

# Трудные для типичного входа задачи (изложение теории Average case complexity Левина)

Н.К. Верещагин

March 24, 2009

## 1 Что значит, что задача проста для типичного входа?

Пусть фиксирована некоторая задача разрешения, то есть, некоторая функция  $A$  из множества всех двоичных слов  $\Xi$  в множество  $\{0, 1\}$ .

Игра между Загадчиком (Challenger) и Отгадчиком (Solver) состоит из двух ходов. Первым ходит  $Z$ , выбирая некоторый вопрос  $x \in \Xi$ , затем ходит  $O$ , указывая некоторое  $y = 0, 1$ .  $O$  выиграл, если  $y = A(x)$ .

В терминах этой игры можно дать определение классов  $P$ ,  $BPP$ , а также интересующего нас класса простых в типичном случае задач.

Очевидно, что  $O$  имеет выигрышную стратегию в этой игре, однако эта стратегия не обязательно вычислима за полиномиальное время. Если  $O$  имеет полиномиально вычислимую стратегию, то говорят, что задача разрешима за полиномиальное время (принадлежит классу  $P$ ).

Пусть  $p$  положительное число от 0 до 1. Мы говорим, что задача разрешима с вероятностью успеха  $p$ , если  $O$  имеет полиномиально вычислимую вероятностную стратегию, гарантирующую выигрыш с вероятностью не менее  $p$ . Другими словами, если существует вероятностный алгоритм  $R$ , который на вход получает  $x$  и останавливается за полиномиальное от длины  $x$  время (при любых исходах бросания монеток), при этом для всех  $x$  вероятность события  $R(x) = A(x)$  не меньше  $p$ .

Любая задача всегда разрешима с вероятностью успеха  $1/2$  (стратегия  $O$  выдает случайный бит). Если же  $p > 1/2$ , то это уже не так, и

мы получаем класс ВРР (который не зависит от выбора  $p$ ).

Основным в теории сложности для типичного случая является ограничение вычислительных возможностей Загадчика. Может оказаться, например, что у Отгадчика имеется полиномиальный по времени алгоритм, который правильно решает задачу за исключением очень редкого множества входов, причем входы из этого множества очень трудно найти (то есть требуется много времени). Тогда такая полиномиальная стратегия  $O$  будет обыгрывать любого ограниченного во времени Загадчика. Сказанное требует уточнения, поскольку следует определить, чем именно ограничивается время Загадчика.

Будем считать, что оба игрока получают на вход натуральный параметр  $n$ , называемый *параметром безопасности*. Будем рассматривать вероятностные стратегии Загадчика, вычисляемые за полиномиальное от  $n$  время: стратегия получает на вход число  $n$  в унарной записи и выдает на выход некоторое  $x \in \Xi$ .

## Генераторы примеров

Полиномиальные по времени вероятностные стратегии для Загадчика будем называть *генераторами примеров*. То есть, генератор примеров — это вероятностный алгоритм  $G$ , который, получив на вход любое натуральное  $n$  (в унарной записи) выдает за время, ограниченное некоторым полиномом от  $n$ , некоторое слово  $x \in \Xi$ . При любых результатах бросаний алгоритм  $G$  должен остановиться за полиномиальное от  $n$  время.

Для каждого  $n$  рассмотрим распределение вероятностей  $\mu_n$ , получаемое на выходе алгоритма  $G$ . То есть,  $\mu_n(x)$  есть вероятность того, что алгоритм на входе  $n$  выдаст  $x$ . Семейство  $\mu_n$  распределений вероятностей, для которых существует генератор примеров, будем называть "генерируемыми семействами распределений". (В литературе очень похожее понятие называется "samplable distributions". Распределение  $\mu$  (а не семейство распределений) называется *samplable*, если существует вероятностный алгоритм без входа, который останавливается за время ограниченное полиномом от длины своего результата и распределение на выходах которого совпадает с  $\mu$ .)

*Полиномиальной по времени вероятностной стратегией Отгадчика* будем называть вероятностный алгоритм, который получает на вход число  $n$  в унарной записи и слово  $x$  и выдает за полиномиальное (от  $n$  и

длины  $x$ ) время 0 или 1.

## Распределенные задачи

Пары  $(A, \mu_n)$ , состоящие из задачи  $A$  и некоторого генерируемого распределения  $\mu_n$  на ее входах, называются *распределенными* задачами.

Мы рассмотрим два определения простой для типичного входа распределенной задачи, слабое и сильное, и в дальнейшем изложении ограничимся слабым понятием.

**Сильное определение.** Будем называть распределенную задачу *равномерно простой для типичного входа*, если существует вероятностный алгоритм  $S$ , который на вход получает два числа  $n, k$  и слово  $x$  и за полиномиальное от  $n + k + |x|$  время выдает 0 или 1, причем для всех  $n$  с вероятностью не меньше  $1 - 1/k$  выполнено  $S(1^n, 1^k, x) = A(x)$  (по распределению  $\mu_n$  на  $x$  и по равномерному распределению на исходах случайных бросаний алгоритма  $S$ ).

**Слабое определение.** Будем называть распределенную задачу *простой для типичного входа*, если для любого полинома  $q(n)$  существует полиномиальный вероятностный алгоритм  $S$  такой, что для всех  $n$  с вероятностью не меньше  $1 - 1/q(n)$  выполнено  $S(1^n, x) = A(x)$  (по распределению  $\mu_n$  на  $x$  и по равномерному распределению на исходах случайных бросаний алгоритма  $S$ ).

Если задача равномерно проста, то она проста. Действительно, пусть алгоритм  $S$  удовлетворяет определению равномерной простоты. Тогда для любого полинома  $q(n)$  алгоритм  $S(1^n, 1^{q(n)}, x)$  является полиномиальным и решает задачу  $A$  с вероятностью ошибки не более  $1/q(n)$  по распределению  $\mu_n$ .

С другой стороны, если задача  $A$  принадлежит ВРР, то для любого семейства распределений  $\mu_n$ , распределенная задача  $(A, \mu_n)$  равномерно проста. Действительно, существует полиномиальный вероятностный алгоритм  $S$ , который для любого входа  $x$  вычисляет  $A(x)$  с вероятностью ошибки не более  $1/3$ . Повторяя вычисление  $S(x)$  не более  $O(\log k)$  раз и выдавая наиболее популярный результат, мы можем сделать вероятность ошибки меньше  $1/k$  (на любом входе  $x$ ). При этом время вычисления будет равно  $O(\log k \cdot \text{poly}(|x|))$ , что даже меньше требуемого  $\text{poly}(k, |x|)$ .

Пример: пусть задача состоит в определении, содержит ли данный граф гамильтонов цикл, а распределение  $\mu_n$  равномерно на графах с  $n$

вершинами (каждое из возможных  $n(n-1)/2$  ребер присутствует с вероятностью  $1/2$  и все ребра независимы). Эта задача является равномерно простой для типичного входа (я не знаю, как это доказать).

## 2 Трудные для типичного входа задачи

### Сводимость распределенных задач

Мы хотим определить, что значит, что распределенная задача  $(A, \mu_n)$  сводится к распределенной задаче  $(B, \nu_n)$ . Хотелось бы, чтобы отношение сводимости уважало простоту (если задача сводится к простой задаче  $B$ , то она тоже проста) и было рефлексивным и транзитивным.

Годится ли обычная полиномиальная сводимость (сводимость Карпа):  $A$  сводится к задаче  $B$ , если для некоторой полиномиально вычислимой функции  $f$  для всех  $x$  выполнено  $A(x) = B(f(x))$ ?

Если не накладывать дополнительных ограничений, то это определение не годится. Причина в том, что  $f$  может отображать входы с большой вероятностью (относительно  $\mu$ ) во входы с малой вероятностью (относительно  $\nu$ ). Может оказаться, что вход  $x$ , имеющий вероятность, близкую к 1, отображается во вход  $f(x)$ , имеющий вероятность близкую к нулю. Вероятностный алгоритм для решения задачи  $B$  может выдавать на входе  $f(x)$  случайный ответ (0 или 1 с равными вероятностями). Ошибка с вероятностью  $1/2$  на этом входе мало повлияет на общую вероятность его успеха, поскольку  $f(x)$  имеет очень малую вероятность. А нам ответ этого алгоритма не дает никакой информации (мы и сами могли подбросить монетку).

Поэтому надо запретить функции  $f$  отображать входы с большой вероятностью во входы с малой вероятностью. А именно, мы потребуем, чтобы распределение  $\nu$  в некотором смысле мажорировало распределение, которое генерируется функцией  $f$  при распределении  $\nu$  на ее входах. Это распределение обозначается  $f(\mu_n)$  и определяется как

$$f(\mu_n)(y) = \mu_n\{x \mid f(x) = y\}.$$

Скажем, что распределение  $\mu_n$  мажорирует распределение  $\nu_n$ , если для некоторых полиномов  $p(n), q(n) \geq n$  и для всех  $n, x$  выполнено

$$\nu_n(x) \leq p(n)\mu_{q(n)}(x).$$

Разрешим функции  $f$  также зависеть от  $n$ . Итак, определение аналога сводимости Карпа таково. Распределенная задача  $(A, \mu_n)$  сводится по Карпу к распределенной задаче  $(B, \nu_n)$ , если существует полиномиально вычислимая функция  $f$  такая, что, во-первых,  $A(x) = B(f(1^n, x))$  для всех  $n, x$  и, во-вторых, распределение  $f(1^n, \mu_n)$  мажорируется распределением  $\nu_n$ . Нетрудно убедиться, что сводимость Карпа сохраняет простоту и является транзитивной. (Доказательство опускается.)

## Наибольшее генерируемое распределение

Заметим, что если распределение  $\nu_n$  мажорирует распределение  $\mu_n$ , то распределенная задача  $(A, \mu_n)$  сводится к задаче  $(A, \nu_n)$  (положим  $f(1^n, x) \equiv x$ ).

Оказывается для каждой задачи  $A$  существует самая трудная ее рандомизация  $(A, \mu_n)$ : если эта рандомизация проста, то и все остальные рандомизации просты. Более того, распределение  $\mu_n$  можно взять одним для всех задач  $A$ . В качестве такого распределения мы возьмем генерируемое распределение  $\mu_n$ , которое мажорирует все другие генерируемые распределения (наибольшее генерируемое распределение). Существование такого распределения гарантируется следующей леммой.

**Лемма 1.** *Существует генерируемое распределение  $\mu_n$ , которое мажорирует все другие генерируемые распределения.*

*Proof.* Неформально,  $\mu_n$  есть взвешенная сумма всех полиномиально генерируемых распределений. Точнее, чтобы сгенерировать случайную строку по наибольшему распределению  $\mu_n$ , мы выбираем случайным образом генератор  $G$ , затем находим наибольшее  $k$ , для которого время работы  $G$  на входе  $1^k$  не превосходит  $n$  и выдаем результат работы  $G$  на этом входе.

Точнее надо сказать так. Будем обозначать через  $\tilde{G}$  строку, которая идентифицирует программу генератора  $G$  и полином  $p_G$ , ограничивающий его время работы. Можно так выбрать эту строку  $\tilde{G}$ , чтобы существовала универсальная машина, которая на входе  $1^k 0 \tilde{G}$  моделирует работу генератора  $G$  на входе  $1^k$  с полиномиальным замедлением времени. Это означает, что ее время работы на входе  $1^k 0 \tilde{G}$  не больше  $q_G(k)$  (для некоторого полинома  $q_G$ ). Мы выбираем случайно  $G$ , находим  $k$ , для которого  $q_G(k) = n$  (если такого  $k$  нет, то выдаем, что угодно). Затем

запускаем универсальную машину на входе  $1^k 0 \tilde{G}$  и выдаем её результат на выход.

Мы не уточнили, какое распределение вероятностей на генераторах мы используем. Нам достаточно, чтобы при порождении распределения  $\mu_{q_G(k)}$  генератор  $G$  появлялся бы с положительной вероятностью  $\varepsilon(G)$ , не зависящей от  $k$ . Тогда для распределения  $\nu_k$ , порождаемого генератором  $G$ , выполнено

$$\nu_k(x) \geq \varepsilon(G) \mu_{q_G(k)}(x).$$

□

Генерируемое распределение, мажорирующее все другие генерируемые распределения, будем называть *наибольшим*.

Сводимость Карпа можно обобщить, разрешив сводящему алгоритму быть вероятностным и разрешив равенству  $A(x) = B(f(1^n, x))$  не выполняться с небольшой вероятностью. Но проще сразу рассмотреть значительно более слабую сводимость — аналог полиномиальной сводимости Тьюринга (сводимости Кука).

Сводимость Кука естественным образом определяется на задачах более общего типа. А именно, пусть задано бинарное отношение  $E$  на множестве двоичных слов, такое, что для каждого  $x$  существует  $y$ , для которого  $E(x, y)$ . Задача состоит в том, что для данного  $x$  надо найти такое  $y$ .

Будем называть *решателем* процедуру  $S$ , которой можно задать любой вопрос вида  $(1^n, x)$  и в ответ она нам мгновенно выдает некоторую случайно выбранную по некоторому распределению строку. (Формально, решатель — это любое семейство независимых в совокупности случайных величин  $S(1^n, x)$ .) Будем говорить, что  $S$  решает задачу  $(B, \nu_n)$  с вероятностью ошибки не более  $1/h(n)$ , если для любых  $n$  при случайном выборе  $x$  по распределению  $\nu_n$  вероятность события “ $S(1^n, x)$  неправильно решает задачу  $B$ ” не больше  $1/h(n)$ . Будем говорить, что распределенная задача  $(A, \mu_n)$  *сводится по Куку* к распределенной задаче  $(B, \nu_n)$ , если для любого любого полинома  $g(n)$  существует полиномы  $h(n), q(n)$  и вероятностная полиномиальная машина Тьюринга  $M$  с оракулом, обладающая следующим свойством. Для любого решателя  $S$  задачи  $(B, \nu_n)$  с вероятностью ошибки не более  $1/h(n)$  алгоритм  $M^S$  правильно решает задачу  $A$  (по мере  $\mu_n$  и по случайности, входящий в состав  $S$  и  $M$ ) с вероятностью ошибки не более  $1/g(n)$ . При этом алго-

ритму  $M$  на входе  $1^n$ ,  $x$  разрешается задавать решателю только вопросы вида  $1^{q(n)}$ ,  $y$ .

Очевидно, что сводимость Кука сохраняет простые для типичного входа задачи. Действительно, если задача  $(B, \nu_n)$  проста для типичного входа, то в качестве  $S$  можно взять полиномиальный ее решатель, после чего  $M^S$  превратится в полиномиальный решатель задачи  $(A, \mu_n)$ .

Сводимость Кука обобщает сводимость Карпа: сводящая машина выдает  $S(1^{q(n)}, f(1^n, x))$ , где  $S$  ошибается с вероятностью не более  $1/h(n) = 1/(g(n)p(n))$ .

## 2.1 DistNP-полные задачи

Мы говорим, что распределенная задача  $(A, \mu_n)$  принадлежит классу DistNP, если задача  $A$  принадлежит классу NP. Этот класс содержит полные задачи.

**Теорема.** [3] Существует задача в классе DistNP, к которой сводится по Карпу любая другая задача из этого класса.

**Доказательство.** Если спарить любую NP-полную задачу  $A$  с наибольшим генерируемым распределением  $\mu_n$ , то получится распределенная задача, которая полна. Действительно, пусть дана произвольная задача  $(B, \nu_n)$  из DistNP. Пусть  $f$  — функция, сводящая  $B$  к  $A$ . Рассмотрим распределение вероятностей  $\rho_n = f(\nu_n)$ . Это распределение генерируемо, а значит мажорируется распределением  $\mu_n$ . (Конец доказательства.)

## 2.2 DistNP-полные задачи с вычислительно простым распределением на входах

Недостаток доказательства этой теоремы состоит в том, что наибольшее генерируемое распределение на входах сложно устроено с вычислительной точки зрения. Например, мы не можем по  $n$  и  $x$  даже приблизительно вычислить  $\mu_n(x)$ . Оказывается, этот недостаток можно устранить, и построить полные задачи с очень просто устроенным распределением (почти равномерным).

Объясним, какие распределения мы будем считать просто устроенными. Напомним, что мы называем распределение  $\mu_n$  генерируемым, если существует полиномиальный вероятностный алгоритм  $G$ , для ко-

того

$$\mu_n(x) = P[G(1^n) = x]$$

для всех  $n, x$ . Количество бросаний сделанных алгоритмом  $G(1^n)$  ограничено некоторым полиномом  $p(n)$ . Зафиксируем этот полином. Пусть  $r$  обозначает строку длины  $p(n)$ . Будем через  $G_n(r)$  обозначать выход алгоритма  $G(1^n)$ , если в качестве результатов бросаний ему давать по очереди биты слова  $r$ . Таким образом,  $\mu_n(x)$  есть доля тех  $r$ , для которых  $G_n(r) = x$ .

Множество тех  $r$ , для которых  $G_n(r) = x$ , вообще говоря, может быть устроено довольно сложным образом. Теперь потребуем, чтобы для всех  $n$  и всех  $x$  из носителя  $\mu_n(x)$  это множество было устроено очень просто. А именно, потребуем, чтобы для всех  $n$  и всех  $x$  из носителя  $\mu_n$  множество

$$\{r \in \{0, 1\}^{p(n)} \mid G_n(r) = x\}$$

состояло из всех продолжений некоторого слова, которое мы будем обозначать через  $code(n, x)$ . Потребуем также, чтобы отображение  $(n, x) \mapsto code(n, x)$  было вычислимым за полиномиальное от  $n$  количество шагов. (Если  $x$  не принадлежит носителю  $\mu_n$ , то алгоритм вычисления функции  $code(n, x)$  должен сообщать об этом.)

Нетрудно понять, что если это свойство выполнено для какого-нибудь полинома, ограничивающего длину используемых случайных битов, то оно верно и для всех таких полиномов. Будем распределения, для которых существует генератор с таким свойством, называть *вычислительно простыми*. Из вычислительной простоты распределения следует что, что существует код Шеннона — Фано для распределения  $\mu_n$  с полиномиально (от  $n$ ) вычислимыми кодирующей и декодирующей функциями. (Кодом Шеннона — Фано называется кодирование с длиной кода  $-\log \mu_n(x) + O(1)$ . Такой код существует для любого распределения вероятностей.)

Если существуют генераторы псевдослучайных чисел, то не каждое генерируемое распределение мажорируется вычислительно простым распределением. (Распределения, порождаемые генераторами псевдослучайных чисел, не мажорируются вычислительно простыми распределениями.)

*Отступление.* Данное нами определение вычислительно простых распределений отличается от определения Левина. Во-первых, как уже сказано, по Левину распределение в распределенной задаче одно. Но

главное отличие в том, что по Левину требуется, чтобы функция  $y \mapsto \sum_{x < y} \mu(x)$  была вычислима за полиномиальное время (слова одной длины сравниваются лексикографически, а слова разных длин сравниваются по длине). Для простоты мы считаем, что  $\mu(x)$  принимает только двоично-рациональные значения и алгоритм должен вычислить указанную функцию точно. Требование Левина обеспечивает возможность построения кода Шеннона — Фано с полиномиально вычислимыми кодирующей и декодирующей функциями. Поэтому, для вычислительно простых по Левину распределений, семейство условных распределений  $\mu_n(x)$  (вероятность  $x$  при условии, что длина  $x$  равна  $n$ ) мажорируется вычислительно простым в нашем смысле (если дополнительно известно, что суммарная вероятность всех слов длины  $n$  не меньше  $1/\text{poly}(n)$ ).

Определение Левина обладает следующим недостатком: пусть  $f(x)$  вычислимая и обратимая за полиномиальное время перестановка множества всех слов. Может оказаться, что распределение  $\mu$  вычислительно просто по Левину, а распределение  $\mu(f(x))$  — нет. Вспомним, что в исходными данными для задач обычно являются двоичные коды конечных объектов, отличных от слов, например, пропозициональных формул. Перекодирование из одной системы в другую выполняется вычислимыми и обратимыми за полиномиальное время перестановками. Представляется неправильным, что в одной кодировке распределение на формулах может быть вычислительно простым, а в другой кодировке — нет. Заметим, что наше определение лишено этого недостатка.

Будем обозначать через  $\text{Dist}'\text{NP}$  класс распределенных задач  $(A, \nu_n)$ , у которых  $A$  принадлежит  $\text{NP}$ , а семейство  $\nu_n$  вычислительно просто.

**Теорема.** Существуют  $\text{Dist}'\text{NP}$ -полные задачи (относительно сложности Карпа).

**Доказательство.** Полная задача  $U$  состоит в том, что по тройке  $(1^n, M, x)$  надо узнать, останавливается ли недетерминированная машина Тьюринга с программой  $M$  на входе  $x$  за  $n$  шагов. При разумном выборе способа записи программ это множество принадлежит классу  $\text{NP}$ .

Теперь определим распределение вероятностей  $\mu_n$  на тройках  $(1^n, M, x)$ . Первая компонента тройки берется равной параметру безопасности, затем выбираются независимо длины второй и третьей компонент среди слов длины менее  $n$ , а сами они выбираются равномерно среди слов выбранной длины. То есть вероятность тройки  $(1^n, M, x)$  равна  $n^{-2} 2^{-|M|-|x|}$

при условии, что длины  $M$  и  $x$  меньше  $n$ , и нулю иначе.

Строго говоря, это распределение вероятностей не является простым и даже генерируемым, поскольку число  $n^{-2}2^{-|M|-|x|}$  может не быть двоично рациональным. Однако существует вычислительно простое распределение  $\mu_n$ , которое меньше этого не более, чем вчетверо, которое мы и будем использовать.<sup>1</sup>

Пусть  $(A, \nu_n)$  произвольная распределенная задача из NP. Зафиксируем недетерминированную машину Тьюринга  $M$ , допускающую  $A$ . Обозначим через  $\hat{n}$  запись числа  $n$  в двоичной системе, в которой все биты продублированы, а в конце приписано 01 (например,  $\hat{5} = 11001101$ ). Пусть  $G$  генератор распределения  $\mu_n$ . Рассмотрим машину  $M'$  которая на входе  $\hat{n}q$  вычисляет  $x = G_n(q)$  (если длина  $q$  меньше, чем требуется случайных битов, то недостающие случайные биты полагаются равными, скажем, нулю), а затем запускает  $M$  на  $x$ . Время работы  $M'$  на входе  $\hat{n}q$  ограничено некоторым полиномом от  $n$ , обозначим его через  $p(n)$ .

Сводящий алгоритм на паре  $(1^n, x)$  вычисляет такое слово  $q$ , для которого множество тех  $r$ , для которых  $G_n(r) = x$ , состоит из всех продолжений  $q$ . Поскольку  $\nu_n$  вычислительно простое, такое  $q$  существует и его можно сделать за полиномиальное время. Длина  $q$  очевидно равна  $\log(1/\nu_n(x))$ . Сводящий алгоритм затем выдает тройку  $(1^{p(n)}, M', \hat{n}q)$ . Поскольку  $M'$  на входе  $\hat{n}q$  работает так же, как  $M$  на входе  $x$ , это сведение корректно.

Осталось убедиться, что  $\mu_{p(n)}$ -вероятность тройки  $(1^{p(n)}, M', \hat{n}q)$  отличается от  $\nu_n$ -вероятности слова  $x$  не более чем в полином от  $n$  раз. Без ограничения общности, мы можем считать, что  $p(n)$  больше  $|M'|$  и больше  $|\hat{n}q|$  для всех  $n, x$ . С точностью до полинома от  $n$   $\mu_{p(n)}$ -вероятность тройки  $(1^{p(n)}, M', \hat{n}q)$  равна  $2^{-|M'|-|\hat{n}q|}$ , что (с точностью до полинома) равно просто  $2^{-|q|}$ . А по построению  $2^{-|q|} = \nu_n(x)$ . (Конец доказательства.)

Более того, для большинства NP-полных задач  $B$  существует вычислительно простое распределение  $\nu_n$ , для которого распределенная за-

---

<sup>1</sup>Вычислительно простое распределение  $\mu_n$  порождается таким генератором. Сначала в случайной строке  $r$  считывается  $\lceil \log n \rceil$  битов, которые задают длину записи  $M$ , затем считываются другие  $\lceil \log n \rceil$  битов, которые задают длину  $x$ . После этого считываются  $|M| + |x|$  битов, которые задают  $M$  и  $x$ . Вероятность выдать тройку  $(1^n, M, x)$  будет равна  $2^{-2\lceil \log n \rceil} 2^{-|M|-|x|}$ , что не более, чем вчетверо меньше  $n^{-2}2^{-|M|-|x|}$ .

дача  $(B, \nu_n)$  Dist'NP-полна.

Действительно, как правило, доказательство NP-полноты данной задачи  $B$  устроено так. Для любой NP-задачи  $A$  мы строим сводящую  $A$  к  $B$  инъективную функцию  $f$ , которая не только вычислима, но и обратима за полиномиальное время. Возьмем в качестве  $A$  любую задачу, имеющую вычислительно простую Dist'NP-полную рандомизацию  $(A, \mu_n)$ . Построим сводящую  $A$  к  $B$  функцию с указанными свойствами. Поскольку  $f$  сводит  $(A, \mu_n)$  к  $(B, f(\mu_n))$ , последняя задача тоже Dist'NP-полна. При этом распределение  $f(\mu_n)$  вычислительно просто: кодирующая функция для  $f(\mu_n)$  получается применением кодирующей функции для  $\mu_n$  к обратной для  $f$ .

Тем не менее представляют интерес поиск естественных распределений на входах NP-полных задач, для которых распределенная задача Dist'NP-полна. Для сводимости Карпа такое распределение существует для задачи о замощении, а для сводимости Кука универсальная задача, рассмотренная выше является полной для равномерного распределения на входах.

**Теорема.** Существуют Dist'NP-полные задачи с равномерным распределением на входах (относительно сводимости Кука).

**Доказательство.** Полная задача  $U$  состоит в том, что по слову  $\widehat{M|u|ix}$  надо узнать, останавливается ли недетерминированная машина Тьюринга с программой  $M$  на входе  $x$  за  $|u|$  шагов. Распределение  $\mu_n$  — равномерное распределение на всех строках длины  $n$ .

Доказательство полноты почти повторяет доказательство предыдущей теоремы. Будем использовать обозначения, примененные в том доказательстве. Пусть генератору  $G_n$  требуются случайные строки длины  $s(n)$ .

Рассмотрим следующий сводящий алгоритм: на входе  $x$  длины  $n$  найдем слово  $q$  и выберем случайное  $r$  длины  $s(n) - |q|$  (так что  $G_n(qr) = x$ ) и случайное  $u$  длины  $p(n)$ . Затем запускает данный нам, как оракул, решатель  $S$  задачи  $U$  на слове  $\widehat{M|u|u\hat{n}qr}$  и выдаем его ответ в качестве результата. В качестве параметра безопасности решателю  $S$  даем длину слова  $\widehat{M|u|u\hat{n}qr}$ , то есть  $N = |\widehat{M}| + 2 \log p(n) + p(n) + 2 \log n + 2 + s(n)$ .

Фиксируем  $n$ . Вероятность ошибки сводящего алгоритма (при случайно выбранном  $x$  по мере  $\nu_n$  и случайно выбранных  $u, r$ ) равна условной вероятности того, что  $S$  ошибается на входе  $y$  длины  $N$  при условии, что  $y$  имеет вид  $\widehat{M|u|u\hat{n}z}$ . Вероятность самого условия есть  $1/\text{poly}(n)$ .

Поэтому вероятность ошибки сводящего алгоритма не более, чем в полином раз, превосходит вероятности ошибки  $S$  на случайно выбранном слове длины  $N$ . (Конец доказательства.)

### 2.3 Dist'NP-полнота задачи о замощении.

Пусть зафиксировано некоторое конечное множество цветов  $\{0, 1, 2, \dots, c-1\}$ . Исходным данным в задаче о замощении является набор квадратиков с раскрашенными в эти цвета ребрами и последовательность цветов  $z$ . Надо выяснить, можно ли замостить квадрат размера  $|z| \times |z|$  маленькими квадратиками так, чтобы цвета на смежных ребрах квадратиков совпадали и чтобы последовательность цветов на нижнем крае большого квадрата была равна  $z$ . Любой из исходных квадратиков можно дублировать в любом количестве, но поворачивать и переворачивать квадратик нельзя.

Распределение  $\rho_n$  на входах следующее: мы случайно выбираем набор квадратиков (все  $2^{c^4}$  наборов считаются равновероятными). Затем мы выбираем натуральное число  $a < n$ . Затем выбираем раскраску нижнего края по следующему правилу: первый цвет в  $z$  равен 2, следующие  $a$  цветов выбираются случайно и независимо среди цветов 0,1, а остальные цвета равны 0.

Нетрудно проверить, что указанное распределение вычислительно просто и что задача о замощении принадлежит NP.

**Теорема.** При достаточно большом  $c$  задача о замощении Dist'NP-полна.

**Доказательство.** Пусть  $(A, \nu_n)$  — произвольная задача из Dist'NP. Зафиксируем машину  $M$ , допускающую  $A$ . Без ограничения общности мы можем считать, что время работы  $M$  ограничено полиномом от длины входа при любых ее догадках. Как и в доказательстве предыдущей теоремы, построим машину  $M'$ , которая на входе  $\hat{n}q$  запускает машину  $M$  на входе  $G_n(q)$  (где  $G$  — генератор распределения  $\nu_n$ ). Рассмотрим универсальную машину  $V$ : на входе  $0\widehat{M}'\hat{n}q$  она моделирует работу  $M'$  на входе  $\hat{n}q$  с не более чем полиномиальным замедлением по времени. То есть,  $V$  допускает  $0\widehat{M}'\hat{n}q$  тогда и только тогда, когда  $G_n(q) \in A$ . При этом, время ее работы на входе  $\hat{n}q$  меньше некоторого полинома  $p_A(n)$ .

Изменим немного программу  $V$ : измененная машина действует как старая до тех пор, пока та не остановится. После этого измененная

машина продолжает работу (делая что-нибудь тривиальное), если старая машина выдала положительный ответ и останавливается иначе. Тогда  $G_n(q) \in A$  тогда и только тогда, когда существует вычисление  $V$  на входе  $\hat{n}q$ , длящееся по крайней мере  $p_A(n)$  шагов.

Ниже мы докажем, что для некоторого фиксированного набора цветов существует набор квадратиков  $\alpha$ , со следующим свойством. Если  $z = 2u$ , то квадрат размера  $|z| \times |z|$  можно замостить так, чтобы последовательность цветов на нижнем краю была равна  $z$ , тогда и только тогда, когда машина  $V$  может проработать  $|z|$  шагов при начальном содержимом  $0u$  входной ленты.

Сведение задачи  $(A, \nu_n)$  к задаче о замощении устроено так. Слово  $x$  отображается в пару  $(\alpha, 2\widehat{M'}\hat{n}q00\dots 0)$ , где  $q$  — такое слово длины  $\log(1/\nu_n(x))$ , для которого  $x = G_n(q)$ , а количество нулей равно  $p_A(n) - |2\widehat{M'}\hat{n}q|$ . По построению  $x \in A$  тогда и только тогда, когда  $V$  может проработать на входе  $0\widehat{M'}\hat{n}q$  не менее  $p_A(n)$ . Последнее эквивалентно существованию замощения.

Теперь сравним  $\nu_n(x)$  и  $\rho_{p_A(n)}$ -вероятностью пары  $(\alpha, 2\widehat{M'}\hat{n}q00\dots 0)$ . Последняя с точностью до полинома от  $n$  равна  $2^{-|\widehat{M'}\hat{n}q|}$ , что с точностью до полинома равно  $2^{-|q|} = \nu_n(x)$ .

Осталось построить набор квадратиков  $\alpha$  с нужным свойством. Возьмем множество цветов равным алфавиту  $\Gamma$  конфигураций машины  $V$  (ее внутренний алфавит, объединенный с множеством пар (буква, состояние машины)) плюс множество пар букв из  $\Gamma$ . Квадратики на нижней и верхней сторонах будут раскрашены в цвета из  $\Gamma$ , а на боковых гранях — в цвета из  $\Gamma \times \Gamma$ . При этом цвет нижнего ребра всегда будет второй компонентой цвета левого вертикального ребра и первой компонентой цвета правого вертикального ребра. Цвет верхнего ребра будет равен значению функции перехода машины на тройке, состоящей из первой компоненты цвета левого ребра, цвета нижнего ребра и второй компоненты цвета правого ребра. Пара  $(0, \text{начальное состояние})$  отождествляется с цветом 2. Любое замощение квадрата  $|z| \times |z|$ , в котором цвета нижней стороны равны  $z = 2u$  моделирует  $|z|$  шагов работы  $V$  на входе  $0u$ , и обратно. (Конец доказательства.)

Замечание. Можно разными естественными способами определять, распределение вероятностей для задачи о замощении так, чтобы полученная задача стала полной. Но, к сожалению, не удастся доказать полноту для равномерного распределения на последовательностях цветов

нижнего края.

## 2.4 Сведение произвольной задачи из DistNP к некоторой задаче из Dist'NP

В этом разделе мы докажем следующее утверждение.

**Теорема.** Каждая задача  $(A, \mu_n)$  из DistNP сводится к некоторой задаче  $(B, \nu_n)$  из Dist'NP.

Из этой теоремы следует, что задача о замощении DistNP-полна. В ее доказательство нам понадобятся так называемые задачи поиска. Задача поиска задается полиномиально разрешимым бинарным отношением  $R(x, y)$  на множестве двоичных слов таким, что для некоторого полинома  $p(n)$  для всех  $x$  выполнено  $R(x, y) \Rightarrow |y| = p(|x|)$ . Алгоритм решает эту задачу, если для каждого  $x$  он находит такое  $y$ , для которого  $R(x, y)$  (если он есть), а если такого  $y$  нет, то алгоритм должен сообщить об этом.

Сначала мы сведем произвольную задачу поиска с вычислительно простым распределением на входах к некоторой задаче из DistNP, у которой распределение на входах также вычислительно простое.

**Теорема.** Каждая распределенная задача поиска с вычислительно простым распределением на входах сводится (по Куку) к некоторой задаче из Dist'NP.

**Доказательство.** В доказательстве мы используем универсальное семейство хэш-функций. Что это такое? Семейство функций  $H$  типа  $\{0, 1\}^l \rightarrow \{0, 1\}^k$  называется универсальным семейством хэш-функций, если для любого  $y \in \{0, 1\}^l$  случайная величина  $h(y)$  (где  $h$  выбирается случайно из  $H$ ) равномерно распределена в  $\{0, 1\}^k$  и для любых различных  $y_1, y_2$  случайные величины  $h(y)_1, h(y)_2$  независимы (но тройки могут быть уже зависимы).

Приведем пример такого семейства. Рассмотрим булеву матрицу  $A$  размера  $k \times l$  и булев вектор  $u$  длины  $k$ . Пара  $A, u$  задает линейное отображение  $h(y) = Ay + u$  (все операции выполняются в поле вычетов по модулю 2). При любых  $k, l$  мы получаем полиномиально вычислимое универсальное семейство хэш-функций, которое мы будем обозначать через  $H_{lk}$ .

**Лемма.** Пусть  $y_1 \neq y_2$ . Тогда случайные величины  $Ay_1 + u$  и  $Ay_2 + u$  независимы и равномерно распределены среди строк длины  $k$  (если  $A, u$

выбираются случайно по равномерному распределению).

**Доказательство.** Равномерная распределенность  $h(y)$  очевидна. Докажем независимость. Отображение  $(A, u) \mapsto (Ay_1 + u, Ay_2 + u)$  является линейным отображением из пространства матриц размера  $k \times (l+1)$  в пространств матриц размера  $k \times 2$ . Поэтому все элементы в образе этого отображения имеют одинаковую кратность (равную мощности его ядра). Значит, нам достаточно доказать его сюръективность. Нетрудно видеть, что это отображение состоит в умножении справа на некоторую матрицу  $M$ . Нам нужно доказать, что ранг этой матрицы равен 2. Это следует из того, что столбцы этой матрицы различны и имеют 1 в качестве последней,  $l+1$ -ой, координаты. Лемма доказана.

Мы будем использовать еще и то, что функции из определенного нами семейства полиномиально вычислимы равномерно по  $k, l$ . Это означает, что по описанию функции  $h$  (матрице и вектору) и ее аргументу  $y$  мы можем за полиномиальное время найти  $h(y)$ .

Пусть данная нам задача поиска задается полиномиально разрешимым отношением  $R(x, y)$  таким, что  $R(x, y) \Rightarrow |y| \leq \text{poly}(|x|)$ . Пусть также дано вычислительно простое семейство распределений вероятностей  $\mu_n$ . Сначала предположим, что для всех  $x$  из носителя распределения  $\mu_n$  из  $R(x, y)$  следует, что длина  $y$  равна некоторому фиксированному полиному  $p(n)$ . (Избавиться от этого предположения будет совсем нетрудно.)

Рассмотрим такую задачу разрешения: даны строка  $x$ , функция  $h : \{0, 1\}^l \rightarrow \{0, 1\}^k$  из универсального семейства хэш-функций  $H_{l,k}$  и число  $i \leq l$ . Надо выяснить, найдется ли такое  $y$  длины  $l$ , для которого верно  $R(x, y)$ , у которого  $i$ -ый бит равен 1 и при этом  $h(y) = \mathbf{0}$ . Ясно, что эта задача принадлежит NP. Распределение  $\nu_n$  на входах такое:  $x$  выбирается случайно по распределению  $\mu_n$ , данному нам. Затем мы полагаем  $l = p(n)$ , выбираем случайно число  $1 \leq k \leq l+1$ , выбираем случайно хэш-функцию  $h$  и число  $i \leq l$ .

Очевидно, что описанное распределение вычислительно просто. Установим, что исходная задача поиска сводится к построенной задаче разрешения. Пусть нам дана процедура  $S$  решения задачи разрешения. Мы построим полиномиальную процедуру решения задачи поиска, вероятность ошибки которой (по построенному распределению) в полином от  $n$  раз больше, чем вероятность ошибки процедуры  $S$  (по распределению  $\mu_n$ ).

Пусть для данного  $x$  надо найти  $y$  для которого  $R(x, y)$ . Пусть  $n$  обо-

значает данный нам параметр безопасности. Мы выбираем случайное  $k \leq l = p(n)$  и хэш-функцию  $h$  из семейства  $H_{lk}$ . Затем для каждого  $i \leq l$  применяем процедуру  $S$  к тройке  $(x, i, h)$ . Пусть  $b_i = 0, 1$  — ответ, выданный  $S$ . Формируем вектор  $y = b_1 \dots b_l$  и проверяем, верно ли  $R(x, y)$ . Если верно, то останавливаемся. Иначе повторяем то же самое с новым  $h$ . Если после  $nl$  повторений нужное  $y$  не найдено, то говорим, что его нет.

Оценим вероятность ошибки этого алгоритма. Ошибка для данного  $x$  может быть только в том случае, если существует такое  $y$ , для которого  $R(x, y)$ , но мы его не нашли. Пусть дано любое такое  $x$ . Рассмотрим число  $k$ , для которого количество  $y$ , для которых истинно  $R(x, y)$ , находится в пределах от  $2^k/3$  до  $2^{k+1}/3$ . Докажем, что с вероятностью более  $2/9$  для случайно выбранной хэш-функции  $h \in H_{lk}$  существует ровно одно  $y$ , для которого  $R(x, y)$  и  $h(y) = \mathbf{0}$  (в этом случае вектор  $b_1 \dots b_l$  совпадает с этим единственным  $y$ , если процедура  $S$  не обманула).

**Лемма.** С вероятностью более  $2/9$  существует ровно одно  $y$ , для которого  $R(x, y)$  и  $h(y) = \mathbf{0}$ .

**Доказательство.** Рассмотрим множество  $Y$ , состоящее из тех  $y$ , для которых  $R(x, y)$ . Для каждого  $y \in Y$  рассмотрим событие “ $h(y) = \mathbf{0}$ ”. Если из этого события вычесть аналогичные события для всех  $y' \neq y$ , то мы получим событие “ $y$  является единственным словом в  $Y$ , для которого  $h(y) = \mathbf{0}$ ”. Вероятность этого события не меньше

$$P[h(y) = \mathbf{0}] - \sum_{y' \neq y} P[h(y) = h(y') = \mathbf{0}].$$

Вычитаемое в этой формуле равно  $2^{-k}$  а вычитаемое —  $(|Y| - 1)2^{-2k}$ . Таким образом, вероятность события “ $y$  является единственным словом в  $Y$ , для которого  $h(y) = \mathbf{0}$ ” больше  $2^{-k} - |Y|2^{-2k}$ . Просуммировав по всем  $y \in Y$ , мы получим больше

$$\frac{|Y|}{2^k} - \left(\frac{|Y|}{2^k}\right)^2.$$

По выбору  $k$  число  $|Y|2^{-k}$  находится в пределах от  $1/3$  до  $2/3$ . Минимальное значение многочлена  $t - t^2$  на этом интервале равно  $2/9$ . Поэтому рассматриваемая вероятность больше  $2/9$ . Лемма доказана.

Назовем  $h$  удачной для данного  $x$ , если существует ровно одно  $y$  для которого  $h(y) = \mathbf{0}$  и  $R(x, y)$ . По лемме для каждого  $x$  с вероятностью не менее  $2/9(l + 1)$  случайно выбранная функция  $h$  удачна.

Назовем  $x$  удобным, если вероятность того, что  $S(m, x, i, h)$  дает неправильный ответ не больше  $1/9l^2$ . Вероятность берется по выше определенному распределению на парах  $(i, h)$ .

Вероятность ошибки сводящего алгоритма можно оценить сверху суммой двух величин: общая  $\mu_n$ -вероятность всех неудобных слов и вероятность того, что вход  $x$  является удобным, но все  $nl$  выбранных  $y$  не удовлетворяют свойству  $R(x, y)$ .

Пусть  $\delta$  обозначает  $\nu_n$ -вероятность ошибки процедуры  $S$ . Тогда первое слагаемое (общая  $\mu_n$ -вероятность всех неудобных слов) не превосходит  $\delta \cdot 9l^2$ . Действительно, каждое неудобное слово вносит вклад  $1/9l^2$  в общую вероятность ошибки, и суммарная вероятность всех неудобных слов не может превышать общую вероятность ошибки, деленную на этот минимальный вклад.

Для любого удобного слова  $x$  вероятность того, что хотя бы для одного  $i$  процедура  $S$  ошибается на тройке  $(x, h, i)$ , не больше  $l/9l^2 = 1/9l$ . Поэтому для любого удобного слова вероятность найти  $y$  в любой из  $nl$  попыток не меньше  $2/9(l + 1) - 1/9l \geq 1/9(l + 1)$ . Значит второе слагаемое не больше  $(1 - 1/9(l + 1))^{nl} \approx e^{-n/9}$ .

Для завершения доказательства теоремы осталось только понять, почему мы можем предполагать, что для всех  $x$  из носителя  $\mu_n$  все искомые  $y$  имеют одну длину. Мы знаем, что любое слово из носителя  $\mu_n$  ограничено по длине некоторым полиномом  $p(n)$ , а следовательно длины всех искомых  $y$  ограничены тоже некоторым полиномом  $q(n)$ . Закодируем каждое  $y$  длины не более  $q(n)$  словом длины  $0^j 1y$  длины ровно  $q(n) + 1$  и рассмотрим новую задачу поиска: по слову  $x$  требуется найти такое  $0^j 1y$ , для которого  $R(x, y)$ , и то же самое распределение на входах. Исходная задача сводится к новой, причем новая удовлетворяет требуемому свойству. Теорема доказана.

Нам осталось доказать, что каждая задача из DistNP сводится к некоторой распределенной задаче поиска, распределение на входах которой вычислительно просто. На самом деле мы докажем даже больше — каждая распределенная задача поиска, распределение на входах которой генерируемо, сводится к некоторой распределенной задаче поиска, распределение на входах которой вычислительно просто.

**Теорема.** Каждую распределенную задачу поиска можно свести к

другой распределенной задаче поиска, у которой распределение на входах вычислительно просто.

**Доказательство.** Пусть данная нам задача состоит в поиске по данному  $x$  такого  $y$ , для которого выполнено  $R(x, y)$ . И пусть распределение на входах  $\mu_n$  генерируется полиномиальным алгоритмом  $G$ . Этот алгоритм получает на вход строку из  $n$  единиц и за полиномиальное от  $n$  время выдает на выход строку, имеющую распределение  $\mu_n$ . Через  $q(n)$  обозначим полином, ограничивающий количество случайных битов, требуемых генератору, а через  $G_n(r)$  мы будем обозначать выход  $G$  при условии, что  $r$  есть случайная строка полиномиальной длины  $l = q(n)$ , использованная алгоритмом  $G$ . То есть,  $\mu_n(x)$  равно доле тех  $r$ , для которых  $G_n(r) = x$ . Без ограничения общности мы предположим, что носитель распределения  $\mu_n$  содержит слова некоторой одной полиномиальной длины  $m = p(n)$ .

Сначала изложим идею доказательства. Допустим исходная задача трудна и построим трудную задачу с равномерным распределением. Рассмотрим два “крайних” случая.

Случай 1: задача “по случайному взятому  $x \in \{0, 1\}^m$  найти такое  $r$ , для которого  $G_n(r) = x$ ” трудна. В этом случае мы имеем трудную задачу с равномерным распределением на входах.

Случай 2: функция  $r \mapsto G_n(r)$  однозначна и легко обратима. Тогда задача “по  $r$  найти такое  $y$ , для которого выполнено  $R(G_n(r), y)$ ” трудна для равномерного распределения на входах. Действительно, если бы была возможность по  $r$  найти такое  $y$ , то мы бы могли и по  $x$  находить  $y$ , обращая функцию  $r \mapsto G_n(r)$ . Более того, однозначность отображения  $r \mapsto G_n(r)$  нам на самом деле не нужна. В общем случае, достаточно по  $x$  уметь выбирать равномерно  $r$  из прообраза  $x$  относительно отображения  $G_n$ . В этом случае мы также имеем трудную задачу поиска с равномерным распределением на входах.

Разумеется, это не доказательство, а только идея. Чтобы от нее перейти к доказательству, надо обойти следующие два препятствия. Первое состоит в том, что отображение  $r \mapsto G_n(r)$  может не подпадать ни под первый, ни под второй случай. Вторая сложность в том, что нам нужно не просто доказать существование трудной задачи поиска с простым распределением, исходя из предположения, что данная нам задача трудна. Нам нужно указать конкретную задачу, к которой данная задача сводится.

Первая трудность преодолевается следующим образом. Если ото-

бражение  $r \mapsto G_n(r)$  не подпадает под второй случай, то оказывается, что функция  $(s, g, h) \mapsto (g(G_n(h(s))), g, h)$  подпадает под первый случай. Здесь  $g, h$  случайно выбираемые функции из подходящего универсального семейства хэш-функций. То есть задача “по  $x$  выбрать равномерно элемент из множества тех  $r$ , для которых  $G_n(r) = x$ ” сводится к задаче “по случайно выбранным  $v, g, h$  найти  $s$  для которого  $g(G_n(h(s))) = v$ ”. (Сводить первую задачу ко второй нужно так: для данного  $x$  случайно выбираем  $g, h$ , и получив решение  $s$  второй задачи для исходной тройки  $g(x), g, h$ , выдаем  $h(s)$ .)

Вторая трудность обходится с помощью соединения задач обращения функции  $(s, g, h) \mapsto (g(G_n(h(s))), g, h)$  и задачи “по  $r$  найти такое  $y$  для которого выполнено  $R(G_n(r), y)$ ” в одну задачу. А именно рассмотрим “конъюнкцию” этих задач: если исходное данное является четверкой вида  $(0, g, h, v)$ , то решатель должен найти такое  $s$ , для которого  $g(G_n(h(s))) = v$ , а исходное данное есть пара  $(1, r)$ , то надо найти  $r$ , для которого  $R(G_n(r), y)$ . Как мы собираемся сводить исходную задачу к этой? Пусть нам дано  $x$ , для которого нужно найти  $y$  со свойством  $R(x, y)$ . Выбираем случайные  $h, g$  и даем решателю четверку  $(0, g, h, g(x))$ . Решатель выдаст нам некоторое  $s$ , для которого  $g(G_n(h(s))) = g(x)$ . Если нам повезет и окажется, что  $G_n(h(s)) = x$  (то есть, совпадения хэш-значений у этих двух слов было не случайным), то даем решателю пару  $(1, h(s))$ . Как видно из этого плана, фактически мы собираемся сводить исходную задачу к следующей задаче: по данным  $g, h, v$  найти такие  $s, y$ , для которых  $g(G_n(h(s))) = v$  и  $R(G_n(h(s)), y)$ . Эту задачу мы и рассмотрим.

Итак, перейдем к формальному доказательству. Рассмотрим следующую задачу поиска. Дано натуральное  $n$  и хэш-функции  $h, g$  из универсальных семейств  $H_{k,l}, H_{m,k'}$ , соответственно, и бинарное слово  $v$  длины  $k'$  (здесь  $l = p(n)$  и  $m = q(n)$ , а  $k, k'$  произвольны). (Напомним, что  $q(n)$  обозначает полином, ограничивающий количество случайных битов алгоритма  $G_n$ , а  $p(n)$  обозначает длину выхода алгоритма  $G_n$ .) Надо найти такие слова  $s, x$  длин  $k, m$  и слово  $y$ , для которых  $G_n(h(s)) = x, g(x) = v$  и  $R(x, y)$ . (Слово  $x$  здесь введено для упрощения обозначений, оно однозначно определяется  $n, h, s$ .)

Теперь зададим распределение вероятностей  $\nu_n$  на входах этой задачи. Число  $n$  уже задано, число  $k$  выбирается равномерно среди всех чисел от 0 до  $l + 1$ , а  $k' = k + 1$  (почему  $k'$  надо выбирать именно таким образом, будет видно из дальнейшего). Функции  $g, h$  и слово  $v$  выби-

раются независимо и равномерно в  $H_{k,l}, H_{m,k'}$ , и  $\{0, 1\}^{k'}$ , соответственно (после того, как уже выбрано  $k$ ). Таким образом  $g, h, v$  независимы при любом фиксированном  $k$  (но зависимы, если  $k$  не фиксировано).

Очевидно, что построенное семейство случайных величин вычислительно просто. Докажем, что исходная задача сводится к построенной. Пусть нам дано исходное слово  $x$ , выбранное случайно по распределению  $\mu_n$  (мы знаем  $n$ ). По условию длина  $x$  равна  $m = p(n)$ . Выбираем случайным образом  $k$  от 0 до  $l + 1$  и случайным образом  $h, g$  из семейств  $H_{k,l}, H_{m,k+1}$ . Запускаем решатель второй задачи на  $h, g, g(x)$  (и натуральном числе  $n$ ). Решатель нам возвратит некоторые  $s, x', y$ . Если  $R(x, y)$  истинно, то выдаем  $y$  и останавливаемся, а иначе повторяем то же самое с новыми  $h, g$ . Повторяем эту процедуру  $nl$  раз, и если нам ни разу не повезет, то говорим, что такое  $y$  отсутствует.

Пусть вероятность того, что решатель новой задачи ошибается на входе, взятом случайно по распределению  $\nu_n$ , равна  $\varepsilon$ . Докажем, что вероятность того, что для случайного по распределению  $\mu_n$  входа  $x$  мы не решим задачу поиска (то есть, не найдем существующее  $y$ , для которого  $R(x, y)$ ), ограничена сверху величиной  $O(l\varepsilon)$  плюс экспоненциально малая (от  $n$ ) величина.

Зафиксируем  $x$  из носителя  $\mu_n$  и рассмотрим такое  $k = k(x)$  для которого  $2/3 < 2^k \mu_n(x) \leq 4/3$ . Назовем пару  $(h, g) \in H_{k,l}, H_{m,k+1}$  удачной для  $x$ , если  $k$  лежит в указанных пределах,  $x$  принадлежит образу отображения  $G_n(h(s))$  и для любого другого  $x'$  в образе этого отображения выполнено  $g(x') \neq g(x)$ .

**Лемма 1.** Для любого  $x$  из носителя  $\mu_m$  вероятность того, что случайно взятая пара  $(h, g) \in H_{k,l} \times H_{m,k+1}$  удачна для  $x$ , больше  $2/9$ .

**Доказательство.** Сначала оценим вероятность того, что для случайного  $h$  слово  $x$  содержится в образе отображения  $F_n(h(s))$ . Обозначим через  $R_x$  полный прообраз  $x$  относительно отображения  $r \mapsto G_n(r)$ . По формуле включений исключений эта вероятность не меньше

$$\sum_s P[h(s) \in R_x] - \sum_{s_1 \neq s_2} P[h(s_1) \in R_x, h(s_2) \in R_x].$$

В силу универсальности семейства эта разность равна

$$\sum_s \mu_m(x) - \sum_{s_1 \neq s_2} \mu_m(x)^2 > 2^k \mu_m(x) - (2^k \mu_m(x))^2 / 2.$$

Многочлен  $t - t^2/2$  принимает наибольшее значение в точке 1 и его наименьшее значение на отрезке  $[2/3; 4/3]$  равно  $4/9$ . В силу выбора  $k$ , число  $2^k \mu_m(x)$  принадлежит этому отрезку, поэтому искомая вероятность не меньше  $4/9$ .

Теперь для фиксированного  $h$  оценим вероятность того, что при случайном выборе  $g$  в образе функции  $G_n(h(s))$  имеется  $x' \neq x$ , для которого  $g(x') = g(x)$ . Мощность этого образа ограничена  $2^k$ , а множество значений  $g$  имеет  $2^{k+1}$  элементов. При фиксированном  $x' \neq x$  вероятность события  $g(x') = g(x)$  равна  $2^{-k-1}$ . Умножая ее на количество различных  $x' \neq x$ , получаем  $1/2$ .

Итак, вероятность события указанного в условии леммы больше  $(4/9)(1/2) = 2/9$ . Лемма доказана.

Назовем слово  $x$  удобным (для решения исходной задачи поиска), если для случайных  $g, h$  вероятность того, что пара  $(g, h)$  удачна для  $x$ , но решатель второй задачи ошибается на тройке  $(g, h, g(x))$ , не больше  $1/9(l+1)$ . Для любого удобного слова  $x$  вероятность того, что наш алгоритм правильно угадает то  $k$ , для которого выполнена Лемма 1, и затем выберет удачную пару, не меньше  $2/9(l+1)$ . Поэтому вероятность с одной попытки решить исходную задачу поиска на слове  $x$  не меньше  $2/9(l+1) - 1/9(l+1) = 1/9(l+1)$ . Поэтому вероятность того, что алгоритм не решит ее ни в одной из  $nl$  попыток, не больше  $(1 - 1/9(l+1))^{nl} \approx e^{-n/9}$ .

Для неудобных слов мы ничего не знаем про вероятность правильного решения, поэтому мы просто оценим общую  $\mu_n$ -вероятность всех неудобных слов. Для этого оценим двумя способами вероятность события “пара  $(g, h)$  удачна для  $x$ , но решатель второй задачи ошибается на тройке  $(g, h, g(x))$ ” (слово  $x$  выбирается по распределению  $\mu_m$ , а  $(g, h)$  выбираются, как описано выше). С одной стороны, каждое неудобное слово дает вклад не менее  $1/9(l+1)$  в эту вероятность. Поэтому она не меньше, чем  $p/9(l+1)$ , где  $p$  — общая  $\mu_n$ -вероятность всех неудобных слов. Докажем, что с другой стороны, вероятность этого события не превосходит вероятности события “решатель ошибается на тройке  $(g, h, v)$ ” (по мере  $\nu_n$ , умноженной на меру на случайных бросаниях решателя), умноженной на константу. Для этого достаточно доказать, что это верно при любом фиксированном  $(g, h) \in H_{k,l} \times H_{m,k+1}$ . Зафиксируем пару  $(g, h)$ . Вероятность любого  $x$ , для которого  $(g, h)$  удачна для  $x$ , не превосходит  $(4/3)2^{-k}$ . Кроме того, для каждого  $v$ , существует не более одного такого  $x$ , для которого  $g(x) = v$  и  $(g, h)$  удачна для  $x$ .

Поэтому интересующая нас вероятность оценивается сверху величиной

$$\begin{aligned} & \sum_x \mu_n(x) P[\text{решатель ошибается на } g, h, g(x)] \\ & \leq (4/3) \sum_x 2^{-k} P[\text{решатель ошибается на } g, h, g(x)] \\ & \leq (4/3) \sum_{v \in \{0,1\}^{k+1}} 2^{-k} P[\text{решатель ошибается на } g, h, v] \\ & = (8/3) P[\text{решатель ошибается на } g, h, v]. \end{aligned}$$

В последней строчке мы предполагаем, что  $v$  выбирается случайно из  $\{0, 1\}^{k+1}$ . Поэтому вероятность события “пара  $(g, h)$  удачна для  $x$ , но решатель второй задачи ошибается на тройке  $(g, h, g(x))$ ” не более, чем в  $8/3$  раз превосходит вероятность события “решатель ошибается на тройке  $(g, h, v)$ ”. Поэтому мы имеем неравенство  $p/9(l+1) \leq 8\varepsilon/3$ , то есть  $p \leq 24(l+1)\varepsilon$ . Теорема доказана.

## References

- [1] Shai Ben-David, Benny Chor, Oded Goldreich, Michael Luby, On the Theory of Average Case Complexity. STOC 1989: 204-216
- [2] Russell Impagliazzo, Leonid A. Levin, No Better Ways to Generate Hard NP Instances than Picking Uniformly at Random. FOCS 1990: 812-821
- [3] Leonid A. Levin, Average Case Complete Problems. SIAM J. Comput. 15(1): 285-286 (1986)