Конспект к лекции 2. (Санкт-Петербург, 8 апреля 2017 г.)

5 Применение однородного экспандера: уменьшение вероятности ошибки алгоритма без увеличения числа случайных битов

В этом параграфе мы рассмотрим несложный способ получения «псевдослучайных» битов с помощью экспандера. Мы покажем, как уменьшить вероятности ошибки вероятностного алгоритма *без увеличения числа используемых случайных битов*. Мы ограничимся рассмотрением алгоритмов с односторонней ошибкой. Напомним стандартное определение класса задач, для которых существует полиномиальный вероятностный алгоритм с односторонней ошибкой.

Определение 5.1 *Язык L принадлежит сложностному классу* RP*, если существует полиномиальный алгоритм $\mathcal{A}$ такой что*

1. *если $x \in L$, то для случайно выбранного набора битов $r \in \{0,1\}^{\mathrm{poly}(n)}$ $\mathrm{Prob}_r \mathcal{A}(x,r) \geqslant 1/2$,*
2. *если $x \notin L$, то $\mathcal{A}(x,r) = 0$ для всех $r \in \{0,1\}^{\mathrm{poly}(n)}$.*

Покажем, что для любого $\delta > 0$ всякий полиномиальный вероятностный алгоритм $\mathcal{A}$ можно переделать в полиномиальный вероятностный алгоритм $\mathcal{A}'$ таким образом, что вероятность ошибки уменьшится с $1/2$ до δ, а число используемых случайных битов при этом не изменится.

Пусть исходный алгоритм использует $k = k(n)$ случайных битов для вычислений на входах длины n. Зафиксируем однородный $(2^k, d, \varepsilon)$-экспандер G для некоторого $\varepsilon > 0$. Индекс (номер) каждой вершины в этом графе записывается последовательностью из k нулей и единиц. Таким образом, мы можем отождествить вершины G и наборы из k битов.

Новый алгоритм действует следующим образом: выбирается случайная вершина v графа (для этого требуется k случайных битов); затем исходный алгоритм $\mathcal{A}$ последовательно запускается на всех d наборах случайных битов, соответствующих соседям вершины v в графе. Если все полученные ответы равны 0, новый алгоритм также возвращает 0; если же получен хотя бы один положительный ответ, то алгоритм возвращает 1.

Покажем, что у нового алгоритма вероятность ошибки не превосходит $\frac{1}{2(1+\varepsilon)}$. Обозначим $B = B(x)$ множество всех *плохих* (для данного x) вершин графа — множество таких вершин w из правой доли графа, которые соответствуют неверному ответу старого алгоритма на входе x. Аналогично, обозначим $C = C(x)$ множество таких вершин v графа, которые для которых новый алгоритм даёт неверный ответ на входе x. Очевидно, C состоит из вершин, все соседи которых лежат в B.

Предположим, что C содержит не менее $\frac{1}{2(1+\varepsilon)}$ вершин. Произвольным образом выберем из множества C некоторое подмножество, состоящее *ровно* из $\frac{1}{2(1+\varepsilon)}$ вершин и назовём его C'. Из определения экспандера следует, что

$$|\Gamma(C)| > (1+\varepsilon)|C'| = n/2.$$

Это противоречит тому, что все соседи C' лежат в B.

Таким образом, мы построили алгоритм, в котором ошибка снизилась с $\frac{1}{2}$ до $\frac{1}{2(1+\varepsilon)}$. Покажем, как понизить вероятность ошибки ещё больше. Зададимся некоторым числом t и построим алгоритм, вероятность ошибки которого меньше $\frac{1}{2(1+\varepsilon)^t}$. В новом алгоритме мы выбираем в графе случайную вершину v (для по-прежнему этого требуется k случайных битов); затем запускаем исходный алгоритм $\mathcal{A}$ на всех наборах случайных битов, соответствующих вершинам w графа, в которые можно попасть из v за t шагов (таких вершин заведомо не более d^t). Если все полученные ответы равны 0, новый алгоритм также возвращает 0; в противном случае возвращается 1.

Оценим вероятность ошибки нового алгоритма. Снова обозначим $C = C(x)$ множество таких вершин v графа, которые для которых новый алгоритм даёт неверный ответ на входе x. Предположим, что C содержит не менее $\frac{1}{2(1+\varepsilon)^t}$ вершин. Выберем среди них подмножество, состоящее из ровно $\frac{1}{2(1+\varepsilon)^t}$ вершин и назовём его C'. Из определения экспандера следует, что

$$|\underbrace{\Gamma(\Gamma(\Gamma(\ldots\Gamma(}_{t \text{ итераций}} C')\ldots)))| > (1+\varepsilon)^t|C'| = n/2$$

Это противоречит тому, что все цепочки из t рёбер, начинающиеся вершиной из C', обязаны заканчиваться вершиной из B.

Выбирая параметр t достаточно большим, мы получим алгоритм с вероятностью ошибки менее $\frac{1}{2(1+\varepsilon)^t} < \delta$. При этом значение t зависит от желаемой вероятности ошибки δ, но не зависит от размера входа n.

Остаётся обсудить время работы построенного алгоритма. Мы используем старый алгоритм как «чёрный ящик» и вызываем его (на разных наборах случайных битов) d^t раз. Поскольку d и t – некоторые константы (не зависящие от входа алгоритма), и исходный алгоритм $\mathcal{A}$ работал за полиномиальное время, можно заключить, что все требуемые вызовы выполняются за полиномиальное время.

Однако кроме нескольких вызовов старого алгоритма нам требуется производить манипуляции с графом G — нужно уметь быстро находить всех соседей заданной вершины графа. Чтобы иметь возможность делать это за полиномиальное время, нам нужна *явная* конструкция экспандера. Более того, нам нужен экспандер *явный в сильном смысле*: размер графа экспоненциально растёт с увеличением k, и нам необходим алгоритм, который по заданному номеру вершины w за время $\mathrm{poly}(k)$ находит список номеров всех соседей w.

Следующее упражнение описывает один из классических вероятностных алгоритмов с односторонней ошибкой.

Упражнение 5.1 (Алгоритм Соловея–Штрассена для проверки простоты числа) *(а) Докажите, что значение* символа Якоби $(\frac{a}{b})$ *можно вычислить за время, полиномиально зависящее от длины входа (т.е., от длины двоичных записей a и b).*

(б) Вероятностный алгоритм Соловея–Штрассена тестирует простоту натурального числа n следующим образом. Выбирается случайное натуральное число a из интервала $2, \ldots, n-1$. Если наибольший общий делитель a и n больше единицы, алгоритм собщает, что число n составное. В противном случае проверяется сравнение $a^{\frac{n-1}{2}} = (\frac{a}{n}) \mod n$. Если данное равенство не выполняется, то алгоритм сообщает, что n составное; иначе алгоритм сообщает, что n простое.

Докажите, что алгоритм всегда возвращает правильный ответ для простых n и ошибается с вероятностью не более $1/2$ для каждого из составных n.

Упражнение 5.2 (Проверка существования совершенного паросочетания в двудольном графе, Mulmuley–Vazirani–Vazirani) *Пусть задан двудольный граф $G = (L, R, E)$, где $|L| = |R| = n$ (множество вершин состоит из двух долей по n вершин в каждой), и $E \subset L \times R$ (каждое ребро соединет некоторую вершину из доли L с некоторой вершиной из доли R). Такой граф можно задать матрицей M размера $n \times n$,*

$$m_{ij} = \begin{cases} 1, & \text{если } i\text{-ая вершина } L \text{ соединена ребром с } j\text{-ой вершиной } R, \\ 0, & \text{иначе.} \end{cases}$$

Заменим каждую единицу в матрице M в клетке с координатами (i, j) на переменную x_{ij}. Определитель полученной матрицы будет многочленом от переменных x_{ij} (этот определитель можно рассматривать как многочлен над $\mathbb{R}$ или над конечным полем $\mathbb{F}_q$).

(а) Докажите, что определитель построенной матрицы является нулевым многочленом (все коэффициенты которого равны нулю), если и только если в графе нет совершенного паросочетания (*совершенным паросочетанием называется такое множества рёбер P, что каждая вершина графа инцидентна ровно одному ребру из P).*

(б) Докажите, что для любого набора значений x_{ij} (из конечного поля $\mathbb{F}_q$) соответствующую величну определителя можно вычислить за время, полиномиально зависящее от n и от размера поля. Указание: *чтобы вычислить определить матрицы размера $n \times n$, нет необходимости выписывать его в явном виде как сумму $n!$ мономов.*

(в) Рассмотрим конечное поле из q элементов, $q > 2n$. Подставим в определитель описанной матрицы случайно и независимо выбранные элементы поля вместо каждой переменной x_{ij}. Докажите, что

- *если в графе есть совершенное паросочетание, то значение определителя, вычисленного на случайном наборе элементов поля, оказывается равным нулю с вероятностью менее $1/2$;*

- *если в графе нет совершенного паросочетания, то значение определителя вычисленного на любом наборе элементов поля равно нулю.*

Указание: *если многочлен небольшой степени не равен нулю тождественно, то он обращается в ноль в сравнительно небольшом числе точек.*

(г) С помощью наблюдения из пункта (б) постройте полиномиальный вероятностный алгоритм, проверяющий существование совершенного паросочетния в двудольном графе размера $n \times n$: если соврешенного паросочетания нет, алгоритм должен всегда выдавать ответ 0, а если хотя бы одно совершенное паросочетание есть, то алгоритм должен выдавать ответ 1 с вероятностью $>1/2$.

6 Применение однородных экспандеров: коды, испрвляющие ошибки

Напомним, что (двоичным) *кодом* называется набор слов $\mathcal{C} \subset \{0,1\}^n$. Элементы $\mathcal{C}$ называют *кодовыми словами*, число n называют *длиной кодового слова*, а $M = |\mathcal{C}|$ — *объёмом кода*. С помощью кода объема S можно пересылать наборы их $s = \lfloor \log S \rfloor$ битов (сопоставив разным наборам из m битов разные кодовые слова). *Минимальным расстоянием кода* называют наименьшее хэмминговское расстояние между парой кодовых слов. Если кодовое расстояние равно r, то говорят, что данный код позволяет исправлять $\left\lfloor \frac{r-1}{2} \right\rfloor$ ошибок. (Если в кодовом слове инвертировать до $\left\lfloor \frac{r-1}{2} \right\rfloor$ битов, мы можем однозначно восстановить исходное слово). Таким образом, если мы хотим исправлять ошибки в доле δ от всех битов кодового слова, то необходимо, чтобы минимальное расстояние кода удовлетворяло неравенству $r \geqslant 2\delta n + 1$.

Таким образом, всякий код характеризуется параметрами n (длина кода), m (чило передаваемых битов) и r (минимальное расстояние между кодовыми словами). Отношение s/n называют *скоростью кода* (это отношение является свого рода коэффициентом полезного действия кода — оно показывает, сколько «полезных» битов удаётся переслать в расчёте на один бит кодового слова). Задача теории кодирования состоит в поиске кодов с оптимальным соотношением параметров: при фиксированных n и s максимизировать r или при фиксированных n и r максимизировать s. Отдельная (и с практической точки зрения очень важная) задача — построение кодов с эффективными алгоритмами декодирования. Такой алгоритм должен восстанавливать кодовое слово, если в нём искажено не слишком большое число битов.

Важным классом кодов с разными замечательными свойствами являются *линейные* коды — такие коды, для которых множество кодовых слов $\mathcal{C} \subset \{0,1\}^n$ оказывается линейным подпространством в $\mathbb{F}_2^n$. Для линейных кодов минимальное расстояние между кодовыми словами равняется минимальному возможному числу единиц в ненулевом кодовом слове. В этом параграфе мы покажем, как с помощью экспандеров строить линейные ко-

ды. Эти коды будут обладающие достаточно хорошим соотношением параметров; кроме того, мы предъявим для этих кодов быстрые алгоритмы декодирования.

Напомним, что линейный код с длиной кодового слова n задаётся его проверочной матрицей H (слово $x \in \{0,1\}^n$ является кодовым словом, если и только если $Hx^{\perp} = 0$). Другими словами, чтобы описать линейный код, мы должны задать систему линейных уравнений для переменных $x_1, \ldots, x_n$ над полем из двух элементов; решения этой системы и будут кодовыми словами.

Зафиксируем некоторый двудольный $(n, m, k, d, \varepsilon)$-экспандер. Сопоставим переменные $x_1, \ldots, x_n$ вершинам в левой доле графа. Вершинам из правой доли будут соответствовать уравнения. А именно, каждой вершине v из правой доли G мы сопоставляем уравнение

$$x_{i_1} + \ldots + x_{i_s} = 0 \ (mod\ 2),$$

где $x_{i_1}, \ldots, x_{i_s}$ – это список вершин, соединённых рёбрами с v.

При этом нам будет нужно, чтобы для параметров $(n, m, k, d, \varepsilon)$-экспандера выполнялись следующие соотношения:

$m < cn$ для некоторого $c < 1$,
$k > 2\delta n$, где δ есть доля исправляемых ошибок,
$\varepsilon < 1/2$.

Число уравнений равно m, так что размерность пространства решений не меньше $n - m$. Это значит, что в нашем коде будет не менее $2^{n-m} = 2^{\Omega(n)}$ кодовых слов.

Остаётся доказать, что данный код действительно исправляет δn ошибок. Для этого нужно проверить, что расстояние между любыми кодовыми словами больше $k = 2\delta n$. Для линейного кода нужное нам условие эквивалентно тому, что в каждом ненулевом кодовом слове должно быть более k единиц.

Предположим противное: пусть существует некоторое ненулевое кодовое слово $x_1 \ldots x_n$ (решение системы линейных уравнений, соответствующих графу G), в котором менее k единиц. Обозначим A множество вершин из левой доли графа, соответствующих единицам в данной последовательности битов. Поскольку $|A| < k$, можно применить определение экспандера: число соседей A достаточно велико,

$$|\Gamma(A)| > (1 - \varepsilon)d|A|.$$

Из A выходит ровно $d|A|$ рёбер. Оценим среднее (по всем вершинам $v \in \Gamma(A)$) число ребер, которое приходит из A в v. Это число не превосходит

$$\frac{d|A|}{(1-\varepsilon)d|S|} = 1/(1-\varepsilon) < 2$$

Итак, среднее число ребер, соединяющих вершину из $\Gamma(A)$ с множеством A, больше нуля и меньше двух. Это значит, что хотя бы у одной вершины

v из правой доли есть *ровно один* сосед из A. Но в таком случае уравнение, соответствующее v, не выполняется на наборе $x_1 \dots x_n$. Значит, набор битов с менее чем k единицами не может быть кодовым словом.

Для построения экспандерных кодов нужны экспандеры *явные в слабом смысле*: нам нужно научиться строить неоднородные экспандеры с n вершинами в левой доле (и подходящими значениями других параметров) за время, полиномиально зависящее от n.