 2-приближенный алгоритм.

14.3.2 1.5-приближенный алгоритм

Рассмотрим минимальное остовное дерево. Рассмотрим вершины его, имеющие нечетные степени. Их четное число. Найдем для них совершенное паросочетание минимального веса в (полном) подграфе, индуцированном этими вершинами. Добавим это паросочетание к дереву.

В этом графе есть эйлеров цикл[3], т.к. все вершины — четной степени.

Вес паросочетания $\leqslant$ половины веса оптимального цикла (пронумеруем ребра в порядке их следования в оптимальном цикле, тогда у нас будет 2 совершенных паросочетания — ребра четные и нечетные; каждое — веса не больше нашего паросочетания минимального веса). Таким образом, суммарный вес полученного решения $\leqslant 3/2$ оптимального веса.

Остается найти искомое паросочетание. Сделать это можно, и даже не очень сложно. Но доказательство длинное, поэтому этот алгоритм мы опустим.

[3] Т.е. проходящий через каждое ребро ровно по одному разу.