Алгоритмические задачи с таблицами значений булевых полиномов[1]

М. Н. Вялый

Аннотация. В данной работе рассматривается алгоритмическая сложность основных задач комбинаторики слов (варианты задачи поиска вхождений подслова в слово) при задании слова сжатым описанием. А именно, в качестве слов, в которых ищутся вхождения подслов, рассматриваются таблицы значений булевых полиномов. Построены эффективные алгоритмы проверки вхождения слова фиксированной длины в таблицу значений полинома фиксированной степени, последовательного порождения вхождений при тех же условиях. Приведены и некоторые другие задачи того же типа, для которых существуют полиномиальные алгоритмы их решения.

В последние годы увеличился интерес к такому давно известному способу задания булевых функций как многочлены Жегалкина (или булевы полиномы, они же полиномы над полем из двух элементов $\mathbb{F}_2$). Как следствие, появились работы [1, 2, 3], в которых изучается комбинаторика таких полиномов и описываются соотношения между алгебраическим и комбинаторными свойствами. Данная работа принадлежит к тому же направлению. В ней изучаются особенности основных задач комбинаторики слов при применении их к таблицам значений булевых полиномов.

Проверка вхождения одного слова в другое, перечисление вхождений и подсчет кратностей вхождений — это основные алгоритмические задачи комбинаторики слов. Нас будут интересовать эти задачи, когда слово, в котором ищутся вхождения вхождения образца, является таблицей значений некоторого булевого полинома. Поскольку любая булева функция однозначно соответствует некоторому булеву полиному, любое слово длины 2^n является таблицей значений некоторого булева полинома. Специфика рассматриваемых здесь задач состоит в том, что предполагается наличие короткого описания для полинома, так что прямое построение таблицы значений требует неприемлемо большого времени.

Есть три основных способа задания булевых полиномов, помимо таблицы значений: списком мономов, формулой и схемой вычисления.

[1]Работа выполнялась при финансовой поддержке проектов РФФИ 02–01–716, 02-01-22001, НЦНИ и гранта поддержки научных школ НШ 1721.2003.1

Моном однозначно определяется двоичным словом $a \in \mathbb{F}_2$ по правилу

$$x^a = x_1^{a_1} x_2^{a_2} \dots x_n^{a_n}, \quad a_j \in \{0,1\}, \quad x^0 = 1,\ x^1 = x. \tag{1}$$

Множеству мономов A соответствует полином $\sum_{a \in A} x^a$, где суммирование производится по модулю 2. Сложностью задания полинома списком мономов естественно считать $|A|$.

Формулы определяются рекурсивно. Константы и переменные — это формулы размера 1. Если F, G — формулы размеров q и r соответственно, то $F + G, FG$ — формулы размера $q + r + 1$.

Схема вычисления размера s — это последовательность булевых функций $f_0, f_1, \dots, f_s$, которая начинается с $1, x_1, x_2, \dots, x_n$ и для каждой f_k, $k > n$, при некоторых $i, j < k$ выполнено $f_k = f_i f_j$ или $f_k = f_i + f_j$.

Основной интересующий нас здесь случай — это полиномы малой степени ($d = O(1)$). В этом случае все три приведенные выше способа задания полинома имеют сопоставимую сложность (полиномиально эквивалентны). В общем случае представление в виде списка мономов дает формулу размера, полиномиально зависящего от числа мономов, а представление в виде формулы — схему, размер которой полиномиален от размера исходной формулы. Таким образом, представление в виде схемы — наиболее сжатое.

Работа устроена следующим образом. В разделе 1 приводятся основные определения и обозначения, используемые в работе. В разделе 2 рассматривается задача вычисления степени слова, которая по определению равна наименьшей степени полинома, таблица значений которого содержит данное слово. Для задачи вычисления степени построен полиномиальный алгоритм. В разделе 3 рассматривается задача проверки вхождения слова в таблицу значений заданного полинома. В общем случае эта задача оказывается NP-полной. В случае полиномов фиксированной степени построен субэкспоненциальный алгоритм проверки вхождения слова. Показатель степени при этом зависит только от степени и длины слова. Таким образом, для проверки вхождения слов фиксированной длины в таблицы значений полиномов фиксированной степени получаем полиномиальный алгоритм. Отдельно рассматривается простейший случай полиномов степени 2. В этом случае для задачи проверки вхождения слова существует алгоритм, полиномиальный и по длине записи слова, и по длине записи полинома. В разделе 4 полученные в предыдущем разделе результаты применяются к задаче последовательного порождения вхождений слова в таблицу значений полинома. Как следствие, построен полиномиальный алгоритм последовательного перебора нулей полинома фиксированной степени. В последнем разделе 5 рассматривается задача вычисления кратности вхождений слова в таблицу значений полинома. Уже для кратности вхождения слова из одного 0 (то есть, попросту для числа нулей полинома) известны результаты, говорящие о трудности этой задачи даже для полиномов степени 3

(см. [8]). Здесь рассматриваются задачи $\mathcal{S}(w)$ подсчета кратности вхождения фиксированного слова w в полиномы третьей степени. Построен полиномиальный алгоритм для задачи $\mathcal{S}(01)$ (фактически, подсчет числа серий в таблице значений как двоичном слове). Приведены примеры слов, для которых задача $\mathcal{S}(w)$ не менее трудна, чем $\mathcal{S}(0)$. На основе рассмотренных примеров сформулирована гипотеза об окончательном виде классификации трудных и простых задач в этом наборе задач.

1. Определения и обозначения

Через $\mathbb{F}_2$ обозначается поле из 2 элементов, через $\mathcal{F}(d,n)$ — полиномы над $\mathbb{F}_2$ (булевы полиномы или полиномы Жегалкина) степени не выше d от n переменных $x_1,\dots,x_n$.

Наборы значений n булевых переменных естественным образом отождествляются с двоичными словами длины n. Для записи двоичных слов (равно как и для соответствующих им наборов переменных) мы будем использовать наряду с обычной и *мультипликативную* запись: $0^{a_1}1^{a_2}\dots$ означает слово, начинающееся с a_1 нулей, за которыми идут a_2 единиц и т.д.

Выписывая значения полинома f в лексикографическом порядке от наименьшего набора 0^n до наибольшего набора 1^n, получаем таблицу значений $T(f)$ полинома f: двоичное слово длины 2^n. Отношение лексикографического порядка будем обозначать $\prec_\ell$.

Степенью слова w назовем наименьшее d, при котором существует булев полином f степени d, таблица значений которого содержит слово w: $T(f)=u_0wu_1$, $u_0,u_1\in\{0,1\}^*$. Степень слова w будем обозначать $\deg w$.

Через $\#_w u$ обозначим число вхождений слова w в слово u, а через $\#_w f$ — количество вхождений слова w в таблицу значений полинома f. В частности, $\#_0 f, \#_1 f$ — количество нулей (соответственно — единиц) полинома f.

2. Вычисление степени слова

В этом разделе мы рассмотрим алгоритмическую задачу вычисления степени слова. Она оказывается простой. Прежде чем описывать алгоритм вычисления степени слова, сделаем несколько предварительных наблюдений.

Пусть f — полином от переменных $x_1,\dots,x_n$. Разобьем переменные на две группы: $u_i=x_i$, $1\le i\le n-k$, и $v_i=x_{i+n-k}$, $1\le i\le k$. Тогда f можно записать в виде

$$f(u,v)=\sum_{a\in\mathbb{F}_2^{n-k}} c_a(u)v^a,\quad \deg f\ge \deg c_a+|a|. \tag{2}$$

Таблица значений $T(f)$ разбивается на блоки длины 2^k:

$$T(f)=T_{0^{n-k}}\dots T_u\dots T_{1^{n-k}}, \tag{3}$$

где T_u — таблица значений полинома $f(u,v)$ при фиксированном u. Аффинная (линейная неоднородная) невырожденная замена переменных u приводит к перестановке блоков, не меняя самих блоков.

Лемма 1. *Пусть a,b — два подряд идущие блока в $T(f)$. Тогда существует такая аффинная замена переменных, что $a=T_{0^{n-k-1}0}$, $b=T_{0^{n-k-1}1}$.*

Доказательство. Аффинные автоморфизмы действуют транзитивно на упорядоченных парах несовпадающих точек пространства $\mathbb{F}_2^n$. □

Теорема 1. *Существует алгоритм, работающий за полиномиальное время, который по слову w вычисляет степень* $\deg w$.

Доказательство. Обозначим $k=\lceil\log_2|w|\rceil$ и рассмотрим разложение (2) относительно k последних переменных. Слово w может попадать не более чем в два последовательных блока этого разложения. Лемма 1 позволяет считать без ограничения общности, что эти два блока — первые блоки разложения. Но это означает, что достаточно проверять вхождения слова w в таблицы значений полиномов от $k+1$ переменной (при ограничении на подпространство степень полинома не возрастает).

Длина таблицы значений полинома от $k+1$ переменной равна $2^{k+1}=O(|w|)$. У слова w есть не более $2^{k+1}-|w|+1$ возможных мест вхождений в таблицу значений. Если зафиксировать пару (i,d), где i — первая позиция вхождения слова w в таблицу значений, а d — степень полинома, то условие существования такого полинома записывается в виде системы линейных уравнений на коэффициенты при его мономах (значение полинома зависит от его коэффициентов линейно). Размер системы полиномиален по $|w|$, а всего систем $O(|w|\log|w|)$.

Таким образом, искомый алгоритм состоит в последовательном решении для всех возможных пар (i,d) системы уравнений $\mathcal{L}(i,d)$, задающей условие вхождения слова w в таблицу значений некоторого полинома из $\mathcal{F}(d,k+1)$, начиная с i-го места. Минимальное d, при котором существует i, для которого система $\mathcal{L}(i,d)$ разрешима и будет степенью w. □

Из приведенного в доказательстве анализа непосредственно извлекается следующее

Следствие 1. *Любое слово w входит в таблицу значений некоторого полинома из $\mathcal{F}(\deg w,\lceil\log_2|w|\rceil+1)$.*

Следствие 2. *Почти все двоичные слова длины n имеют степень* $\Omega(\log n/\log\log n)$.

Доказательство. Из следствия 1 получаем, что слов степени d длины n существует не более $2n2^{(\lceil\log_2 n\rceil+1)^d}$ штук (произведение возможных положений слова длины n на число полиномов в $\mathcal{F}(d,\lceil\log_2|n|\rceil+1)$). Сравнивая с 2^n — общим количеством слов длины n, получаем требуемую оценку. □

Легко привести примеры слов, степень которых отличается от верхней оценки $\lceil\log_2|w|\rceil$ на константу.

Утверждение 1. $\deg 0^k10^k = d$, *где* $k = 2^{d-1}+2^{d-2}-1$.

Доказательство. Пусть 0^k10^k входит в таблицу значений $T(f)$ полинома f. Рассмотрим разложение $T(f)$ на блоки длины 2^{d-1}. При любом расположении слова 0^k10^k относительно блоков найдется как блок вида 0^r10^s, так и блок $0^{2^{d-1}}$. Это означает, что коэффициент $c_{1^{d-1}}(u)$ в разложении (2) по $d-1$ последним переменным не постоянен. Но тогда $\deg f \geq d$. □

3. Проверка вхождения слова в таблицу значений полинома

Как уже упоминалось, полином можно задавать существенно различными по длине записи представления способами. В любом случае, если значение полинома в заданной точке $f(x_1,\ldots,x_n)$ вычисляется по описанию полинома $[f]$ за полиномиальное время от длины описания, задача проверки вхождения заданного слова в таблицу значений заданного полинома принадлежит классу NP. Если не ограничивать степень полинома, то эта задача оказывается NP-полной уже для представления полинома списком мономов.

Итак, рассмотрим алгоритмическую задачу распознавания свойства ПОДСЛОВО, входом которой является некоторое двоичное слово w и список мономов A,

$$A = \{(a_1^{(1)},a_2^{(1)},\ldots,a_n^{(1)}),\ldots,(a_1^{(m)},a_2^{(m)},\ldots,a_n^{(m)})\}.$$

Ответом в задаче является «да», если слово w входит таблицу значений полинома

$$f_A(x) = \sum_{a\in A} x^a,$$

и «нет» в противном случае.

Теорема 2. *Задача* ПОДСЛОВО NP*-полна.*

Доказательство. Сведем к задаче ПОДСЛОВО задачу 3-КНФ. Итак, пусть есть некоторая 3-КНФ $C(u_1,\ldots,u_n)$. Будем считать без ограничения общности, что эта КНФ содержит ровно 2^k дизъюнкций (этого легко добиться простым повторением дизъюнкций, что не влияет на ответ в задаче КНФ). Дизъюнкции будем индексировать наборами k булевых переменных $v=(v_1,\ldots,v_k)$.

Запишем каждую дизъюнкцию $C_{v_1\ldots v_k}$ в виде многочлена третьей степени от переменных $u_1,\ldots,u_n$:

$$C_{v_1\ldots v_k} = \sum_a c_v(a)u^a, \quad |a|\leq 3. \tag{4}$$

Для каждого a коэффициенты $c_v(a)$ можно записать в виде многочлена f_a степени k от переменных v. (Переход от таблицы значений многочлена к его представлению в виде суммы мономов состоит в решении системы линейных уравнений, поэтому осуществим за полиномиальное время.) Теперь введем еще одну булеву переменную z, упорядочим переменные в порядке $u_1,u_2,\ldots,u_n,z,v_1,\ldots,v_k$ и рассмотрим многочлен

$$F_C(u_1,u_2,\ldots,u_n,z,v_1,\ldots,v_k) = (1+z)\sum_{a,|a|\leq 3} f_a(v)u^a. \tag{5}$$

Докажем, что КНФ C выполнима тогда и только тогда, когда слово 1^{2^k} входит в $T(F_C)$. Пусть C выполнима и $(u_1^*,\ldots,u_n^*)$ — выполняющий набор. Тогда подслово $T(F_C)$, начинающееся с позиции u^*00^k и заканчивающееся в позиции u^*01^k, состоит из одних единиц, как следует из (4). Чтобы доказать обратное, заметим, что любой интервал от $u10^k$ до $u11^k$ состоит из одних нулей в силу (5). Таким образом, вхождения 1^{2^k} возможны лишь на интервалах от $u00^k$ до $u01^k$. Вхождение 1^{2^k} в интервал такого вида равносильно выполнимости C на наборе u. □

Если ограничить степень полинома константой, задача ПОДСЛОВО, скорее всего, перестает быть NP-полной. Во всяком случае, для нее есть алгоритм субэкспоненциальной трудоемкости.

Пусть (w,f) — вход задачи ПОДСЛОВО, $|w|=m$, $f\in\mathcal{F}(d,n)$, $d=O(1)$. Аналогично доказательству теоремы 1 рассмотрим разложение (2) относительно k последних переменных, где $k=\lceil\log_2|w|\rceil$. Слово w может попадать не более чем в два последовательных блока этого разложения, индексированных двумя последовательными двоичными словами u_0,u_1 длины $n-k$. Каждый из блоков T_{u_i} должен быть таблицей значений полинома f_i степени $\leq d$ от k переменных. Слово w задает значения t старших позиций таблицы $T(f_0)$ и $m-t$ младших позиций таблицы $T(f_1)$. Обозначим

через r длину общего левого подслова y для слов u_0, u_1. Тогда $u_0 = y01^\alpha$, $u_1 = y10^\alpha$, $\alpha = n - k - r - 1$.

Если зафиксировать r, t, то коэффициенты полиномов f_i станут некоторыми полиномами от y степени не выше d. Эти полиномы легко строятся по f, w, r, t. Всего таких коэффициентов не более $2k^d$ (два полинома из $\mathcal{F}(d,k)$). Поэтому условие вхождения слова w в $T(f)$ при указанных значениях параметров r, t задается системой из не более чем $2k^d$ уравнений степени не выше d на r булевых переменных

$$c_j(y_1, \ldots, y_r) = b_j, \quad 1 \le j \le j_{\max}, \quad j_{\max} \le 2k^d. \tag{6}$$

Разрешимость такой системы равносильна тому, что полином

$$F(y) = \prod_j (c_j(y) + b_j + 1) \tag{7}$$

не равен тождественно 0. Полином не равняется тождественно нулю тогда и только тогда, когда хотя бы один из коэффициентов при его мономах не равен 0. Раскрытие скобок в (6) дает $O(r^{2dk^d})$ слагаемых, коэффициенты при мономах для $F(y)$ получаются из них приведением подобных. Трудоемкость этой операции $n^{O(d \log^d |w|)}$, то есть субэкспоненциальна по длине входа задачи (напомним, что $d = O(1)$).

Заметим также, что возможных значений параметров r, t не более $(n-m)m$. Таким образом, доказано

Утверждение 2. *Существует алгоритм проверки вхождения слова w в таблицу значений полинома степени d трудоемкости $n^{O(d \log^d |w|)}$.*

Особняком стоит случай $d = 2$. В этом случае незначительной модификацией приведенного выше рассуждения можно получить полиномиальный алгоритм проверки вхождения слова.

Утверждение 3. *Существует полиномиальный алгоритм проверки вхождения слова w в таблицу значений полинома степени* 2.

Доказательство. В случае полинома степени 2 степени $c_j(y)$ из системы (6) не превосходят 2. Более того, квадратичные части у всех $c_j(y)$ совпадают (это сумма мономов степени 2 исходного полинома f, составленных из первых r переменных). Поэтому система (6) равносильна системе из одного квадратичного уравнения и нескольких линейных. Проверку разрешимости такой системы можно провести за полиномиальное время. Действительно, найдем базис в подпространстве, задаваемом линейными уравнениями системы, и перепишем в этом базисе единственное квадратичное уравнение. Получим некоторый полином степени 2, заданный набором мономов. Проверка равенства нулю такого полинома выполняется легко.

Всего нужно проверить разрешимость не более чем $(n-|w|)|w|$ таких систем. Таким образом, получаем полиномиальный алгоритм проверки вхождения слова w в таблицу значений полинома степени 2. □

4. Последовательное порождение вхождений слова

Под алгоритмом последовательного порождения вхождений слова w в таблицу значений полинома f мы понимаем такой алгоритм, который получает на вход число $p \in [0..2^n]$ и выдает число $A(p) \in [0..2^n]$ по правилу: $A(0)$ указывает на номер первой позиции первого в лексикографическом порядке вхождения w в $T(f)$ или равен 0, когда вхождений нет; если p — номер первой позиции некоторого вхождения w в $T(f)$, то $A(p)$ — номер первой позиции следующего в лексикографическом порядке вхождения w в $T(f)$ или 0, когда p — номер первой позиции последнего вхождения; во всех остальных случаях $A(p) = 0$.

Оказывается, что последовательное порождение вхождений сводится к задаче проверки существования вхождения. Это сведение основано на следующей лемме, которая уже по существу неявно использовалась выше. Сейчас нам будет удобно называть $\mathbb{F}_2^n$ кубом (булев куб) и называть подкубами подпространства $\mathbb{F}_2^n$, задаваемые уравнениями $x_i = a_i$, $x_{i+1} = a_{i+1}$, $\ldots x_k = a_k$, $1 \le i \le k \le n$. Порядок переменных здесь такой же, какой задает рассматриваемый лексикографический порядок на $\mathbb{F}_2^n$.

Лемма 2. *Любой отрезок лексикографического упорядочения $\mathbb{F}_2^n$ $[a,b] = \{x : a \prec_\ell x \prec_\ell b\}$ представляется в виде дизъюнктного объединения $\le 2n+2$ подкубов.*

Доказательство. Индукция по размерности.

Без ограничения общности можно считать, что a лежит в первой половине, а b — во второй, поскольку и первая, и вторая половины лексикографического порядка — лексикографические порядки на кубе меньшей размерности. Но тогда $[a,b] = [a, 2^{n-1}-1] \cup [2^{n-1}, b]$.

Поскольку обращение лексикографического порядка совпадает с лексикографическим порядком, получающимся после преобразования

$$(x_1, \ldots, x_i, \ldots x_n) \mapsto (x_1 + 1, \ldots, x_i + 1, \ldots x_n + 1) \tag{8}$$

(которое сохраняет множество подкубов), то осталось проверить, что $[0,a]$ разбивается на $\le n+1$ кубов. Пусть $j_1, \ldots, j_s$ — индексы единиц в записи a. Обозначим

$$Q_i = \{x : x_k = a_k, \text{ при } k < j_i, \ x_{j_i} = 0\}.$$

Это подкуб, а из определения лексикографического порядка сразу следует, что $[0,a] = \cup_{i=1}^s Q_i \cup [a,a]$ (если $x \prec_\ell a$ и $x \ne a$, то в некоторой позиции,

на которой в a стоит единица, в x стоит 0; до этой позиции a и x должны совпадать, а что стоит после этой позиции — неважно). □

Обозначим через $\mathcal{I}(w, f)$ предикат вхождения слова w в $T(f)$.

Теорема 3. *Существует алгоритм последовательного порождения вхождений слова w в $T(f)$, $f \in \mathcal{F}(d, n)$, обращающийся к оракулу $\mathcal{I}$ и работающий за полиномиальное время. Количество обращений к оракулу $O(n^2)$, на каждом обращении запрашивается $\mathcal{I}(w, g)$, где степень и число переменных g не превосходят d и n соответственно.*

Доказательство. Очевидно, что последовательное порождение вхождений сводится к нахождению первого в лексикографическом порядке вхождения слова w в отрезок $[p+1, 2^n]$ таблицы значений. Поэтому опишем алгоритм построения первого вхождения в произвольный отрезок $[a, b]$.

Алгоритм состоит в двоичном поиске с обращением к процедуре проверки вхождения w в каждую из частей разбиения. Разобьем отрезок $[a, b]$ на два по возможности равных отрезка $[a, c]$, $[c, b]$. Проверим вхождение w в $[a, c]$. Если вхождение есть, рекурсивно применим алгоритм к отрезку $[a, c]$. Если вхождения нет, сделаем вначале $|w| - 1$ проверку возможных вхождений w на границе отрезков разбиения. Если такие вхождения есть, то алгоритм заканчивает работу, выдавая первое из них. В противном случае алгоритм рекурсивно применяется к отрезку $[c, b]$. Нетрудно видеть, что алгоритм обращается не более $2n$ раз к процедуре вхождения слова w в отрезок, а в остальном работает за полиномиальное время.

Процедура проверки вхождения слова в отрезок $[a, b]$ использует лемму 2. Разобьем отрезок на подкубы в соответствии с леммой 2. Проверим вхождения w на границах подкубов. Таких проверок не более $\leq (2n + 1)(|w| - 1)$ проверок Если вхождений не найдено, найдем для каждого подкуба разбиения ограничение f на этот подкуб (и степень, и число переменных при этом не возрастают) и проконсультируемся с оракулом $\mathcal{I}$ по поводу вхождений w в эти подкубы.

Общее число обращений к оракулу, как нетрудно видеть, $O(n^2)$. □

У леммы 2 и теоремы 3 есть несколько следствий, представляющих самостоятельный интерес.

Следствие 3. *Существует полиномиальный алгоритм, который по полиному f, представленному списком мономов, и точке x определяет наибольший отрезок $[a, b]$ в лексикографическом порядке, который содержит x и на котором полином постоянен ($f|_{[a,b]} = f(x)$).*

Доказательство. По списку мономов полинома легко построить список мономов его ограничения на любой подкуб.

По списку мономов легко проверить, является ли полином константой.

Теперь осталось применить лемму 2 и метод деления пополам из доказательства теоремы 3. □

Следствие 4. *Для любого двоичного слова w, $|w| = O(1)$, и полинома степени $d = O(1)$ существует полиномиальный алгоритм последовательного порождения подслов w, входящих в таблицу значений f.*

Этот результат — прямое следствие утверждения 2 и теоремы 3. Отдельно можно отметить частный случай этого утверждения.

Следствие 5. *Существует полиномиальный алгоритм последовательного порождения нулей полинома.*

5. Подсчет кратности вхождения слова для многочленов степени 2 и 3

Начнем с некоторых предварительных замечаний. При подсчетах часто удобнее использовать разность количества нулей и единиц булева полинома

$$\Delta f = \#_0 f - \#_1 f = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)}. \tag{9}$$

Количество нулей однозначно восстанавливается по $\Delta(f)$ с учетом равенства $\#_0 f + \#_1 f = 2^n$.

Приведем некоторые простые свойства Δf. Во-первых, эта величина не меняется при невырожденных аффинных преобразованиях $\mathbb{F}_2^n$. Во-вторых, при добавлении константы Δf меняет знак:

$$\Delta(1 + f) = \Delta(\neg f) = -\Delta f. \tag{10}$$

Лемма 3. *Пусть $\ell \colon \mathbb{F}_2^n \to \mathbb{F}_2$ — линейный функционал на $\mathbb{F}_2^n$, отличный от константы. Тогда $\Delta \ell = 0$.*

Доказательство. Аффинной заменой можно привести ℓ к виду $\ell(x) = x_1$. Для такого функционала утверждение очевидно. □

Очевидна следующая лемма.

Лемма 4. *Пусть $\ell \colon \mathbb{F}_2^n \to \mathbb{F}_2$ — линейный функционал на $\mathbb{F}_2^n$, отличный от константы. Тогда $\Delta f = \Delta f|_{\ell(x)=0} + \Delta f|_{\ell(x)=1}$.*

Из формулы (9) сразу следует

Лемма 5. *Пусть $f(x, y) = g(x) + h(y)$. Тогда $\Delta f = \Delta g \cdot \Delta h$.*

Хорошо известно, что эти свойства, наряду с теоремой Диксона (см. [4]), позволяют построить полиномиальный алгоритм подсчета Δf (а, следовательно, и $\#_0 f$) для квадратичных функций ($\deg f = 2$).

Теорема Диксона (см., например, [4], [5]) утверждает, что квадратичную булеву функцию можно привести с помощью аффинного невырожденного преобразования к канонической форме

$$c_{t,s}(x) = x_1x_2 + x_3x_4 + \cdots + x_{2t-1}x_{2t} + x_{2t+1} + \cdots + x_{2t+s}. \qquad (11)$$

Используя лемму 5, легко выписать в явном виде $\Delta c_{t,s}$:

$$\Delta c_{t,s} = \begin{cases} 2^t, & \text{если } s = 0, \\ 0, & \text{иначе.} \end{cases} \qquad (12)$$

Доказательство теоремы Диксона конструктивно и по нему непосредственно строится полиномиальный алгоритм приведения к канонической форме.

Итак, справедливо

Утверждение 4. *Существует полиномиальный алгоритм определения $\#_0 f$ при $\deg f = 2$.*

Теорема 4. *Существует полиномиальный алгоритм, который по полиному f, $\deg f = 2$, и слову w вычисляет $\#_w f$.*

Доказательство. Нужно применить утверждение 4 и рассуждение, приведенное в доказательстве утверждения 3. Вхождения w в $T(f)$ параметризуются решениями совокупности $\leq (n - |w|)|w|$ систем (6), которые, как указано в доказательстве утверждения 3, находятся во взаимно однозначном соответствии с нулями некоторой квадратичной функции. □

Замечание 1. *Используя теорему 4 и метод двоичного поиска, описанный в доказательстве теоремы 3, можно построить полиномиальный алгоритм, который по k, w, f, $\deg f = 2$, строит k-е вхождение слова w в $T(f)$.*

Задача определения Δf для $f \in \mathcal{F}(3, n)$ алгоритмически трудна. Уже определение знака Δf в этом случае оказывается полной в классе PP [8]. Класс PP — максимальный класс введенной Феннером, Фортнау и Курцем [7] иерархии перечислительной сложности. Полноту здесь нужно понимать как полноту относительно полиномиальных сводимостей между частично определенными предикатами.

Обозначим через $\mathcal{S}(w)$ задачу вычисления по полиному степени 3 кратности вхождения слова w в таблицу значений этого полинома. Будем называть задачу *трудной*, если к ней сводится по Тьюрингу класс PP, и *простой*, если для задачи есть полиномиальный алгоритм. Из вышесказанного следует, что $\mathcal{S}(0)$, $\mathcal{S}(1)$ — трудные задачи.

Заметим, что трудность или простота задач $\mathcal{S}(w)$, $\mathcal{S}(w+1)$ и $\mathcal{S}(w^{-1})$ (w^{-1} — слово, записанное в обратном порядке) совпадают. Действительно,

$$\#_w f = \#_{w+1}(f+1) = \#_{w^{-1}} Rf, \qquad (13)$$

где R — обращающее лексикографический порядок преобразование (8).

Теорема 5. *Существует полиномиальный алгоритм, который по полиному f степени 3 вычисляет $\#_{01} f + \#_{10} f$.*

Доказательство. Нам нужно посчитать, сколько раз меняется значение полинома при переходах вида

$$w_1 = w0\underbrace{1\ldots1}_{k} \mapsto w_2 = w1\underbrace{0\ldots0}_{k}, \quad k = 0, \ldots, n-1.$$

Разность значений полинома в точках w_1 и w_2 является полиномом второй степени от w. Действительно, после подстановки констант мономы третьей степени зависят только от переменных w и входят одинаковым образом в $f(w_1)$ и $f(w_2)$.

Для завершения доказательства осталось применить утверждение 4. □

Теперь заметим, что $\#_{01} f = \#_{10} f + E(f)$, где $E(f)$ — некоторая величина (краевой эффект), определяемая за полиномиальное время по начальному и конечному куску $T(f)$ длины, равной сумме длин слов, входящих в указанное равенство (в данном случае можно сказать точнее, но это несущественно). Для равенств с точностью до таких «малых эффективных» добавок будем использовать знак $\sim$. Таким образом,

$$\#_{01} f \sim \#_{10} f. \qquad (14)$$

Поэтому $\mathcal{S}(01)$ (и $\mathcal{S}(10)$) — легкие задачи.

С другой стороны, $\#_0 \sim \#_{00} + \#_{01}$, поэтому $\mathcal{S}(00)$ — трудная.

Полная классификация трудных и простых задач $\mathcal{S}(w)$ не построена. Неизвестно также, все ли задачи $\mathcal{S}(w)$ либо простые, либо трудные (даже по модулю разумных сложностных гипотез). Мы приведем некоторые предварительные наблюдения.

Во-первых, если $\deg w > 3$, то $\mathcal{S}(w)$ — простая (нужно писать 0, не читая входа).

Во-вторых, имеется довольно сильное достаточное условие трудности задачи $\mathcal{S}(w)$. Пусть $f \in \mathcal{F}(3, k)$ таков, что w содержится в $T(f)$. Обозначим $a = \#_w \tilde{f}$, $b = \#_w \widetilde{(f+1)}$, $c = \#_w \hat{f}$, $d = \#_w (\hat{f}+1)$, $a_0 = \#_w f$, $b_0 = \#_w (f+1)$, где

$$\begin{aligned} \tilde{f}(x_0, x_1, \ldots, x_n) &= f(x_1, \ldots, x_n), \\ \hat{f}(x_0, x_1, \ldots, x_n) &= x_0 + f(x_1, \ldots, x_n). \end{aligned} \qquad (15)$$

Назовем полином f *подходящим* для w, если

$$a - a_0 \neq b - b_0. \tag{16}$$

Лемма 6. *Если для w существует подходящий полином, то задача $\mathcal{S}(w)$ — трудна.*

Доказательство. Для произвольного полинома $g(u_1, \ldots, u_n)$ рассмотрим полином

$$(g \boxplus f)(u_1, \ldots, u_n, x_1, \ldots, x_k) = g(u_1, \ldots, u_n) + f(x_1, \ldots, x_k). \tag{17}$$

Подсчитывая вхождения w в пары соседних блоков длины 2^k, вычитая пересечения и учитывая равенства $\#_0 f \sim \#_{00} f + \#_{01} f$ и $\#_1 f \sim \#_{10} f + \#_{11} f$, получаем

$$\begin{aligned}\#_w(g \boxplus f) &\sim a\#_{00} g + b\#_{11} g + c\#_{01} g + d\#_{10} g - a_0\#_0 g - b_0\#_1 g \sim \\ &\sim (a - a_0)\#_{00} g + (b - b_0)\#_{11} g + (c - a_0)\#_{01} g + (d - b_0)\#_{10} g. \end{aligned} \tag{18}$$

Если мы имеем оракул для $\mathcal{S}(w)$ и выполнено (16), то мы получаем для чисел $\#_{00} g$ и $\#_{11} g$ систему из двух линейных уравнений с правыми частями, которые могут быть вычислены за полиномиальное время при обращении к оракулу $\mathcal{S}(w)$ (второе уравнение имеет вид $\#_{00} g + \#_{11} g \sim 2^n - \#_{01} g - \#_{10} g$). Таким образом мы можем найти $\#_{00} g$, что и доказывает трудность $\mathcal{S}(w)$. □

Простой и удобный в применении частный случай (16) формулируется так: w входит в $T(f)$, но не входит в $T(\widetilde{f+1})$.

Приведем некоторые вычисления с использованием леммы 6. Для начала рассмотрим задачи $\mathcal{S}(0^r)$. Они все трудные. Действительно, возьмем такое число k, что

$$2^{k-3} < r < 7 \cdot 2^{k-3}. \tag{19}$$

Тогда полином $f(x_1, x_2, \ldots, x_k) = x_1 x_2 x_3$ будет подходящим для 0^r. Действительно, первое неравенство в (19) гарантирует, что $\#_{0^r} \widetilde{f+1} = 0$, а второе гарантирует, что $\#_{0^r} f > 0$.

Задача $\mathcal{S}(001)$ также трудная, потому что

$$\begin{aligned}&\#_{001}(x_1 + 1)x_2 = \#_{001}(0010) = 1 > 0, \\ &\#_{001}(\widetilde{(x_1 + 1)x_2} + 1) = \#_{001}(11011101) = 0.\end{aligned} \tag{20}$$

Отсюда можно заключить, что все задачи $\mathcal{S}(w)$, $|w| = 3$, — трудные. Из равенства $\#_{01} \sim \#_{001} + \#_{101}$ заключаем трудность $\mathcal{S}(101)$. Остальные слова

длины 3, в которые входят и 0, и 1 (для 0^3 и 1^3 трудность уже установлена), эквивалентны рассмотренным в силу (13): $\mathcal{S}(010)$ равносильна $\mathcal{S}(101)$, а $\mathcal{S}(100)$, $\mathcal{S}(011)$, $\mathcal{S}(110)$ равносильны $\mathcal{S}(001)$.

Рассмотрим еще слова длины 4. Для 0^4 и 1^4 трудность уже установлена. Полином с таблицей значений 00101111 (*любой* полином от трех переменных имеет степень не выше 3!) удовлетворяет условиям

$$\#_w(00101111) > 0, \#_w(1101000011010000) = 0 \tag{21}$$

для всех подслов длины 4, т.е. для $w \in \{(0010), (0101), (1011), (0111), (1111)\}$. Отсюда получаем трудность задач $\mathcal{S}(w)$ для всех слов длины 4, за исключением двух эквивалентных пар (1100, 0011) и (0110, 1001). Подходящим для 0110 будет полином с таблицей значений 11000110, так как

$$\#_{0110}(11000110) = 1, \ \#_{0110}(0011100100111001) = 0. \tag{22}$$

Для слова 1100 удобно рассмотреть другой вариант применения условия (16): для полинома с таблицей значений $0^6 1^2$ во введенных выше обозначениях получаем $a = 1$, $a_0 = 0$, $b = b_0 = 0$.

Разобранные примеры позволяют предположить справедливость гипотезы: если для слова w длины больше 2 нет подходящего полинома, то $\deg w > 3$.

Литература

[1] Вялый М.Н., Леонтьев В.К., Осетров М.В. Монотонные булевы полиномы // Дискретный анализ и исследование операций. Серия 1, том 9, N 4, 2002. С. 41-49.

[2] Леонтьев В.К. О некоторых задачах, связанных с булевыми полиномами // ЖВМиМФ, Т. 39, N 6, 1999. С. 1045–1054.

[3] Леонтьев В. К., Морено О. О нулях булевых полиномов // ЖВМиМФ, т. 38, N 9, 1998. С. 1606–1613.

[4] Р. Лидл, Х. Нидерейтер. Конечные поля. М.: Мир, 1989.

[5] Мак-Вильямс Ф. Дж., Слоан Н. Дж. А. Теория кодов, исправляющих ошибки. М.: Связь. 1979.

[6] Eherenfeucht A., Karpinski M. The computational complexity of (XOR,AND)-counting problems. Preprint. Bohn: Iniv. Bohn. 1989.

[7] Fenner S., Fortnow L., Kurtz S. Gap-definable counting classes // J. of Comp. and Sys. Sci. 1994. V. 48. P. 116–148.

[8] Vyalyi M. A comparison of zeroes and ones of a boolean polynomial. 2001. LANL eprint: cs.CC/0111052.