Введение в экспоненциальные алгоритмы

Юрий Лифшиц*
yura@logic.pdmi.ras.ru.

Осень'2005

План лекции

1. Общая идеология экспоненциальных алгоритмов
2. Применения рекурсии
3. Локальный поиск
4. Вероятностные алгоритмы

1 Общая идеология

При изучении алгоритмов одно из ключевых мест занимает анализ их времени работы. Первым шагом является нахождение асимптотических оценок. Для широкого круга возникающих на практике задач удается получить такую оценку, которая выражается некоторым полиномом от размера обрабатываемых данных.

Но, все же, во множестве случаев в наше время найдены только решения, перебирающие экспоненциально зависящее от размера входных данных количество вариантов. Обычно в этом случае можно сконструировать программу, проверяющую правильность найденного решения за полиномиальное время (верификатор). Если это так, то задачу относят к классу NP. В нем выделяется подкласс NP-полных задач, к которым можно свести (за полиномиальное время) все остальные. То есть нахождение полиномиального решения для какой-нибудь из NP-полных задач означало бы возможность непосредственного построения полиномиального алгоритма для любой задачи класса NP. А поскольку уже довольно длительное время человечество не может найти удовлетворительное решение ни для одной из них, существует гипотеза о том, что его невозможно найти вообще.

*Законспектировал И. Синев.

Но необходимость решать эти задачи на практике не отпала. Сформировались два принципиально разных подхода: пытаться максимально соптимизировать неполиномиальные решения или, в случаях, когда это уместно, ограничиться поиском приближенного решения.

Сейчас речь пойдет о точных алгоритмах, которые всегда ищут оптимальное решение.

Безусловно, при больших размерах решаемых задач применение алгоритмов с экспоненциальным временем невозможно. Однако, для обработки данных небольшого объема экспоненциальные решения очень часто могут оказаться приемлемыми. Например, при $n = 100$ экспоненциальная оценка 1.1^n гораздо лучше, чем $10n^3$.

При анализе экспоненциальных алгоритмов возникает необходимость в поиске оценок, отличных от тех, которые приходилось искать при анализе полиномиальных алгоритмов. Предположим, что для некоторой задачи известен алгоритм, решающий ее за $O(2^n)$ действий. Тогда удвоение производительности компьютеров (которое, согласно закону Мура происходит каждые три года) позволит увеличить размер поддающихся решению задач приблизительно на единицу. Если же вместо этого удалось бы найти алгоритм, выполняющий $O(1.41^n)$ операций, это дало бы удвоение параметров задач, которые можно решить. Отсюда видно, что константа в экспоненциальных алгоритмах имеет решающее значение.

1.1 Методы решения NP-полных задач

Существует множество методов построения алгоритмов для решения NP-полных задач. Как основные, среди них можно выделить:

- Рекурсия
- Усовершенствованный перебор
- Локальный поиск
- Применение случайного порядка действий
- Вероятностное сведение к простой задаче
- “Случайное блуждание”
- Разделяй и властвуй

В данной лекции будут рассмотрены несколько примеров таких алгоритмов для двух конкретных задач.

2 Задача выполнимости булевой формулы (SAT) и задача о раскраске графа в три цвета.

Первая из задач формулируется следующим образом: по заданной в конъюнктивной нормальной форме булевой формуле найти подстановку (набор

значений переменных), при которой формула становится истинной, или сообщить о том, что это невозможно. Можно показать, что любую булеву формулу можно привести к 3-конъюктивной нормальной форме (3-CNF). При этом она должна быть представлена конъюнкцией нескольких подформул, каждая из которых — дизъюнкция ровно трех компонент вида x или $\not x$. Приведение заключается в описании в виде 3-CNF формулы дерева разбора исходной булевой формулы.

Вторая задача, которая будет рассмотрена — задача о 3-раскрашиваемости: по заданному неориентированному графу найти его правильную раскраску в три цвета или сообщить, что ее не существует.

Нетрудно видеть, что для первой задачи полный перебор дает решение со временем работы $O(2^n)$. В то время, как для второй для достижения такой же асимптотической оценки надо заметить, что цвет первой вершины в каждой компоненте связности можно задать произвольно, а для всех последующих один из трех цветов будет запрещен.

3 Рекурсия

3.1 Рекурсивный алгоритм для 3-SAT (1.84^n)

Данный алгоритм был придуман Мониеном и Шпекенмайером в 1985 году. Простой перебор поочередно пробует подставить вместо каждой из переменных 0 или 1, а затем рекурсивно решает задачу меньшего размера. Тогда можно применить следующую оптимизацию. Рассмотрим какую-нибудь подформулу (например $(x \vee y \vee z)$). Попытаемся подставить $x = 1$. Если найти решение не удалось, то больше никогда не имеет смысла подставлять вместо x единицу. Таким образом, на следующем шаге перебора можно попытаться подставить сразу два значения — $x = 0, y = 1$. Если попытка опять была неудачной, остается подставить $x = 0, y = 0, z = 1$, так как хотя бы одна из переменных в скобке должна быть равна 1. Для времени работы данного алгоритма $T(n)$ можно записать рекуррентное соотношение

$$T(n) \leq T(n-1) + T(n-2) + T(n-3)$$

решая которое получаем $T(n) = O(1.84^n)$.

3.2 1.89^n для 3-раскрашиваемости

В обеих рассматриваемых задачах число 3 является существенным. Ведь и для задаче о раскраске графа в 2 цвета, и для задачи 2-SAT есть полиномиальные алгоритмы. Вспомнив это, к решению задачи о раскраске графа в 3 цвета можно подойти следующим образом. Очевидно, что в какой-то из цветов покрашено не более $n/3$ вершин. Предположим, что это синий цвет. Переберем, какие именно вершины будут синими, и для каждого из таких способов раскраски решим задачу о раскраске в 2 цвета для оставшихся вершин (это несложно сделать с помощью обхода в глубину или в ширину).

А так как $\sum_{i=0}^{n/3} C_n^i \leq 1.89^n$, то время работы данного алгоритма составляет $O(1.89^n)$

4 Локальный поиск

Следующий алгоритм использует идею о постепенном улучшении существующего решения — метод локального поиска. Действительно, пусть зафиксированы значения нескольких переменных. Рассмотрим какую-нибудь невыполненную скобку. В выполняющем наборе, который мы ищем, хотя бы одна из переменных в этой скобке принимает значение, отличное от текущего. Значит, если рассмотреть подстановки, отличающиеся от текущей в значении ровно одной из трех переменных, то в одном из этих случаев подстановка улучшится в смысле уменьшения количества различий с целевой. Таким образом, если начальная подстановка отличалась от искомой в значениях d переменных, то за 3^d шагов решение будет найдено. Заметим теперь, что любой набор, в частности, выполняющий, должен отличаться или от подстановки из всех 0, или от подстановки из всех 1 не больше, чем в $n/2$ позициях. Это значит, что если от каждой из этих двух подстановок запустить описанный выше перебор с глубиной $n/2$ шагов, решение будет найдено. Время работы такого алгоритма составляет $O(3^{n/2}) = O(1.74^n)$

Дальнейшее улучшение этого алгоритма можно получить с помощью покрывающих кодов — наборов подстановок, что наименьшее расстояние от любой точки B^n до некоторой из них минимально среди наборов такого размера.

5 Вероятностные алгоритмы

Когда идет речь о поиске экспоненциальных алгоритмов, часто полезными оказываются вероятностные методы. В этом случае разрешается, чтобы вычисления в некоторых случаях не завершались или даже давали неправильный ответ. Но, как оказывается, иногда это не исключает возможности их практического применения.

Пусть известен алгоритм, работающий за время $T(n)$ и выдающий правильный ответ с вероятностью p и ошибающийся только в одну сторону. Если повторить его $c \cdot p^{-1}$ раз, то в итоге вероятность ошибки станет

$$(1-p)^{c/p} \leq e^{-c}$$

.

Поскольку константы не учитываются, то можно потребовать, чтобы вероятность ошибки была меньше любой наперед заданной константы. Например, вероятности сбоя вычислителя. Поэтому на практике вероятностные алгоритмы используются столь же широко, как и детерминированные.

5.1 Классификация вероятностных алгоритмов

Вероятностные алгоритмы можно поделить на три основные группы:

- Алгоритмы с двусторонней ошибкой:

  Если ответ “Да”, алгоритм говорит “Да” с вероятностью хотя бы 2/3

  Если ответ “Нет”, алгоритм говорит “Нет”

  с вероятностью хотя бы 2/3

- Алгоритмы с односторонней ошибкой:

  Если ответ “Да”, алгоритм говорит “Да”

  Если ответ “Нет”, алгоритм говорит “Нет”

  с вероятностью хотя бы 2/3

- Алгоритмы с нулевой ошибкой:

  С вероятностью хотя бы 2/3 — правильный ответ

  В остальных случаях алгоритм говорит “не знаю”

Самыми удобными для применения являются алгоритмы третьей группы, но, как сейчас будет видно, соблюдать такие сильные ограничения при их построении получается не всегда.

5.2 Сведение к полиномиальной задаче

Достаточно простая идея при позволяет получить вероятностный алгоритм для задаче о раскраске, использующий полиномиальное решение для более простой задачи. Случайным образом назначим для каждой вершины пару цветов, в которые ее можно покрасить. Для каждой вершины вероятность угадать подходящую пару составляет 2/3, а тогда вероятность нахождения такой “предраскраски” для каждой из вершин сразу равна $(2/3)^n$. Но для случая, когда для каждой из вершин необходимо выбрать один из двух цветов, существует полиномиальный алгоритм. Значит, многократное повторение такой операции дает вероятностный алгоритм с оценкой времени работы $O(1.5^n)$. При этом ошибка может произойти только в случае отрицательного результата.

5.3 Изменение порядка шагов

На первый взгляд может показаться, что, так как перебор в любом случае должен рассматривать все существующие варианты, то порядок выполнения шагов шаги не имеет значения. Но это не так. Алгоритм для решения задачи о выполнимости, найденный Paturi и другими в 1997 году, всегда дает точный ответ, но использует недетерминированность для улучшения максимального по всем наборам входных данных среднего времени работы.

Пусть переменные переупорядочены случайным образом. В этом порядке им рекурсивно присваиваются некоторые значения. Тогда достаточно часто оказывается, что на некотором шаге в какой-то подформуле неизвестной остается только одна переменная и ее значение определяется однозначно. Из-за этого глубину рекурсии можно оценить как $2n/3$. Тогда время работы алгоритма в среднем составляет $O(2^{2n/3}) = O(1.58^n)$.

Идея о перестановке не изменяет времени работы алгоритма в худшем случае. Ведь микроскопическая вероятность переупорядочить переменные именно так, чтобы алгоритм делал максимальное число шагов, все же существует. Но это не уменьшает ценность метода. В данном случае существует возможность, например, поставить ограничение на время работы и получить вероятностный алгоритм с нулевой ошибкой.

5.4 Вероятностный локальный поиск

Очень эффективным оказывается применение вероятностного подхода в комбинации с описанной выше идеей локального поиска. При детерминированном локальном поиске вычислитель должен был перебирать все $O(3^d)$ вариантов совершить d изменений.

Вместо этого можно каждый раз пытаться производить изменения случайным образом. Тогда с вероятностью приблизительно $(1/3)^d$ ответ, если он существует, будет найден. Пока не достигнуто никакого улучшения по сравнению с обычным локальным поиском. Но если разрешить каждый раз совершать не d, а $3d$ приближений, то вероятность нахождения ответа станет близкой к $(1/2)^d$ (константа $(1/2)^d$ достигается при стремлении количества действий к бесконечности). Этот результат связан с теорией Марковских цепей: с вероятностью $1/3$ происходит переход в сторону цели, с вероятностью $2/3$ в противоположенную сторону.

Если теперь случайным образом выбран исходный набор, то значение каждой из переменных совпадает с правильным с вероятностью $1/2$. Тогда математическое ожидание того, что алгоритм найдет ответ равняется

$$E[(1/2)^{X_1+\cdots+X_n}] = \prod_{i=1}^{n} E[(1/2)^{X_i}] = (3/4)^n$$

откуда итоговое время работы алгоритма составляет $(4/3)^n = 1.33^n$.

Заключение

Напоследок стоит напомнить основные моменты, затронутые на этой лекции:

- При решении трудоемких задач очень большую роль играет изобретение принципиально новых методов
- Особую важность имеют вероятностные алгоритмы. Очень часто они оказываются даже более полезными, чем детерминированные