МАТЕМАТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ

УДК 510.52

О ГЕНЕРИЧЕСКОЙ NP-ПОЛНОТЕ ПРОБЛЕМЫ ВЫПОЛНИМОСТИ БУЛЕВЫХ ФОРМУЛ[1]

А. Н. Рыбалов

Омский государственный технический университет, г. Омск, Россия

Вводится понятие полиномиальной генерической сводимости алгоритмических проблем, которое сохраняет свойство разрешимости проблемы для почти всех входов и обладает свойством транзитивности и рефлексивности. Доказывается, что классическая проблема выполнимости булевых формул является полной относительно этой сводимости в генерическом аналоге класса NP.

Ключевые слова: *генерическая сложность, проблема выполнимости, NP-полнота.*

DOI 10.17223/20710410/36/8

ON GENERIC NP-COMPLETENESS OF THE BOOLEAN SATISFIABILITY PROBLEM

A. N. Rybalov

Omsk State Technical University, Omsk, Russia

E-mail: alexander.rybalov@gmail.com

Generic-case approach to algorithmic problems was suggested by Miasnikov, Kapovich, Schupp and Shpilrain in 2003. This approach studies behavior of an algorithm on typical (almost all) inputs and ignores the rest of inputs. Many classical undecidable or hard algorithmic problems become feasible in the generic case. But there are generically hard problems. In this paper we introduce a notion of generic polynomial reducibility algorithmic problems, which preserve the property of polynomial decidability of the problem for almost all inputs and has the property of transitivity. It is proved that the classical satisfiability problem of Boolean formulas is complete with respect to this generic reducibility in the generic analogue of class NP.

Keywords: *generic complexity, Boolean satisfiability problem, NP-completeness.*

Введение

Важнейшим понятием классической теории сложности вычислений является понятие полиномиальной сводимости алгоритмических проблем. С его помощью можно сравнивать проблемы по вычислительной сложности и развивать богатую теорию NP-

[1]Работа поддержана грантом РНФ № 17-11-01117.

полноты [1]. В работе [2] введено понятие полиномиальной сводимости и NP-полноты в среднем. В [3] приведены примеры алгоритмических проблем, которые являются NP-полными в среднем. При анализе вычислительной сложности проблемы в среднем изучается математическое ожидание времени работы алгоритма для всех входов данного размера. В рамках генерического подхода [4] алгоритмическая проблема рассматривается не на всём множестве входов, а на некотором подмножестве «почти всех» входов. Такие входы образуют так называемое генерическое множество. Понятие «почти все» формализуется введением естественной меры на множестве входных данных. С точки зрения практики алгоритмы, быстро решающие проблему на генерическом множестве, так же хороши, как и быстрые алгоритмы для всех входов. Классическим примером такого алгоритма является симплекс-метод — он за полиномиальное время решает задачу линейного программирования для большинства входных данных, но имеет экспоненциальную сложность в худшем случае. Более того, может так оказаться, что проблема трудноразрешима или вообще неразрешима в классическом смысле, но легкоразрешима на генерическом множестве. Для многих классических NP-полных проблем существуют полиномиальные генерические алгоритмы [5].

В данной работе вводится понятие генерической полиномиальной сводимости алгоритмических проблем, которое сохраняет свойство разрешимости проблемы за полиномиальное время для почти всех входов при условии равенства классов $\mathrm{P} = \mathrm{BPP}$, где класс BPP состоит из проблем, разрешимых за полиномиальное время на вероятностных машинах Тьюринга. В работе [6] доказано, что это равенство следует из весьма правдоподобных гипотез о вычислительной сложности некоторых трудных проблем. Далее в работе доказывается, что проблема выполнимости булевых формул [7] является полной относительно генерической полиномиальной сводимости в генерическом аналоге класса NP.

1. Генерические алгоритмы

Пусть I — некоторое множество. Функция $\mathrm{size} : I \to \mathbb{N}$ называется *функцией размера*, если для любого $n \in \mathbb{N}$ множество $I_n = \{x \in I : \mathrm{size}(x) = n\}$ конечно. Например, если $I = \Sigma^*$ — множество слов над конечным алфавитом Σ, то функцией размера будет функция, определённая для любого слова w как его длина $|w|$. Для множества натуральных чисел $\mathbb{N}$ функция размера сопоставляет любому натуральному числу длину его двоичной записи. Как обычно делается в теории вычислимости, будем под алгоритмическими проблемами понимать проблемы распознавания подмножеств из некоторого множества входов с определённой на нем функцией длины. Для подмножества $S \subseteq I$ определим последовательность

$$\rho_n(S) = \frac{|S_n|}{|I_n|},\ n = 1, 2, 3, \ldots,$$

где $S_n = S \cap I_n$ — множество входов из S размера n. Заметим, что $\rho_n(S)$ — это вероятность попасть в S при случайной и равновероятной генерации входов из I_n. *Асимптотической плотностью* S назовем предел (если он существует)

$$\rho(S) = \lim_{n\to\infty} \rho_n(S).$$

Множество S называется *генерическим*, если $\rho(S) = 1$, и *пренебрежимым*, если $\rho(S) = 0$. Очевидно, что S генерическое тогда и только тогда, когда его дополнение $I \setminus S$ пренебрежимо.

Следуя [4], назовём множество S *строго пренебрежимым*, если последовательность $\rho_n(S)$ экспоненциально быстро сходится к нулю, т. е. существуют константы σ, $0 < \sigma < 1$, и $C > 0$, такие, что для любого n

$$\rho_n(S) < C\sigma^n.$$

Теперь S называется *строго генерическим*, если его дополнение $I \setminus S$ строго пренебрежимо.

Алгоритм $\mathcal{A}$ с множеством входов I и множеством выходов $J \cup \{?\}$ $(? \notin J)$ называется (*строго*) *генерическим*, если

1) $\mathcal{A}$ останавливается на всех входах из I;
2) множество $\{x \in I : \mathcal{A}(x) \neq ?\}$ является (строго) генерическим.

Аналогично определяется понятие вероятностного генерического алгоритма. Генерический алгоритм $\mathcal{A}$ вычисляет функцию $f : I \to J$, если $(\mathcal{A}(x) = y \in J) \Rightarrow (f(x) = y)$ для всех $x \in I$. Ситуация $\mathcal{A}(x) = ?$ означает, что $\mathcal{A}$ не может вычислить функцию f на аргументе x. Но условие 2 гарантирует, что $\mathcal{A}$ корректно вычисляет f на почти всех входах (входах из генерического множества). Множество $S \subseteq I$ называется (*строго*) *генерически разрешимым за полиномиальное время*, если существует (строго) генерический полиномиальный алгоритм, вычисляющий его характеристическую функцию. Различие между генерически разрешимыми проблемами и строго генерически разрешимыми проблемами поясняется в работе [8].

2. Генерическая полиномиальная сводимость

Пусть I, J — некоторые множества входов с определёнными на них функциями размера, $P(J)$ — множество всех подмножеств J. Множество $A \subseteq I$ *генерически полиномиально сводится* к множеству $B \subseteq J$ (обозначается $A \leqslant_{\mathrm{Gen}P} B$), если существуют вероятностный полиномиальный алгоритм $\mathcal{R} : I \times \mathbb{N} \to P(J) \cup \{?, !\}$, полином $p(n)$, полином $q(n)$ степени больше 2 и константа $C > 0$, такие, что:

1) для всех $x \in I$ либо $(\forall n\, \mathcal{R}(x, n) = ?)$, либо для всех $n \geqslant q(k)$, где $k = \mathrm{size}(x)$, выполнены следующие условия:
   а) $\forall y \in \mathcal{R}(x, n)$ $(y \neq ! \;\Rightarrow\; \mathrm{size}(y) = n)$;
   б) все элементы в $\mathcal{R}(x, n) \setminus \{!\}$ выдаются алгоритмом $\mathcal{R}$ равновероятно;
   в) вероятность получить ответ ! в $\mathcal{R}(x, n)$ не больше 2^{-Ck};
   г) $\dfrac{|\mathcal{R}(x, n)|}{|J_n|} > \dfrac{1}{(p(n))^k}$;
   д) $x \in A \Rightarrow \mathcal{R}(x, n) \subseteq B$;
   е) $x \notin A \Rightarrow \mathcal{R}(x, n) \subseteq J \setminus B$;
2) множество $\{x \in I : \forall n\ (\mathcal{R}(x, n) = ?)\}$ строго пренебрежимо.

Рассмотрим подробнее каждое из свойств данного определения. Помимо элементов множества J, алгоритм $\mathcal{R}$ может выдавать два особых выхода: ? — неопределённый ответ и ! — неудачная генерация. Вероятностный полиномиальный алгоритм $\mathcal{R}$ (если его выход не равен ?) каждому элементу x множества входов I проблемы A и каждому размеру $n \geqslant q(\mathrm{size}(x))$ сопоставляет достаточно большое подмножество (свойство 1, *г*) $\mathcal{R}(x, n)$ входов J проблемы B размера n (свойство 1, *а*). При этом каждый элемент из $\mathcal{R}(x, n)$, за исключением элемента !, генерируется равновероятно (свойство 1, *б*). Вероятность получить неудачно сгенерированный выход ! экспоненциально мала (свойство 1, *в*). Свойства 1, *д* и 1, *е* гарантируют, что это подмножество полностью попадает в B, если $x \in A$, и полностью лежит вне B для $x \notin A$. Свойство 2 говорит о том, что

алгоритм $\mathcal{R}$ лишь для пренебрежимого множества входов из I может всегда выдавать неопределённый ответ.

Теорема 1. Если $A \leqslant_{\mathrm{Gen}P} B$ и B строго генерически разрешимо за полиномиальное время, то для A существует полиномиальный вероятностный алгоритм, распознающий A на некотором полиномиальном строго генерическом множестве. Таким образом, при условии $\mathrm{P} = \mathrm{BPP}$ множество A строго генерически разрешимо за полиномиальное время.

Доказательство. Пусть $\mathcal{R}$ — полиномиальный вероятностный алгоритм, осуществляющий генерическую полиномиальную сводимость $A \leqslant_{\mathrm{Gen}P} B$; $\mathcal{B}$ — строго генерический полиномиальный алгоритм, вычисляющий характеристическую функцию множества B; $q(n)$ — многочлен из определения сводимости $A \leqslant_{\mathrm{Gen}P} B$. Вероятностный генерический алгоритм $\mathcal{A}$, вычисляющий характеристическую функцию множества A, работает на входе x размера n следующим образом:

1) запустить алгоритм $\mathcal{R}(x, q(n))$, который выдаёт ответ y;
2) если $y = ?$, выдать ?;
3) если $y = !$, выдать НЕТ;
4) иначе запустить алгоритм $\mathcal{B}$ на $y \in J$;
5) если $\mathcal{B}(y) = ?$, выдать НЕТ;
6) иначе выдать ответ $\mathcal{B}(y)$.

Очевидно, что этот вероятностный алгоритм полиномиален. Кроме того, неопределённый ответ он выдаёт на строго генерическом множестве — это следует из свойства 2 определения генерической полиномиальной сводимости. Заметим, что неправильный ответ алгоритм может выдать на шагах 3 и 5. Докажем, что вероятность этого меньше $1/2$. Для шага 3 вероятность меньше 2^{-Cn} по свойству 1, *в* из определения сводимости — меньше $1/4$ при достаточно большом n. Для шага 5 она не больше

$$\frac{|\{y \in J : \mathcal{B}(y) = ?\}_{q(n)}|}{|\mathcal{R}(x, q(n))|} = \frac{|\{y \in J : \mathcal{B}(y) = ?\}_{q(n)}|}{|J_{q(n)}|} \frac{|J_{q(n)}|}{|\mathcal{R}(x, q(n))|}.$$

Напомним, что индекс $q(n)$ внизу означает, что мы рассматриваем в соответствующих множествах только элементы размера $q(n)$. Так как алгоритм $\mathcal{B}$ строго генерический, то существует константа $\alpha > 0$, такая, что

$$\frac{|\{y \in J : \mathcal{B}(y) = ?\}_{q(n)}|}{|J|_{q(n)}} < \frac{1}{2^{\alpha q(n)}}.$$

Из свойства 1, *г* определения сводимости следует, что существует полином $p(n)$, такой, что для любого n

$$\frac{|J|_{q(n)}}{|\mathcal{R}(x, q(n))|} < p(q(n))^n = 2^{n \log(p(q(n)))}.$$

Получаем оценку сверху для вероятности выдачи ответа на шаге 5

$$\frac{2^{n \log(p(q(n)))}}{2^{\alpha q(n)}} = 2^{n \log(p(q(n))) - \alpha q(n)} < 1/4$$

для достаточно больших n, так как многочлен $q(n)$ имеет степень не меньше 2. Итого, получаем, что вероятность неправильного ответа меньше $1/4 + 1/4 = 1/2$. Таким отбразом, вероятностный алгоритм $\mathcal{A}$ работает корректно. ■

Докажем транзитивность полиномиальной генерической сводимости.

Теорема 2. Если $A \leqslant_{\mathrm{Gen}P} B$ и $B \leqslant_{\mathrm{Gen}P} C$, то $A \leqslant_{\mathrm{Gen}P} C$.

Доказательство. Пусть $\mathcal{R}_1$, $\mathcal{R}_2$ — полиномиальные вероятностные алгоритмы, осуществляющие генерические полиномиальные сводимости $A \leqslant_{\mathrm{Gen}P} B$ и $B \leqslant_{\mathrm{Gen}P} C$. Пусть $p_1(n), p_2(n), q_1(n), q_2(n)$ — соответствующие полиномы и C_1, C_2 — соответствующие константы. Вероятностный алгоритм $\mathcal{R}_3$, осуществляющий сводимость $A \leqslant_{\mathrm{Gen}P} C$, работает на входе (x, n), где $n > q_2(q_1(k))$, $k = \mathrm{size}(x)$, следующим образом:

1) запустить алгоритм $\mathcal{R}_1(x, q_1(k))$, который выдаст ответ y;
2) если $y = ?$, выдать ?;
3) если $y = !$, выдать !;
4) иначе запустить алгоритм $\mathcal{R}_2(y, n)$, который выдаст ответ z;
5) если $z = ?$, выдать !;
6) если $z = !$, выдать !;
7) иначе выдать z.

Очевидно, что этот вероятностный алгоритм является полиномиальным. Необходимо проверить, что выполнены все свойства полиномиальной генерической сводимости. Свойство 2 выполнено, так как алгоритм выдаёт неопределённый ответ только на тех элементах x, на которых это делает алгоритм $\mathcal{R}_1$. Свойства $1, а$ и $1, б$ выполняются, так как они выпонены для алгоритма $\mathcal{R}_2$. Свойства $1, д$ и $1, е$ выполняются, так как они верны для алгоритмов $\mathcal{R}_1$ и $\mathcal{R}_2$. Свойство $1, г$ также следует из того, что оно выполнено для алгоритма $\mathcal{R}_2$, при этом соответсвующий полином $p_3(n) = p_2(n)$. Докажем свойство $1, в$. Ответ ! может быть выдан на шагах 3, 5 и 6. Оценим вероятность для каждого шага. Для шага 3 вероятность меньше $2^{-C_1 k}$. На шаге 5 вероятность оценивается как в доказательстве теоремы 1: она не больше

$$2^{k \log(p_1(q_1(k))) - \alpha q_1(k)} < 2^{-C_1 k}$$

для достаточно больших k. Для шага 6 эта вероятность меньше $2^{-C_2 \mathrm{size}(y)} < 2^{-C_2 k}$ по свойству $1, в$ для сводимости $B \leqslant_{\mathrm{Gen}P} C$. Итого получаем оценку сверху

$$2^{-C_1 k} + 2^{-C_1 k} + 2^{-C_2 k} < 2^{-C_3 k}$$

для некоторой константы $C_3 > 0$. ■

3. Генерическая NP-полнота проблемы выполнимости булевых формул

В работе [8] введено представление булевых формул с помощью бинарных деревьев. Это представление более компактно и практично, чем классическое представление булевых формул с помощью таблиц истинности, когда размер таблицы растёт экспоненциально с ростом числа переменных. Представление формул с помощью бинарных деревьев часто используется в программировании различных приложений, связанных с символьными вычислениями. Кроме того, оно удобно для различного рода подсчётов.

Напомним вкратце, в чём заключается этот способ представления. Булевой формуле ϕ в базисе $\{\vee, \wedge, \neg\}$ сопоставляется бинарное дерево T_ϕ, внутренние вершины которого помечены символами $\vee$ и $\wedge$, а листья — переменными или их отрицаниями. С бинарным деревом T_ϕ, имеющим n листьев, связывается множество переменных $\{x_1, \ldots, x_n\}$ так, что каждому листу T_ϕ может быть приписана, вообще говоря, произвольная переменная из множества $\{x_1, \ldots, x_n\}$ либо её отрицание. Под размером $s(\phi)$ формулы ϕ будем понимать число листьев n дерева T_ϕ. В дальнейшем будем отождествлять каждую булеву формулу ϕ с деревом T_ϕ. Обозначим через $\mathcal{F}$ множество всех булевых формул, а через $\mathcal{F}_n$ — множество всех формул в $\mathcal{F}$ размера n.

Лемма 1 [8]. $|\mathcal{F}_n| = 2^{n-1}(2n)^n C_{n-1}$, где $C_n = \dfrac{1}{n+1}\binom{2n}{n}$ — n-е число Каталана.

Для любой формулы ϕ определим множество

$$Eq(\phi) = \{\phi \vee ((x_1 \wedge \neg x_1) \wedge \psi),\ \psi \text{ — произвольная формула}\}.$$

Очевидно, что ϕ выполнима тогда и только тогда, когда любая формула из $Eq(\phi)$ выполнима.

Лемма 2. Для любой формулы ϕ имеет место

$$\frac{|Eq(\phi) \cap \mathcal{F}_n|}{|\mathcal{F}_n|} > \frac{1}{(16(n-2))^k}$$

для любого $n > k+2$, где k — размер формулы ϕ.

Доказательство. Пусть формула ϕ имеет размер k. Тогда для любой формулы $\phi \vee ((x_1 \wedge \neg x_1) \wedge \psi)$ из множества $Eq(\phi) \cap \mathcal{F}_{n+2}$ формула ψ должна иметь размер $n-k$. Кроме того, в этой формуле может участвовать любая из n переменных. Поэтому, аналогично тому, как это делалось в доказательстве леммы 1, можно подсчитать

$$|Eq(\phi) \cap \mathcal{F}_{n+2}| = 2^{n-k-1}(2n)^{n-k} C_{n-k-1}.$$

Отсюда

$$\frac{|Eq(\phi) \cap \mathcal{F}_{n+2}|}{|\mathcal{F}_{n+2}|} = \frac{2^{n-k-1}(2n)^{n-k}C_{n-k-1}}{2^{n-1}(2n)^n C_{n-1}} = \frac{1}{(4n)^k}\frac{C_{n-k-1}}{C_{n-1}} = \frac{1}{(4n)^k}\frac{n}{n-k}\frac{\binom{2(n-k-1)}{n-k-1}}{\binom{2(n-1)}{n-1}} >$$

$$> \frac{1}{(4n)^k}\frac{(n-1)!}{(n-k-1)!}\frac{2(n-k-1)\dots(n-k)}{2(n-1)\dots n} = \frac{1}{(4n)^k}\frac{((n-1)\dots(n-k))^2}{2(n-1)\dots(2n-2k-1)} >$$

$$> \frac{1}{(4n)^k}\Big(\frac{(n-1)\dots(n-k)}{2(n-1)\dots(2n-2k-1)}\Big)^2 > \frac{1}{(4n)^k}\frac{1}{2^{2k}} = \frac{1}{(16n)^k}.$$

Таким образом, имеем $\dfrac{|Eq(\phi)_{n+2}|}{|\mathcal{F}_{n+2}|} > \dfrac{1}{(16n)^k}$, откуда $\dfrac{|Eq(\phi)_n|}{|\mathcal{F}_n|} > \dfrac{1}{(16(n-2))^k}$. ■

Определим генерический аналог класса NP. Множество $S \subseteq I$ принадлежит классу $\operatorname{sg}\mathrm{NP}$, если существует полиномиальное строго генерическое множество $G \subseteq I$, такое, что $S \cap G \in \mathrm{NP}$. Множество $S \in \operatorname{sg}\mathrm{NP}$ называется *генерически NP-полным*, если для любого $A \in \operatorname{sg}\mathrm{NP}$ имеет место $A \leqslant_{\mathrm{Gen}P} S$.

Теорема 3. Проблема выполнимости булевых формул генерически NP-полна.

Доказательство. Пусть $A \in \operatorname{sg}\mathrm{NP}$. Тогда существует полиномиальное строго генерическое множество G, такое, что $A \cap G \in \mathrm{NP}$. Отсюда, по теореме Кука, следует, что существует классическая полиномиальная сводимость f множества $A \cap G$ к проблеме выполнимости. Полиномиальный вероятностный алгоритм $\mathcal{R}$, генерически сводящий проблему A к проблеме выполнимости, работает на входе (x, n) следующим образом:

1) проверяет, принадлежит ли x множеству G;
2) если $x \notin G$, выдаёт ?;
3) если $x \in G$, то строит формулу $\phi = f(x)$, такую, что ϕ выполнима тогда и только тогда, когда $x \in A$;

4) случайно и равновероятно генерирует формулу из множества $Eq(\phi) \cap \mathcal{F}_n$. Как это делается за полиномиальное время, описано в [8].

Проверим свойства полиномиальной сводимости. Свойство 2 следует из того, что множество G строго генерическое. Свойства $1, а, б$ следуют из описания шага 4. Свойство $1, в$ очевидно выполняется: алгоритм вообще не выдаёт ответ !. Свойства $1, д, е$ также выполняются по построению формулы ϕ. Наконец, свойство $1, г$ следует из леммы 2. ■

Автор выражает благодарность рецензенту за полезные замечания и предложения по улучшению текста статьи.

ЛИТЕРАТУРА

1. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982. 419 с.
2. *Levin L.* Average case complete problems // SIAM J. Computing. 1987. V. 15. P. 285–286.
3. *Gurevich Y.* Average case completeness // J. Computer System Sciences. 1991. V. 42. P. 346–398.
4. *Kapovich I., Miasnikov A., Schupp P., and Shpilrain V.* Generic-case complexity, decision problems in group theory and random walks // J. Algebra. 2003. V. 264. No. 2. P. 665–694.
5. *Gilman R., Miasnikov A. G., Myasnikov A. D., and Ushakov A.* Report on generic case complexity // Herald of Omsk University. 2007. Special Issue. P. 103–110.
6. *Impagliazzo R. and Wigderson A.* P = BPP unless E has subexponential circuits: Derandomizing the XOR Lemma // Proc. 29th STOC. El Paso: ACM, 1997. P. 220–229.
7. *Cook S. A.* The complexity of theorem proving procedures // Proc. 3d Annual ACM Symposium on Theory of Computing. N. Y., USA, 1971. P. 151–158.
8. *Рыбалов А.* О генерической сложности проблемы общезначимости булевых формул // Прикладная дискретная математика. 2016. № 2(32). С. 119–126.

REFERENCES

1. *Garey M. and Johnson D.* Computers and Intractability. N. Y., Freeman & Co, 1979. 340 p.
2. *Levin L.* Average case complete problems. SIAM J. Computing, 1987, vol. 15, pp. 285–286.
3. *Gurevich Y.* Average case completeness. J. Computer System Sciences, 1991, vol. 42, pp. 346–398.
4. *Kapovich I., Miasnikov A., Schupp P., and Shpilrain V.* Generic-case complexity, decision problems in group theory and random walks. J. Algebra, 2003, vol. 264, no. 2, pp. 665–694.
5. *Gilman R., Miasnikov A. G., Myasnikov A. D., and Ushakov A.* Report on generic case complexity. Herald of Omsk University, 2007, Special Issue, pp. 103–110.
6. *Impagliazzo R. and Wigderson A.* P = BPP unless E has subexponential circuits: Derandomizing the XOR Lemma. Proc. 29th STOC, El Paso, ACM, 1997, pp. 220–229.
7. *Cook S. A.* The complexity of theorem proving procedures. Proc. 3d Annual ACM Symposium on Theory of Computing, N. Y., USA, 1971, pp. 151–158.
8. *Rybalov A.* O genericheskoy slozhnosti problemy obshcheznachimosti bulevykh formul [On generic complexity of the validity problem for Boolean formulas]. Prikladnaya Diskretnaya Matematika, 2016, no. 2(32), pp. 119–126. (in Russian)