Записки научных
семинаров ПОМИ
Том 277, 2001 г.

**М. А. Всемирнов, Э. А. Гирш,
Е. Я. Данцин, С. В. Иванов**

АЛГОРИТМЫ ДЛЯ ПРОПОЗИЦИОНАЛЬНОЙ ВЫПОЛНИМОСТИ И ВЕРХНИЕ ОЦЕНКИ ИХ СЛОЖНОСТИ

1. Введение

Задача пропозициональной выполнимости (ВЫП) – одна из наиболее естественных $\mathcal{NP}$-полных задач, поэтому ее сложность является принципиально важной для теории сложности вычислений. Поскольку ВЫП – $\mathcal{NP}$-полна, маловероятно, что она может быть решена за полиномиальное время. Тем не менее, важно понять, какое время требуется для решения ВЫП, даже если это время экспоненциально: алгоритм, решающий ВЫП, скажем, за время $2^{n/1000}$, был бы весьма полезен для многих приложений, например, для современных задач разработки микросхем.

Исследования в области алгоритмов для ВЫП включают *экспериментальное* изучение их работы, а также *теоретическое* изучение их сложности. Данный обзор посвящен теоретическому аспекту этих исследований. В нем обсуждаются некоторые из последних алгоритмов для ВЫП, имеющих нетривиальные верхние оценки сложности в наихудшем случае. Также обсуждаются интересные близкие вопросы, например, возможно ли находить выполняющие наборы для одновыполнимых булевых формул быстрее, чем для произвольных выполнимых булевых формул? В работе дается более подробный обзор существующих алгоритмов для ВЫП с рекордными на данный момент верхними оценками времени работы в наихудшем случае. Мы рассматриваем семейства таких алгоритмов и приводим некоторые поясняющие иллюстрации и открытые вопросы.

Новые результаты данной работы включают в себя следующие три пункта: мы приводим упрощенный вариант дерандоми-

---

При поддержке EPSRC, INTAS, NATO, РФФИ (грант 99-01-00113) и гранта 6-го конкурса-экспертизы научных проектов молодых ученых РАН.

зации леммы о кодировании выполняющего набора [38], приводим два новых доказательства леммы Вэлианта-Вазирани [46] и доказываем некоторые результаты, касающиеся алгоритмов для ВЫП, основанных на случайных блужданиях.

Основные определения. Мы рассматриваем алгоритмы для задачи выполнимости булевых формул в конъюнктивной нормальной форме (*КНФ*). Формула в КНФ представляет собой конъюнкцию клозов, клоз есть дизъюнкция литералов, а литерал есть булева переменная или ее отрицание. Задача выполнимости (обозначаемая *ВЫП*) классически формулируется как задача распознавания: для данной булевой формулы F в КНФ выдать ответ “Выполнима”, если существует набор истинностных значений ее переменных, выполняющий каждый клоз F; в противном случае, выдать ответ “Невыполнима”. Однако, мы будем рассматривать ее как задачу поиска выполняющего набора: выдать выполняющий набор (если он существует), в противном случае выдать ответ “Невыполнима”. Ясно, что эти две версии ВЫП эквивалентны с точностью до полиномиального сомножителя, т.е. если одна из них разрешима за время $T(F)$, то другая разрешима за время $\mathrm{poly}(|F|) \cdot T(F)$, где $|F|$ — длина формулы F. На протяжении данной работы запись $f(t) = \mathrm{poly}(t)$ означает, что существует многочлен P, такой что для любого t выполнено неравенство $|f(t)| \leqslant P(t)$. Использование обозначения $\mathrm{poly}(\cdot)$ аналогично использованию символа $O(\cdot)$.

На протяжении данной работы F обозначает входную формулу, $|F|$ – ее длину, т.е. общее число вхождений всех переменных, n – количество переменных. Каждый из наших алгоритмов получает на входе формулу F и выдает либо выполняющий набор значений ее переменных, либо ответ “Невыполнима”.

В том, что касается вероятностных алгоритмов, мы рассматриваем *алгоритмы с односторонней допустимой вероятностью ошибки*, т.е. выдаваемый выполняющий набор всегда является правильным, а ответ “Невыполнима” является правильным с вероятностью, по крайней мере, $\frac{1}{\mathrm{poly}(|F|)}$. Повторяя такой алгоритм полиномиальное количество раз, можно понизить вероятность ошибки, так чтобы она стала меньше любой наперед заданной константы.

Мы обозначаем через $F[x]$ формулу, полученную из формулы F присваиванием значения `true` переменной x, т.е. удалением всех клозов, содержащих x без отрицания и удалением литерала $\neg x$ из оставшихся клозов. Аналогично определяется формула $F[\neg x]$ (переменной x присваивается значение `false`). Наборы представляются множествами литералов. Если положительный литерал x принадлежит набору A, это значит, что значение x в наборе A есть `true`; если $\neg x \in A$, то значение x в наборе A есть `false`. Для набора $A = \{l_1, l_2, \dots, l_t\}$ формула $F[A]$ определяется как $F[l_1][l_2]\dots[l_t]$. Заметим, что набор может быть *частичным*, т.е. значения некоторых переменных могут быть не заданы.

Мы называем *k-клозом* клоз, состоящий ровно из k литералов. Формулой в *k-КНФ* называется формула, содержащая только i-клозы при $i \leqslant k$. Задача выполнимости для формул в k-КНФ обозначается *k-ВЫП*.

Расщепляющие алгоритмы. Многие алгоритмы для ВЫП используют *расщепление*. Под *расщепляющим алгоритмом* мы понимаем алгоритм, который сводит задачу для входной формулы F к задаче для полиномиального числа формул $F_1, \dots, F_p$. Это сведéние может быть детерминированным (рекурсивно вызывать алгоритм для каждой из формул F_i) или вероятностным (случайно выбирать одну из формул F_i). Естественно разделить современные расщепляющие алгоритмы для ВЫП на два семейства: DPLL-подобных и PPSZ-подобных алгоритмов.

DPLL-подобные алгоритмы основаны на процедурах, описанных в работах Дэвиса и Патнема [14] и Дэвиса, Лоджмана и Лавлэнда [13]. Грубо говоря, такой алгоритм заменяет входную формулу F двумя формулами $F[x]$ и $F[\neg x]$, полученными присваиванием некоторой переменной x значений `true` и `false` соответственно. Затем алгоритм упрощает каждую из полученных формул и рекурсивно вызывает процедуру для каждой из упрощенных формул. Основным методом анализа таких алгоритмов являются рекуррентные соотношения для количества листьев в дереве рекурсии. Используя этот метод, Данцин [9] и Мониен и Шпекенмейер [33] получили первые нетривиальные верхние оценки для k-ВЫП. Эта техника имеет простое описание в терминах *векторов расщепления* Кульмана и Люкхардта [29, 31]. Мы описываем общую схему DPLL-подобного алгоритма и использования векторов расщепления в разделе 2. Мы также приводим некоторые эвристики, используемые в современных DPLL-подобных алгоритмах для упрощения формул и выбора переменных для присваивания значений `true` и `false`.

Другое семейство расщепляющих алгоритмов составляют предложенные Патури, Пудлаком, Саксом и Зейном [37, 38] *PPSZ-подобные алгоритмы*. Основное отличие этих алгоритмов от DPLL-подобных алгоритмов заключается в двух моментах: во-первых, в выборе переменных для присваивания значений (случайный выбор в PPSZ-подобных алгоритмах) и, во-вторых, в методах анализа. В отличие от локального анализа DPLL-подобных алгоритмов, анализ PPSZ-подобных алгоритмов основан на глобальных рассуждениях, например, оценке количества переменных, которые никогда не используются для рекурсивных вызовов, поскольку удаляются во время упрощения. Эти алгоритмы существенно используют случайные биты, однако, возможна некоторая их дерандомизация. Мы рассматриваем эти алгоритмы в разделе 3, уделяя особое внимание вопросам дерандомизации.

Формула, имеющая только один выполняющий набор, называется *одновыполнимой формулой*. Одно-ВЫП, т.е. задача нахождения выполняющего набора для одновыполнимой формулы, несколько проще для PPSZ-подобных алгоритмов (или, по крайней мере, для их анализа). Рекордные на данный момент оценки для Одно-ВЫП лучше, чем в общем случае. Это контрастирует со знаменитой леммой Вэлианта-Вазирани [46], согласно которой вероятностные сложности ВЫП и Одно-ВЫП полиномиально связаны. В разделе 4 мы приводим два новых простых доказательства этой леммы и обсуждаем проблему ее применения в нашем контексте.

Алгоритмы, основанные на локальном поиске. Имеется большое семейство алгоритмов, которые решают ВЫП, используя *локальный поиск* (обзор см., напр., в [20]). Типичный алгоритм, основанный локальном поиске, берет начальный набор и изменяет его шаг за шагом, пытаясь приблизиться к выполняющему набору. Если после определенного количества шагов выполняющий набор не найден, алгоритм порождает другой начальный набор и опять изменяет его шаг за шагом. Число таких попыток ограничено; если ни в одной из них не удается найти выполняющий набор, алгоритм заканчивает работу, с ответом “Невыполнима”.

Методы изменения наборов разнообразны. Например, *жадные алгоритмы* (напр., [27, 44]) выбирают переменную и изменяют ее значение в текущем наборе на противоположное, так чтобы значение некоторой функции от набора (например, число клозов, которые он выполняет) возрастало как можно сильнее. Другой метод используется в *алгоритмах, основанных на случайных блужданиях* [39]. Такой алгоритм изменяет на противоположное значение переменной, выбранной случайно из невыполненного клоза. Сложность алгоритмов, основанных на случайных блужданиях, можно оценить, используя их связь с одномерными случайными блужданиями. Основные результаты в области верхних оценок для этих алгоритмов принадлежат Пападимитриу [39] и Шонингу [43]. Как показано в [39], 2-ВЫП может быть решена алгоритмом, основанным на случайных блужданиях, за полиномиальное время. В недавней работе [43] показано, что k-ВЫП может быть решена алгоритмом, основанным на случайных блужданиях, за время $(2-2/k)^n$ с точностью до полиномиального сомножителя. При $k=3$ эта оценка составляет $(4/3)^n$, что является наилучшей из известных верхних оценок для алгоритмов, решающих 3-ВЫП.

Мы обсуждаем алгоритмы, основанные на случайных блужданиях, в разделе 5. В частности, в этом разделе показано, что алгоритм Пападимитриу может быть использован для решения ВЫП за полиномиальное время для формул, изоморфных хорновским. Мы также приводим алгоритм Шонинга и его дерандомизацию [11, 12]. Затем мы модифицируем понятие алгоритмов, основанных на случайных блужданиях, используя недавний подход Фэйджина и др. [15], предложившим *цепи Маркова с "кнопкой возврата"*. А именно, мы определяем алгоритмы, основанные на случайных блужданиях с "кнопкой возврата", и сравниваем их сложность со сложностью обычных алгоритмов случайного блуждания. Алгоритмы, основанные на случайных блужданиях с "кнопкой возврата", могут рассматриваться как некая комбинация подхода, использующего случайные блуждания, и подхода, использующего расщепление.

Близкие вопросы. Исследования в области верхних оценок в наихудшем случае для трудных задач не ограничиваются ВЫП. В прошедшие годы много внимания уделялось и другим $\mathcal{NP}$-полным задачам (например, МАКС-ВЫП [35], МАКС-2-ВЫП, Максимальное сечение [19], 3-Раскрашиваемость [3], Максимальное независимое множество [2, 41]). Однако, в упомянутых работах используются, в основном, DPLL-подобные методы; вызывает интерес, можно ли получить новые верхние оценки для таких задач с помощью PPSZ-подобных методов или методов, использующих локальный поиск.

Известно, что степень полиномиального по времени приближения для некоторых трудных задач имеет некоторые границы в предположении, что $\mathcal{P} \neq \mathcal{NP}$ (см., напр., [1]). В частности, для МАКС-3-ВЫП не существует полиномиального алгоритма (если $\mathcal{P} \neq \mathcal{NP}$), который находил бы набор, выполняющий $\geqslant (\frac{7}{8} + \epsilon)M$ клозов, где M – максимально возможное число одновременно выполнимых клозов, а $\epsilon > 0$ – сколь угодно малое число. Однако, существуют алгоритмы (см., напр., [10, 23]), которые находят такие приближенные решения быстрее, чем наилучшие на данный момент алгоритмы находят точные решения; в последней работе используется алгоритм, основанный на случайных блужданиях, аналогичный алгоритмам из [39, 43].

Благодарности. Авторы признательны Наталии Цилевич за помощь в доказательстве теоремы 5.5 и Диме Пасечнику за плодотворные обсуждения, касающиеся раздела 5.

2. DPLL-подобные алгоритмы

Общая схема. Алгоритмы, описанные в работах Дэвиса и Патнема [14] и Дэвиса, Лоджмана и Лавлэнда [13], принято считать первыми алгоритмами для ВЫП. Большинство алгоритмов, предложенных на протяжении последующих сорока лет, основаны на алгоритмах из [13, 14]. Современное описание таких *DPLL-подобных алгоритмов* выглядит следующим образом:

Процедура 2.1.

1. Упростить входную формулу F, т.е. преобразовать F в некоторую другую формулу G, используя некоторые *правила преобразования*.
2. Если задача выполнимости для G тривиальна, выдать ответ.
3. Выбрать переменную v, входящую в G, используя некоторую *эвристику*. Построить формулы $G[v]$ и $G[\neg v]$ и рекурсивно вызвать данную процедуру для каждой из них. Если хотя бы один из рекурсивных вызовов вернул выполняющий набор, обновить его, добавив v или $\neg v$ соответственно, и вернуть результат (обновление может также включать изменения, вызванные использованием правил преобразования). В противном случае, выдать ответ “Невыполнима”. •

Таким образом, эта процедура имеет следующие параметры:

1. Правила преобразования, служащие для упрощения формул (предполагается, что упрощение занимает полиномиальное время).
2. Эвристика для выбора переменной для расщепления (также за полиномиальное время).

Известно огромное количество различных правил преобразования и эвристик. Простой (но иногда трудоемкий) метод анализа содержится в следующем наблюдении.

Векторы расщепления. Работа процедуры 2.1 может быть представлена с помощью *дерева расщепления*. Его корень помечен формулой, полученной в результате упрощения входной формулы F. Если вершина дерева помечена формулой G, то два ее сына помечены формулами, полученными в результате упрощения формул $G[v]$ и $G[\neg v]$. Листья помечены тривиальными (не содержащими переменных) формулами.

Если имеется дерево расщепления, то для каждой его вершины можно записать рекуррентное неравенство для верхней оценки $T(n)$ времени работы процедуры 2.1. Тривиальным образом, можно записать

$$T(n) \leqslant 2 \cdot T(n-1) + \mathrm{poly}(|F|),$$

поскольку расщепление уменьшает количество переменных, по крайней мере, на единицу. Обычно возможно написать неравенство "получше". Его "качество" зависит от

1. Правил преобразования.
2. Выбора переменной для расщепления.
3. Синтаксических свойств формулы, которой помечена вершина (ясно, что 1 и 2 также влияют на эти свойства).

Вообще говоря, мы можем заменить простое расщепление

$$G \longrightarrow G[v],\ G[\neg v]$$

чем-нибудь более сложным, например,

$$G \longrightarrow G[v,w],\ G[\neg v,w],\ G[v,\neg w],\ G[\neg v,\neg w].$$

Если мы сможем доказать какое-нибудь специальное утверждение о конкретной формуле G, например, что невыполнимость формул $G[v, \neg w]$ и $G[\neg v, w]$ влечет невыполнимость формул $G[v, w]$ и $G[\neg v, \neg w]$, то тогда мы сможем использовать расщепления вида

$$G \longrightarrow G[v, \neg w],\ G[\neg v, w],$$

получая все более и более сложные рекуррентные неравенства.

Замечательная техника для работы с такими рекуррентными неравенствами была развита Кульманом и Люкхардтом [29, 31]. Вместо рекуррентного неравенства каждой вершине дерева расщепления приписывается *вектор расщепления*. Например, если нас интересует сложность относительно числа переменных в формуле, вектор расщепления образуется следующим образом. Рассмотрим расщепление

$$G \longrightarrow G_1, \ldots, G_d$$

формулы, содержащей n переменных, на формулы $G_1, \ldots, G_d$, содержащие $n_1, \ldots, n_d$ переменных соответственно. Тогда вектор расщепления $(t_1, \ldots, t_d)$ состоит из произвольных чисел $t_i \leqslant n - n_i$. Соответствующее *число расщепления* есть единственное решение уравнения

$$\sum_{i=1}^{d} x^{-t_i} = 1$$

на интервале $(0, +\infty)$. Тогда время работы процедуры 2.1 ограничено сверху величиной $\mathrm{poly}(|F|) \cdot \tau^n$, где τ – наибольшее из чисел расщепления для всех вершин нашего дерева. Оценка $\mathrm{poly}(|F|) \cdot 1.505^n$ для 3-ВЫП из работы [29], полученная таким образом, являлась рекордной среди детерминированных алгоритмов в течение шести лет[1].

Аналогичным образом можно оценивать время работы относительно числа m клозов во входной формуле, относительно общего числа l вхождений всех переменных и относительно любой другой “разумной” меры сложности входной формулы. Рекордные на данный момент оценки относительно параметров m и l равны 1.239^m и 1.074^l с точностью до полиномиального множителя [22]. Они имеют место для произвольных формул в КНФ, а не только в 3-КНФ.

[1]Первый препринт появился в 1994 году.

Правила преобразования. Приведем список некоторых правил преобразования, используемых в DPLL-подобных алгоритмах. Читателя, интересующегося дальнейшими подробностями, мы отсылаем к работам [28, 30, 31].

(2.1) **Удаление единичных клозов.** Если F содержит клоз, состоящий из единственного литерала l, положить значение l равным `true`.

(2.2) **Чистый литерал.** Если F содержит *чистый* литерал, т.е. литерал l, такой что его отрицание не входит в F, положить значение l равным `true`.

(2.3) **Резолюция.** Пусть x – переменная из F. Добавить к F все резольвенты по x и удалить из F все клозы, содержащие x или его отрицание (резольвентой двух клозов $x \vee l_{11} \vee \ldots \vee l_{1s}$ и $\neg x \vee l_{21} \vee \ldots \vee l_{2t}$ по x называется клоз $l_{11} \vee \ldots \vee l_{1s} \vee l_{21} \vee \ldots \vee l_{2t}$, если $l_{1i} \neq \neg l_{2j}$ для всех i, j, и `true` в противном случае).

(2.4) **Поглощение.** Если F содержит два клоза $C \subseteq D$, удалить D.

(2.5) **Автаркность.** Если существует (частичный) набор A, такой что $F[A]$ не содержит клозов, не входящих в формулу F, заменить F на $F[A]$.

(2.6) **Черные и белые литералы.** Пусть P – некоторое свойство формул и литералов, вычислимое за полиномиальное время. Предположим также, что свойства $P(F, x)$ (“x – белый литерал, $\neg x$ – черный литерал”) и $P(F, \neg x)$ (“$\neg x$ – белый литерал, x – черный литерал”) не могут выполняться одновременно. Если каждый клоз F, который содержит литерал l, удовлетворяющий $P(F, l)$, содержит также литерал l', удовлетворяющий $P(F, \neg l')$, заменить F на $F[\{l' \mid P(F, \neg l')\}]$.

(2.7) **Заблокированный клоз.** Клоз C называется *заблокированным*, если он содержит литерал l, такой что каждый клоз F, содержащий $\neg l$, содержит также отрицание некоторого другого литерала из C. Любой заблокированный клоз может быть удален из F.

Выбор переменной. Основная эвристика для выбора переменной заключается в следующем: “выбрать переменную, соответствующую наименьшему числу расщепления”. Хотя возможно выяснить этот факт за полиномиальное время, эта эвристика не выглядит слишком практичной. На самом деле, в большинстве случаев можно заменить эту эвристику на что-нибудь более конструктивное. Простейшими примерами являются эвристики "выбрать переменную, входящую в самый короткий клоз" и "выбрать переменную, входящую в наибольшее количество клозов". В типичном случае анализ расщепляющего алгоритма содержит длинный список случаев, соответствующих различным синтаксическим свойствам формул, и эвристика состоит в том, чтобы выбрать переменную, удовлетворяющую хотя бы одному из этих свойств.

Открытый вопрос. Как было упомянуто выше, типичный анализ расщепляющего алгоритма состоит из длинного списка случаев. Доказательства, соответствующие этим случаям, представляют собой очень простые комбинаторные рассуждения. Было бы интересно разработать программу, которая автоматически порождала бы такой список (т.е. доказательство) для данного искомого числа расщепления.

3. PPSZ-подобные алгоритмы и дерандомизация леммы о кодировании выполняющего набора

Лемма о кодировании выполняющего набора. DPLL-подобные алгоритмы определяют переменную для расщепления на каждом шаге, используя только "локальные" свойства формулы. В разных ветвях дерева расщепления переменные выбираются независимо. Оценка времени работы получается из решений рекуррентных неравенств или из чисел расщепления для каждой из вершин дерева расщепления.

Патури, Пудлак и Зейн [38] предложили иной ("глобальный") метод оценки времени работы алгоритмов расщепления. Их алгоритм выбирает случайную перестановку переменных исходной формулы и производит расщепления в соответствующем порядке. Однако, расщеплять по *всем* переменным может не потребоваться, поскольку значения некоторых переменных в выполняющем наборе могут быть определены в результате применения правил преобразования, например, в результате удаления единичных клозов. Анализ работы такого алгоритма основан на оценке количества переменных, не требующих расщепления. Если эта оценка равна s, то мы можем ограничиться рассмотрением деревьев расщепления глубины не более $n-s$. Если выполняющий набор не найден в результате построения дерева глубины $n - s$, то формула невыполнима.

Имеется несколько вариантов алгоритма из [38], некоторые из них имеют наилучшие на данный момент верхние оценки [37]. Наиболее простой вариант этого алгоритма можно представить следующим образом (для простоты мы предполагаем $k = 3$):

Алгоритм 3.1.

1. Выбрать случайным образом перестановку π из S_n, где S_n означает множество всех перестановок $\{1, \ldots, n\}$.
2. Используя процедуру 2.1 с единственным правилом преобразования (2.1), построить дерево расщепления глубины[2] не более $2n/3$. На каждом шаге расщепления выбирать переменную x_i таким образом, чтобы x_i входила в формулу и $\pi(i)$ было наименьшим. Если выполняющий набор найден при помощи процедуры 2.1, выдать его и остановиться; в противном случае выдать ответ "Невыполнима". •

Очевидно, время работы этого алгоритма равно $\mathrm{poly}(|F|) \cdot 2^{2n/3}$. Доказательство того, что этот алгоритм имеет допустимую вероятность ошибки для формул с не более чем одним выполняющим набором, следует из *леммы о кодировании выполняющего набора,* описанной ниже.

Рассмотрим для входной формулы F дерево расщепления, построенное таким же образом, как в алгоритме 3.1, однако, без ограничения $2n/3$ на его глубину. Если S – выполняющий набор для F, то это дерево содержит ветвь, ведущую в S. Каждое расщепление вдоль этой ветви присваивает значение переменной. Пусть S' – набор, отвечающий всем таким прсваиваниям. Подчеркнем, что значения переменных, определенные при помощи правил преобразования, не включаются в S'. Назовем S' *описанием* S. Под *длиной* описания мы понимаем количество литералов в S'. Если мы знаем описание, то мы можем восстановить набор, используя правило преобразования (2.1).

Лемма о кодировании выполняющего набора [38]. *Пусть F – одновыполнимая формула в 3-КНФ, а S – ее выполняющий набор. Рассмотрим описание относительно перестановки, случай-*

[2] Ясно, что процедура 2.1 легко может быть модифицирована таким образом, чтобы она выдавала `false`, если глубина рекурсии больше $2n/3$.

но и равновероятно выбранной из симметрической группы S_n. *Тогда математическое ожидание длины описания* S *не превосходит* $2n/3$.

Доказательство. Для каждой переменной x из F определим *x-критический клоз* как клоз, в котором выполняющий набор делает истинным литерал, соответствующий x, и ложными – все остальные литералы этого клоза. Заметим, что для каждого x формула F содержит, по крайней мере, один x-критический клоз (в противном случае, изменяя значение переменной x в наборе S, мы получили бы другой выполняющий набор). Поскольку мы выбираем перестановку π случайно и равновероятно, переменная x окажется последней переменной (относительно π) x-критического клоза C с вероятностью не менее $1/3$. Нетрудно видеть, что последняя переменная C не появится в описании S. Следовательно, переменная x не попадет в описание выполняющего набора с вероятностью, по крайней мере, $1/3$. Теперь утверждение леммы следует из линейности математического ожидания. •

Пусть p_ℓ – вероятность того, что длина описания выполняющего набора в точности ℓ. По лемме о кодировании выполняющего набора

$$\frac{2n}{3} \geqslant \sum_{l=0}^{n} p_\ell \ell \geqslant \left(\left\lfloor \frac{2n}{3} \right\rfloor + 1 \right) \sum_{\ell=\lfloor 2n/3 \rfloor + 1}^{n} p_\ell .$$

Следовательно,

$$\sum_{\ell=\lfloor 2n/3 \rfloor + 1}^{n} p_\ell \leqslant \frac{2n}{3} \left(\left\lfloor \frac{2n}{3} \right\rfloor + 1 \right)^{-1} \leqslant \frac{2n}{3} \cdot \frac{3}{2n+1} = 1 - \frac{1}{2n+1}.$$

Значит,

$$\sum_{\ell=0}^{\lfloor 2n/3 \rfloor} p_\ell \geqslant \frac{1}{2n+1},$$

т.е. для случайной перестановки длина описания не превосходит $2n/3$ с вероятностью, составляющей, по крайней мере, $1/(2n+1)$. Таким образом, алгоритм 3.1 имеет допустимую вероятность ошибки.

Дерандомизация леммы о кодировании выполняющего набора. Как избавиться от использования случайных битов в алгоритме 3.1? Один из способов состоит в том, чтобы найти "небольшое" множество перестановок B_n, для которого лемма о кодировании выполняющего набора по-прежнему выполняется. В этом случае нужную перестановку можно будет найти перебором множества B_n, а не множества S_n всех перестановок. Для построения такого множества [38] использует пространство случайных переменных, независимых по три. Этот метод дает множество перестановок, "почти независимых по три". У этого подхода имеется два недостатка.

1. Полученное множество перестановок имеет мощность $O(n^9)$.
2. Вследствие "почти" независимости (вместо "обычной" независимости) время работы алгоритма еще более увеличивается.

Хотя эти недостатки и не влияют на теоретическую оценку $\mathrm{poly}(|F|) \cdot 2^{2n/3}$ времени работы детерминированного алгоритма, они делают его бесполезным на практике, поскольку огромный полиномиальный сомножитель делает эту оценку хуже тривиальной 2^n, по крайней мере, при $n \leqslant 200$.

Ниже мы предлагаем другую (и даже более простую) явную конструкцию множества перестановок B_n, мощность которого составляет всего $O(n^3)$. Заметим, что следующее условие достаточно для выполнения леммы: для любых трех различных натуральных чисел $x, y, z \leqslant n$ и для случайной перестановки π из B_n, вероятность того, что $\pi(x) = \max\{\pi(x), \pi(y), \pi(z)\}$, составляет, по крайней мере, $1/3$. Это частный случай "перестановок, независимых по три относительно максимума" [4][3].

Статья [4] не дает примера такого семейства полиномиального размера; конструкция полиномиального размера дана только в том случае, когда равномерное распределение на множестве перестановок заменяется на некоторое неравномерное. Однако, это множество (и распределение на нем) получается в результате решения линейной программы экспоненциального размера. В [4] имеются также приближенные конструкции, но они довольно сложны. Кроме того, за счет их приближенного характера полиномиальный сомножитель времени работы получаемого в ито-

[3] В [4] рассматривается симметричный случай перестановок, независимых относительно минимума (min-3-wise).

ге алгоритма оказывается больше, чем соответствующий множитель в приведенной ниже конструкции. В нашей конструкции B_n будет мультимножеством, хотя, очевидно, что для целей нашего алгоритма достаточно перебирать различные перестановки из B_n лишь по одному разу.

Пусть p – наименьшее простое число, такое, что $n \leqslant p + 1$. Ясно, что $p \leqslant 2n$. В дальнейшем $\mathbb{Z}_p$ обозначает конечное поле из p элементов. Рассмотрим проективную прямую P над $\mathbb{Z}_p$, т.е. классы элементов $(\mathbb{Z}_p \times \mathbb{Z}_p) \setminus \{(0, 0)\}$ относительно эквивалентности

$$(x, y) \sim (x', y') \text{ если и только если } xy' = x'y.$$

Рассмотрим перестановки ее точек. Проективная прямая содержит $p+1$ элементов, поэтому такие перестановки можно отождествить с элементами симметрической группы S_{p+1}. Мы найдем подмножество B_{p+1} группы S_{p+1}, независимое по три относительно максимума. Искомое мультимножество B_n перестановок n элементов получится в результате ограничения[4] перестановок из B_{p+1} на первые n элементов. Ясно, что свойство независимости относительно максимума сохраняется.

В качестве множества B_{p+1} возьмем группу $PGL(2, p)$ проективных автоморфизмов P, т.е. отображений на P, индуцированных обратимыми линейными преобразованиями $\mathbb{Z}_p \times \mathbb{Z}_p$: $f((x, y)) = (ax + by, cx + dy)$ при условии $ad - bc \neq 0$. Имеется в точности $(p + 1)p(p - 1)$ различных проективных автоморфизмов. Хорошо известно, что для любых двух упорядоченных троек попарно различных точек проективной прямой над полем существует единственный проективный автоморфизм, переводящий первую тройку во вторую (см., например, [26, часть 3, §8, пп. 8, 9]). В частности, под действием случайной перестановки из B_{p+1} каждая тройка различных элементов P переходит в любую другую равновероятно. Следовательно, наша конструкция корректна.

Замечание. *Для любых* $n \geqslant k \geqslant 2$ *существует мультимножество* $C_{n,k}$ *независимых по* k *относительно максимума (или, ана-*

[4] Под ограничением перестановки π на первые n элементов мы подразумеваем единственную перестановку $\pi' \in S_n$, такую что для всех $x, y \in [1..n]$ неравенство $\pi'(x) < \pi'(y)$ имеет место в том и только в том случае, когда $\pi(x) < \pi(y)$. А именно, $\pi'(k)$ есть мощность множества $\{\pi(1), \ldots, \pi(n)\} \cap [1..\pi(k)]$.

логично, относительно минимума) перестановок n элементов, такое что мощность $C_{n,k}$ не превосходит $n^{(1+1/\log n)k}\mathrm{lcm}(1,\ldots,k)$; см. [25]. Более тщательный анализ действия группы $PGL(2,p)$ на четверках точек показывает, что построенное мультимножество B_n размера $O(n^3)$ будет даже независимым по четыре относительно максимума, см. также [47]. В частности, эта конструкция улучшает верхнюю оценку на размер наименьшего множества перестановок, независимых по четыре относительно максимума, приведенную в [25].

Более сложные алгоритмы, основанные на лемме о кодировании выполняющего набора. Как показано выше, алгоритм 3.1 имеет допустимую вероятность ошибки на формулах, имеющих не более одного выполняющего набора. На самом деле, это утверждение верно и для произвольных формул в 3-КНФ. Более того, очевидно, что лемма о кодировании выполняющего набора верна и для k-КНФ при $k > 3$ (в этом случае длина описания составит $(1 - 1/k)n$), так что алгоритм, аналогичный алгоритму 3.1, может быть использован для решения k-ВЫП за время $\mathrm{poly}(|F|) \cdot 2^{(1-1/k)n}$.

Однако дерандомизация алгоритма 3.1 для формул с (возможно) более чем одним выполняющим набором оказывается более сложной. Приведем алгоритм из [38]:

Алгоритм 3.2.

1. Выбрать γ, такое, что
$$1 - \gamma/k \approx -\gamma \log_2 \gamma - (1-\gamma)\log_2(1-\gamma).$$
2. Для каждого набора A, присваивающего значения всем n переменным и содержащиего не более γn положительных литералов, проверить, является ли A выполняющим для F, и, если так, выдать его и остановиться.
3. Для каждой перестановки $\pi \in B_n$, построить дерево расщепления глубины не более $n(1-\gamma/k)+1$, используя процедуру 2.1 с единственным правилом преобразования (2.1). При каждом расщеплении выбирать переменную x_i, такую что x_i входит в формулу, и $\pi(i)$ – наименьшее из возможных. Если выполняющий набор найден при помощи процедуры 2.1, выдать его и остановиться.
4. Выдать ответ “Невыполнима”. •

Оценка времени работы алгоритма 3.2 хуже, чем для алгоритма 3.1; для формул в k-КНФ его время работы составляет $\mathrm{poly}(|F|) \cdot 2^{n(1-\gamma/k)}$, что является $\mathrm{poly}(|F|) \cdot 2^{0.896n}$ для 3-КНФ.

Лемма о кодировании выполняющего набора учитывает только переменные, которые исключаются из описания выполняющего набора из-за применения правила (2.1) к переменной x в следующем случае

- имеется x-критический клоз, содержащий x, y и z,
- $\pi(x) > \pi(y)$, и
- $\pi(x) > \pi(z)$.

Однако, условие $\pi(x) > \pi(z)$ не является необходимым для того, чтобы переменная z исчезла из этого клоза. Например, z в свою очередь может быть исключена из-за применения правила (2.1). Замечательный способ использовать и учитывать такие зависимости описан в [37]:

Алгоритм 3.3.

1. Добавить в F все клозы, которые могут выведены из нее при помощи резолюции, примененной к клозам размера не более $r(n)$.
2. Действовать как в алгоритме 3.1, но для меньшей глубины $d(n)$ дерева. •

Анализ этого алгоритма и его дерандомизованного варианта довольно сложен; в [37] даны оценки $\mathrm{poly}(|F|) \cdot 2^{0.448n}$ для 3-ВЫП и $\mathrm{poly}(|F|) \cdot 2^{0.387n}$ для одновыполнимых формул в 3-КНФ.

Более общий метод, основанный на алгоритме 3.1, можно описать следующим образом:

Алгоритм 3.4.

1. Построить дерево расщепления *определенной глубины,* используя процедуру 2.1 с *определенными правилами преобразования*. Выбирать переменную для каждого расщепления случайным образом равномерно среди переменных, все еще входящих в формулу. Если выполняющий набор найден при помощи процедуры 2.1, выдать его и остановиться; в противном случае выдать ответ “Невыполнима”. •

Открытый вопрос. В этом разделе и работе [47] строятся множества B_n для $k = 3$ и $k = 4$, соответственно. Эти конструкции, представляющие самостоятельный интерес, основаны на свойствах групп $PGL(k-1, p^s)$ для $k = 3, 4$. Можно ли обобщить это конструкции на случай произвольного k? В [47] высказана гипотеза, что группа $PGL(k-1, p^s)$, действующая на точках $(k-2)$-мерного проективного пространства, обладает свойством независимости относительно минимума (максимума) для некоторого линейного порядка на точках проективного пространства.

4. Еще два доказательства Леммы Вэлианта-Вазирани

Лемма Вэлианта-Вазирани [46] является одним из первых нетривиальных результатов об отношениях между сложностными классами.

Лемма Вэлианта-Вазирани. *Имеется вероятностное (с односторонней ошибкой) полиномиальное по времени сведéние задачи выполнимости к ее примерам, имеющим не более одного выполняющего набора.*

Иными словами, по заданной формуле F в КНФ можно построить формулы $F_1, \ldots, F_m$ в КНФ, такие что

- если F выполнима, то с вероятностью, большей 1/2, по крайней мере одна из формул $F_1, \ldots, F_m$ имеет в точности один выполняющий набор;
- если F невыполнима, то все формулы $F_1, \ldots, F_m$ невыполнимы.

Обобщение леммы Вэлианта-Вазирани использовано, например, в известной теореме Тода [45], которая утверждает, что любой язык полиномиальной иерархии сводится по Тьюрингу (детерминированно за полиномиальное время) к языку из класса $\mathcal{PP}$. В нашем контексте подобное утверждение также было бы полезно, поскольку современные верхние оценки для одновыполнимых формул в 3-КНФ [37] (см. также раздел 3) лучше оценок для произвольных формул в 3-КНФ [43] (см. также раздел 5).

Известно много различных доказательств этой леммы [5, 6, 7, 34]. Также имеется несколько доказательств близкого результата о существовании вероятностного сведéния ВЫП к языку из класса $\oplus\mathcal{P}$ [21, 36]. В этом разделе мы приводим еще два новых доказательства. Оба наших доказательства основаны на идее, использованной в [5]. В [5] эта идея комбинируется с использованием сложности по Колмогорову. В наших доказательствах мы избавляемся от применения сложности по Колмогорову. Это упрощает доказательство из [5] и выявляет его теоретико-числовую суть.

Замечание. Как было отмечено выше, утверждение, подобное лемме Вэлианта-Вазирани, было бы полезно в контексте нашей статьи. Однако, для этой цели необходимо сведéние, удовлетворяющее дополнительному требованию: исходная формула в 3-КНФ сводится к формулам в 3-КНФ с таким же (или почти таким) количеством переменных. Первоначальное сведéние, предложенное Вэлиантом и Вазирани, добавляет к исходной формуле $\Omega(n)$ равенств вида $l_1 \oplus l_2 \oplus \cdots \oplus l_n = 0$, где $l_i = x_i$ или $l_i = \neg x_i$ (выбор между переменной или ее отрицанием производится случайным образом), а $\oplus$ означает сложение по модулю 2. Для того, чтобы представить все эти равенства в 3-КНФ, требуется ввести $\Omega(n^2)$ новых переменных. Более того, ни одно из известных сведéний (включая наши) не удовлетворяет вышеприведенному требованию.

Первое сведéние. Пусть F – формула, а набор A присваивает значения всем ее переменным. Отождествим A с n-битным числом $a = a_0a_1 \ldots a_{n-1}$, таким что $a_j = 1$, если соответствующая переменная в наборе A имеет значение `true`, и $a_j = 0$ в противном случае. Выберем целые числа p_i и r_i следующим образом. Сначала выберем равновероятно случайным образом целое $i \in [0..n]$. Затем выберем равновероятно случайным образом $p_i \in [1..b_i]$ и $r_i \in [0..b_i]$, где $b_i = 4 \cdot 2^i n^2$. Заменим F на формулу

$$F \wedge (a \bmod p_i = r_i).$$

Выражение “$(a \bmod p_i = r_i)$” обозначает здесь булеву формулу в КНФ от переменных $a_0, \ldots, a_{n-1}$ (возможно, также использующую дополнительные переменные), которая представляет соответствующее арифметическое сравнение. Например, эта формула может быть получена посредством кодирования обычного умножения “в столбик”. Очевидно, что это сведéние полиномиально по времени и переводит невыполнимую формулу в невыполнимую формулу. Остается доказать, что если F выполнима, то с большой вероятностью новая формула одновыполнима.

Пусть $a^{(1)}, \ldots, a^{(D)}$ – все выполняющие наборы формулы F. Заметим, что $i = \lceil \log_2 D \rceil$ с вероятностью $1/(n+1)$. Предположим, что это событие имеет место. Заметим, что для данных j, h $(j \neq h)$ имеется не более n простых делителей разности $a^{(j)} - a^{(h)}$.

С другой стороны, для достаточно больших n имеется, как минимум, $0.92129 \cdot b_i/\ln b_i > b_i/\log_2 b_i \geqslant 2^{i+1}n$ простых чисел, не превосходящих b_i [8]. Таким образом, имеется, по крайней мере, $2^{i+1}n - 2^i n = 2^i n$ чисел p, не превосходящих b_i, таких, что остаток выполняющего набора $a^{(j)}$ по модулю p отличается от остатков всех других выполняющих наборов по модулю p. Следовательно, по крайней мере $2^i n$ пар $0 \le p_i, r_i \le b_i$ "отличают" набор $a^{(j)}$ от всех остальных. Заметим, что для различных выполняющих наборов множества "отличающих" пар дизъюнктны. Следовательно, имеется не менее $2^i n \cdot D \geqslant 2^{2i-1}n$ искомых пар. Таким образом, для достаточно больших n вероятность выбрать такую пару составляет, по крайней мере, $\frac{2^{2i-1}n}{16 \cdot 2^{2i}n^4} = \frac{1}{32n^3}$.

Домножая на вероятность выбрать "правильное" i, получаем, что вероятность ошибки в нашем свед́ении не превосходит $1 - \frac{1}{32n^4+32}$. Выбирая тройки (i, p_i, r_i) случайным образом $O(n^4)$ раз, получаем константную вероятность ошибки.

Второе свед́ение. Пусть $\mathbb{Z}_p[t]$ обозначает кольцо многочленов от одной переменной над конечным полем из p элементов. Хорошо известно, что с точки зрения арифметики кольцо $\mathbb{Z}_p[t]$ является "правильным" аналогом кольца $\mathbb{Z}$. С другой стороны, в отличие от асимптотических формул для простых чисел в интервале, для количества неприводимых многочленов данной степени существует явная точная формула, имеющая элементарное доказательство. Кроме того, булева формула $(a \bmod p_i = r_i)$, где a, p_i, r_i – многочлены над $\mathbb{Z}_2$, имеет более простой вид, поскольку здесь не возникают переносы из разряда в разряд. Это может оказаться полезным в свете открытого вопроса 2 (см. ниже). Отметим также, что подход, подобный приведенному ниже, может быть реализован над произвольным конечным полем, а не только над $\mathbb{Z}_2$.

В первом свед́ении мы отождествили набор A с n-битным числом $a = a_0, \ldots, a_{n-1}$. Теперь мы отождествим A с коэффициентами многочлена $a(t) = a_0 + a_1 t + \cdots + a_{n-1}t^{n-1}$ над полем $\mathbb{Z}_2$, где a_j определяются, как при первом свед́ении. Вновь выберем $i \in [0..n]$ случайным образом и заменим входную формулу F на формулу

$$F \wedge (a \bmod p_i = r_i),$$

где теперь p_i, r_i – многочлены, выбранные случайным образом, такие что $\deg p_i = d_i$ (т.е. $p_i = x_i^{d_i} + \sum_{t=0}^{d_i-1} c_t x_i^t$, где c_t выбраны случайным образом), $\deg r_i < d_i$, а $d_i = i + \lceil \log_2 n \rceil + 4$.

Предположим, что $i = \lceil \log_2 D \rceil$. Возьмем $d = d_i$. Обозначим через N_d количество неприводимых над $\mathbb{Z}_2$ многочленов степени d. Для N_d имеется явная формула (см. например, [24, гл. 7, §2, следствие 2]). А именно,

$$N_d = \frac{\sum_{s|d} \mu(\frac{d}{s}) 2^s}{d},$$

где μ обозначает функцию Мебиуса:

- $\mu(1) = 1$;
- $\mu(h) = 0$, если h не свободно от квадратов (т.е. h делится на квадрат простого);
- $\mu(h) = (-1)^u$, если $h = q_1 \dots q_u$, где $q_1, \dots, q_u$ – попарно различные простые.

Поскольку для всех h выполняется $\mu(h) \geqslant -1$ и $\mu(1) = 1$, имеем

$$N_d \geqslant \frac{2^d - \sum_{s \leqslant d/2} 2^s}{d} \geqslant \frac{2^d - 2^{d/2+1}}{d} \geqslant \frac{2^{d-1}}{d}.$$

С другой стороны, каждая из $D - 1$ разностей $a^{(j)} - a^{(h)}$ (где $j \neq h$) имеет не более чем n/d неприводимых делителей степени d. Таким образом, по крайней мере $N_d - \frac{nD}{d} \geqslant \frac{2^{d-1}}{d} - \frac{2^{d-4}}{d} \geqslant \frac{2^{d-2}}{d}$ пар (p_i, r_i) (где $\deg p_i = d$, $\deg r_i < d$) “отличают” набор $a^{(j)}$ от всех остальных выполняющих наборов.

Аналогично первому сведе́нию, имеется не менее $\frac{2^{d-2}}{d} \cdot D$ искомых пар. Вероятность выбрать такую пару составляет не менее

$$\frac{2^{d-2} D}{d 2^{2d-1}} \geqslant \frac{1}{64nd} \geqslant \frac{1}{128n^2}$$

при $n \geqslant 8$. Домножая на вероятность выбрать “правильное” i, имеем, что вероятность ошибки нашего сведе́ния не превосходит $1 - \frac{1}{128n^3+128}$, т.е., повторяя случайный выбор $O(n^3)$ раз, можно получить константную вероятность ошибки.

Открытые вопросы.

1. Вопрос о дерандомизации леммы Вэлианта-Вазирани остается открытым. Например, можно ли заменить вероятностное полиномиальное по времени сведе́ние на детерминированное сведе́ние, осуществляемое за время $\mathrm{poly}(|F|) \cdot c^n$ для

некоторой константы $c < 2$? Аналогичный вопрос для теоретически более слабого сведéния выполнимости к ее примерам, имеющим нечетное число либо нуль выполняющих наборов также остается открытым.

2. Все известные доказательства леммы Вэлианта-Вазирани сводят выполнимость к формулам в КНФ с клозами произвольной длины, даже если исходная формула была в 3-КНФ. Последующее сведéние к 3-КНФ может значительно увеличить число переменных в формуле. Имеется ли естественное сведéние к формулам в 3-КНФ, такое, что рост количества переменных не слишком велик, например, появляется лишь $o(n)$ новых переменных?

5. Алгоритмы, основанные на локальном поиске

Алгоритмы для ВЫП, основанные на локального поиске, включают в себя жадный поиск [27, 44], "осторожный" поиск [18], случайное блуждание [39, 43], другие стратегии и их комбинации (см. обзор в [20]). Хотя многие из этих алгоритмов хорошо изучены экспериментально, хорошие верхние оценки доказаны только для *алгоритмов, основанных на случайных блужданиях*, наиболее простых из них. В этом разделе мы обсуждаем алгоритмы для ВЫП, основанные на случайных блужданиях, их дерандомизацию и возможность комбинирования в одном алгоритме подхода, использующего случайные блуждания, и подхода, использующего расщепление (рассматриваемое как бэктрекинг).

Алгоритмы, основанные на случайных блужданиях. Алгоритмы для ВЫП, основанные на случайных блужданиях, – это очень простые вероятностные алгоритмы, которые выбирают начальный набор случайным образом и шаг за шагом двигаются к выполняющему набору: на каждом шаге алгоритм изменяет значение одной переменной, выбранной случайным образом из невыполненного клоза, на противоположное. Такие алгоритмы решают 2-ВЫП за полиномиальное время [39] и решают k-ВЫП за время $(2 - 2/k)^n$ с точностью до полиномиального сомножителя, где n – число переменных во входной формуле [43]. В частности, для $k = 3$ эта оценка равна $(4/3)^n$, что является наилучшей из известных на данный момент верхних оценок для алгоритмов, решающих 3-ВЫП.

Следующий алгоритм (параметризованный функциями α и τ) описывает семейство алгоритмов для ВЫП, основанных на случайных блужданиях. Для данной входной формулы F, имеющей n переменных, алгоритм производит не более $\alpha(n)$ блужданий, выходящих из случайного начального набора; при этом каждое блуждание состоит из не более чем $\tau(n)$ шагов.

Алгоритм 5.1.

1. Повторить $\alpha(n)$ раз:
1a. Выбрать случайным образом набор A (в соответствии с равномерным распределением).
1b. Если A выполняет F, выдать ответ A и остановиться. В противном случае повторить следующие инструкции $\tau(n)$ раз:
   - Взять произвольный невыполненный клоз C в F.
   - Выбрать случайным образом (в соответствии с равномерным распределением) переменную x среди переменных, входящих в C.
   - Модифицировать набор A, изменив значение переменной x в A на противоположное.
   - Если модифицированный набор A выполняет F, выдать ответ A и остановиться.

2. Выдать ответ “Невыполнима” и остановиться. •

Алгоритм 5.1 при $\alpha(n) = 1$ и $\tau(n) = 2n^2$ представляет собой полиномиальный алгоритм Пападимитриу для 2-ВЫП [39]. Алгоритм 5.1 при $\alpha(n) = (2 - 2/k)^n$ и $\tau(n) = 3n$ представляет собой алгоритм Шонинга сложности $(2 - 2/k)^n$ для k-ВЫП [43].

Для анализа алгоритма 5.1 мы используем его связь с *одномерными случайными блужданиями* (см., напр., [16]). Рассмотрим работу этого алгоритма на формуле в k-КНФ. Предположим, что входная формула имеет выполняющий набор S, который отличается от начального набора A значениями ровно i переменных. Заметим, что на каждом шаге алгоритм приближается к S с вероятностью по крайней мере $1/k$, поскольку невыполненный клоз всегда содержит хотя бы одну переменную, значения которой в S и в текущем наборе различны. Таким образом, процесс модификации, начинающийся с A, связан со следующим одномерным случайным блужданием.

Рассмотрим частицу, блуждающую на интервале $[0..n]$. Частица стартует с позиции i в момент времени $t = 0$ и совершает $\tau(n)$

шагов. На каждом шаге, если частица находится в позиции j при $0 < j < n$, она переходит в $j-1$ с вероятностью $1/k$ и в $j+1$ с вероятностью $1-1/k$. Если $j=0$, частица остается в той же позиции с вероятностью 1. Если $j=n$, частица переходит в $j-1$ с вероятностью 1. Обозначим через $p_{i,t}$ вероятность того, что частица, вышедшая из i, достигнет 0 за t шагов. Следующая лемма устанавливает связь между $p_{i,t}$ и вероятностью ошибки алгоритма 5.1.

Лемма 5.1. *Вероятность ошибки алгоритма 5.1 не превосходит*

$$\exp\left(-\frac{\alpha(n)}{2^n}\sum_{i=0}^{n}\binom{n}{i}p_{i,\tau(n)}\right).$$

Доказательство. Достаточно рассмотреть случай, когда входная формула выполнима. Пусть S – произвольный выполняющий набор. Рассмотрим одно из $\alpha(n)$ блужданий, совершенных алгоритмом. Для каждого i начальный набор A отличается от S значениями ровно i переменных с вероятностью $\binom{n}{i}/2^n$. Легко видеть, что такое блуждание находит S (или другой выполняющий набор, если некоторый предшествующий в ходе блуждания набор оказался выполняющим) с вероятностью по крайней мере $p_{i,\tau(n)}$. Суммируя по всем возможным значениям i, получаем нижнюю оценку

$$p = \sum_{i=0}^{n}\frac{\binom{n}{i}}{2^n}p_{i,\tau(n)}$$

на вероятность того, что блуждание находит выполняющий набор за $\tau(n)$ шагов. Следовательно, вероятность ошибки алгоритма 5.1 не превосходит

$$\begin{aligned}(1-p)^{\alpha(n)} &= \exp\left(\alpha(n)\ln(1-p)\right) \le \\ &\le \exp(-\alpha(n)\cdot p) = \exp\left(-\alpha(n)\sum_{i=0}^{n}\frac{\binom{n}{i}}{2^n}p_{i,\tau(n)}\right). \qquad \bullet\end{aligned}$$

Алгоритм Пападимитриу. Пападимитриу [39] доказал, что алгоритм 5.1 при $\alpha(n)=1$ и $\tau(n)=2n^2$ (т.е. работающий полиномиальное время) решает 2-ВЫП с допустимой вероятностью ошибки. Доказательство основано на следующем наблюдении: если мы меняем значение переменной, выбранной случайным образом из невыполненного 2-клоза, на противоположное, то мы приближаемся к выполняющему набору с вероятностью, по крайней мере, 1/2. Вероятность $p_{i,2n^2}$ для соответствующего одномерного случайного блуждания составляет, по крайней мере, 1/2 [39, 40]; следовательно, вероятность ошибки этого алгоритма является допустимой.

Мы покажем, что работа алгоритма Пападимитриу на формулах, изоморфных хорновским и не содерщащих единичных клозов, может быть описана с помощью того же одномерного случайного блуждания. Таким образом, алгоритм (расширенный правилом удаления единичных клозов) может быть использован для вычисления выполняющего набора формулы, изоморфной хорновской, за полиномиальное время с допустимой вероятностью ошибки.

Напомним, что формула F называется *хорновской* формулой, если каждый клоз в F содержит не более одного положительного литерала. Пусть l – литерал, входящий в формулу F. *Обращением* литерала l в формуле F мы называем замену всех вхождений l и $\neg l$ на $\neg l$ и l соответственно. Формула, *изоморфная хорновской*, определяется как формула, которая может быть преобразована в хорновскую формулу обращением некоторых литералов [32]. Имеется ряд полиномиальных алгоритмов, решающих ВЫП для формул, изоморфных хорновским, например, SLUR (Single Lookahead Unit Resolution) [17, 42].

Заметим, что если формула F, изоморфная хорновской, не содержит единичных клозов, то F выполнима. Это наблюдение приводит к следующему алгоритму. Сначала мы удаляем все единичные клозы (если они существуют) из входной формулы с помощью правила преобразования (2.1). Легко видеть, что удаление единичных клозов преобразует формулу, изоморфную хорновской, в формулу, изоморфную хорновской. Затем мы применяем алгоритм Пападимитриу для нахождения выполняющего набора.

Теорема 5.2. *Пусть F – формула, изоморфная хорновской и не содержащая единичных клозов. Алгоритм 5.1 при $\alpha(n) = 1$ и $\tau(n) = 2n^2$ находит выполняющий набор для F за полиномиальное время с допустимой вероятностью ошибки.*

Доказательство. Достаточно доказать, что работа алгоритма 5.1 на формуле F имеет те же свойства, что и его работа на формулах в 2-КНФ: при изменении значения переменной, выбранной случайным образом из невыполненного клоза, на противоположное, мы приближаемся к выполняющему набору с вероятностью по крайней мере 1/2.

Ясно, что если F – хорновская формула без единичных клозов, то F выполняется набором S_0, в котором все переменные имеют значение `false`. Далее, для каждого клоза C в F выполнено следующее условие: все литералы из C за исключением, быть может, одного истинны при S_0. Легко видеть, что случай формул, изоморфных хорновским, аналогичен случаю хорновских формул. А именно, если F – формула, изоморфная хорновской и не содержащая единичных клозов, то F имеет выполняющий набор S, такой что для любого клоза C все литералы C за исключением, быть может, одного истинны при S.

Рассмотрим выбор переменной для изменения значения из невыполненного клоза C. Пусть k – число литералов в C. Все эти литералы ложны при текущем наборе. В то же время все они за исключением, быть может, одного должны быть истинны при S. Следовательно, при изменении на противоположное значения переменной одного из этих k литералов, мы приближаемся к S с вероятностью по крайней мере $\frac{k-1}{k}$. Поскольку формула F не содержит единичных клозов, эта вероятность не меньше 1/2. •

Алгоритм Шонинга. Шонинг [43] доказывает, что k-ВЫП может быть решена алгоритмом 5.1 при $\alpha(n) = (2-2/k)^n$ и $\tau(n) = 3n$ с допустимой вероятностью ошибки. Доказательство основано на следующей оценке вероятности $p_{i,3n}$: показано, что $p_{i,3n} \geq (k-1)^{-i}$ с точностью до полиномиального сомножителя для любого $k \geqslant 3$ и любого i. Тогда лемма 5.1 дает верхнюю оценку

$$\exp\left(-\frac{(2-2/k)^n}{2^n}\sum_{i=0}^{n}\binom{n}{i}\frac{(k-1)^{-i}}{\mathrm{poly}(n)}\right) =$$
$$= \exp\left(\frac{(1-1/k)^n}{\mathrm{poly}(n)}\left(1+\frac{1}{k-1}\right)^n\right) = \exp\left(-\frac{1}{\mathrm{poly}(n)}\right) = 1-\frac{1}{\mathrm{poly}(n)}$$

вероятности ошибки алгоритма Шонинга. Последнее равенство[5] следует из неравенства $\exp(-x) \leqslant 1 - x/e$, выполненного при любом $x < 1$.

[5] См. наше соглашение об использовании обозначения poly в разделе 1.

Дерандомизация алгоритма Шонинга. Детерминированная версия алгоритма Шонинга описана в [11, 12]. Идея, стоящая за этой дерандомизацией, проста и интуитивно понятна. Сначала мы покрываем все пространство поиска (состоящее из всех возможных 2^n наборов) шарами некоторого фиксированного радиуса R в метрике Хэмминга. Это покрытие должно быть минимальным, т.е. мы стараемся использовать как можно меньше шаров. Затем мы рассматриваем каждый шар и ищем выполняющий набор внутри него. Ясно, что существует компромисс между числом шаров и временем поиска внутри шара: чем больше шаров мы используем, тем быстрее происходит поиск внутри шара. Таким образом, число шаров (или, эквивалентно, радиус R) имеет "оптимальное" значение, которое минимизирует верхнюю оценку общего времени работы алгоритма. Как показано в [11, 12], если входная формула находится в k-КНФ, "оптимальное" значение R равно $\frac{n}{k+1}$. Общее время работы полученного алгоритма составляет $(2-\frac{2}{k+1}+\epsilon)^n$ с точностью до полиномиального сомножителя, что является наилучшей из известных верхних оценок для детерминированного алгоритма для k-ВЫП.

Для нахождения выполняющего набора внутри шара алгоритм [11, 12] использует очень простую процедуру, основанную на локальном поиске с бэктрекингом. А именно, возьмем невыполненный клоз $l_1 \vee \ldots \vee l_t$ и рассмотрим каждую из t возможностей: неверно значение l_1, ..., неверно значение l_t. Глубина дерева, построенного таким образом, может быть ограничена числом R, и, следовательно, число вершин составляет $O(k^R)$. Используя более эффективную (но более сложную) версию этой процедуры, можно улучшить оценку размера этого дерева и, следовательно, общего времени работы алгоритма. Для 3-ВЫП улучшенный алгоритм имеет время работы $\mathrm{poly}(|F|) \cdot 1.481^n$ [11, 12].

Возможно также избавиться от слагаемого "$+\epsilon$" в оценке времени работы этого алгоритма за счет использования экспоненциальной памяти [11].

Алгоритмы, основанные на случайных блужданиях с "кнопкой возврата". Возможно ли включить *бэктрекинг* в алгоритмы для ВЫП, основанные на случайных блужданиях? Это можно сделать с использованием предложенного недавно подхода Фэйджина и др. [15], которые ввели *цепи Маркова с "кнопкой возврата"* (изучение таких процессов мотивировано их связью с блужданием по Интернету). А именно, разрешается, чтобы каждый шаг с некоторой вероятностью являлся *шагом назад*. Такой шаг, аналогично нажатию на кнопку "Назад", отменяет последнее изменение в текущем наборе. Таким образом, шаги назад могут рассматриваться как вариант бэктрекинга для алгоритмов, основанных на случайных блужданиях.

Целью данного раздела является выяснение того, может ли наличие "кнопки возврата" изменить вероятность нахождения выполняющего набора. Чтобы ответить на этот вопрос, мы сравниваем два типа случайных блужданий на прямой: блуждания с шагами назад и без них. Используя результаты [15], мы приводим сравнительные оценки вероятностей успеха блужданий полиномиальной длины. А именно, мы показываем, что шаги назад не могут увеличить вероятность успеха более, чем на полиномиальный сомножитель, т.е. "кнопка возврата" не дает существенной выгоды. Мы также показываем, что если вероятность шага назад не превосходит 0.5, то наличие "кнопки возврата" не приводит к существенному проигрышу.

Мы преобразуем алгоритм 5.1 (обозначаемый RW – Random Walks) в алгоритм 5.2 (обозначаемый RWB – Random Walks with Back button). Новый алгоритм RWB хранит *стек истории* H, элементами которого являются наборы. Верхний элемент стека соответствует текущему набору. Как и RW, алгоритм RWB совершает не более $\alpha(n)$ блужданий; каждое блуждание состоит из не более чем $\tau(n)$ шагов, но каждый шаг является либо *шагом вперед*, либо *шагом назад*. При шаге вперед RWB модифицирует текущий набор так же, как и RW; если модифицированный набор A выполняет F, то RWB выдает ответ A и останавливается; в противном случае RWB помещает новый набор A в стек H. Шаг назад состоит в том, что RWB извлекает верхний элемент из стека H (таким образом, новым верхним элементом становится текущий набор). Начиная с пустого стека истории, RWB работает следующим образом:

Алгоритм 5.2.

1. Повторить $\alpha(n)$ раз:

1a. Выбрать случайным образом равномерно распределенный набор A.

1b. Если A выполняет F, выдать ответ A и остановиться. В противном случае поместить A в стек H и повторить следующую инструкцию $\tau(n)$ раз:

- Если H содержит только один элемент, сделать шаг вперед. В противном случае сделать шаг назад с вероятностью b и шаг вперед с вероятностью $1-b$.

2. Выдать ответ “Невыполнима” и остановиться. •

Вероятность b называется *вероятностью возврата*.

Чтобы представить работу RWB с помощью одномерных случайных блужданий, мы модифицируем модель одномерного случайного блуждания для RW (описанную после алгоритма 5.1, см. выше) следующим образом. Блуждание для RWB хранит стек истории H, элементами которого являются позиции частицы. Каждый шаг блуждания является либо шагом вперед, либо шагом назад. При шаге вперед частица двигается в соответствии с правилами для RW, и ее новая позиция помещается в стек H. При шаге назад из H извлекается верхний элемент, и частица переходит в позицию, которая после этой операции оказывается верхним элементом стека. В момент времени $t=0$ стек истории H содержит только начальную позицию i. Затем частица двигается аналогично RWB: если H содержит только один элемент, частица совершает шаг вперед; в противном случае она совершает шаг назад с вероятностью b и шаг вперед с вероятностью $1-b$. Обозначим через $p_{i,t}^{(b)}$ вероятность того, что частица достигнет 0 не более, чем за t шагов.

Лемма 5.3. *Вероятность ошибки алгоритма 5.2 не превосходит*

$$\exp\left(-\frac{\alpha(n)}{2^n}\sum_{i=0}^{n}\binom{n}{i}p_{i,\tau(n)}^{(b)}\right).$$

Доказательство. Аналогично доказательству леммы 5.1. •

Лемма 5.1 и лемма 5.3 показывают, что из нижних оценок на $p_{i,t}$ и $p_{i,t}^{(b)}$ следуют верхние оценки вероятностей ошибок алгоритмов RW и RWB соответственно. Таким образом, сравнительные оценки на $p_{i,t}$ и $p_{i,t}^{(b)}$ позволяют сравнить верхние оценки для RW и RWB (полученные с использованием лемм 5.1 и 5.3).

Теперь мы докажем две теоремы. Первая теорема показывает, что наличие “кнопки возврата” не может увеличить вероятность достижения 0 более чем на полиномиальный сомножитель. Вторая теорема показывает, что наличие “кнопки возврата” (при $b \le 0.5$) не может снизить эту вероятность более чем на полиномиальный сомножитель, если допустить квадратичное увеличение длины блуждания. Обе теоремы доказываются с использованием результатов [15].

Пусть $X_{i,t}$ и $X^{(b)}_{i,t}$ обозначают позиции частицы в момент времени t для случайных блужданий, выходящих из i, для RW и RWB соответственно. Пусть H_t обозначает стек истории случайного блуждания для RWB в момент времени t; следуя [15], определим длину $l(H_t)$ стека истории как число позиций частиц, хранимых в нем, *минус* 1 (т.е. мы не учитываем начальную позицию).

Теорема 5.4. *Для любой вероятности возврата b,*

$$p^{(b)}_{i,t} \;\le\; (t+1)^2 \cdot p_{i,t}.$$

Доказательство. Заметим, что значение $p^{(b)}_{i,t}$ ограничено сверху суммой $t+1$ вероятностей $Pr\{X^{(b)}_{i,t'} = 0\}$ при $t' = 0, 1, \ldots, t$. Пусть $t' \in [0..t]$ – значение, при котором достигается максимум $Pr\{X^{(b)}_{i,t'} = 0\}$. Поскольку $0 \le l(H_s) \le s$ при любом s, мы получаем

$$\begin{aligned} p^{(b)}_{i,t} &\leqslant (t+1) Pr\{X^{(b)}_{i,t'} = 0\} = \\ &= (t+1) \sum_{\lambda=0}^{t'} Pr\{X^{(b)}_{i,t'} = 0 \mid l(H_{t'}) = \lambda\} \cdot Pr\{l(H_{t'}) = \lambda\} \le \\ &\le (t+1)(t'+1) \cdot \max_{0 \le \lambda \le t'} Pr\{X^{(b)}_{i,t'} = 0 \mid l(H_{t'}) = \lambda\}. \end{aligned}$$

По теореме 3.1 из [15],

$$Pr\{X^{(b)}_{i,t'} = 0 \mid l(H_{t'}) = \lambda\} = p_{i,\lambda}.$$

Поскольку $p_{i,\lambda} \le p_{i,t}$ при $0 \le \lambda \le t' \le t$, теорема доказана. •

Теорема 5.5. *Для любой вероятности возврата $b \le 0.5$,*

$$p^{(b)}_{i,t^2} \;\ge\; p_{i,t} \,/\, poly(t).$$

Доказательство. Вновь используя теорему 3.1 из [15], получаем

$$p^{(b)}_{i,t^2} \geqslant Pr\{X^{(b)}_{i,t^2} = 0\} = \sum_{\lambda=0}^{t^2} Pr\{X^{(b)}_{i,t^2} = 0 \mid l(H_{t^2}) = \lambda\} \cdot Pr\{l(H_{t^2}) = \lambda\}$$

$$= \sum_{\lambda=0}^{t^2} p_{i,\lambda} \cdot Pr\{l(H_{t^2}) = \lambda\}.$$

Чтобы оценить $Pr\{l(H_{t^2}) = \lambda\}$ при $b \leq 0.5$, оценим сначала $Pr\{l(H_{t^2}) = t\}$ при $b = 0.5$. Рассмотрим следующее случайное блуждание на прямой: частица выходит из 0 и двигается налево и направо с вероятностями, равными 0.5. Легко видеть, что вероятность $Pr\{l(H_s) = \mu\}$ при $b = 0.5$ в точности равна вероятности того, что позиция частицы в момент времени s есть либо μ, либо $-\mu$. В частности, $Pr\{l(H_{t^2}) = t\} = 2r_t$, где r_t обозначает вероятность того, что положение частицы в момент времени t^2 есть t. Вероятность r_t составляет в точности $\binom{t^2}{(t^2-t)/2} \cdot 2^{-t^2}$, что отличается на полиномиальный сомножитель от

$$\frac{t^{2t^2}}{2^{t^2} \cdot (\frac{t^2-t}{2})^{(t^2-t)/2} \cdot (\frac{t^2+t}{2})^{(t^2+t)/2}}$$

(по формуле Стирлинга). Непосредственные вычисления показывают, что при достаточно большом t последнее выражение больше константы. Таким образом, мы доказали утверждение для $b = 0.5$.

Если $b \leqslant 0.5$, вероятность $Pr\{l(H_{t^2}) \geqslant t\}$ даже больше, чем в случае $b = 0.5$ (и, очевидно, больше, чем оцененная выше вероятность $Pr\{l(H_{t^2}) = t\}$). В частности, эта вероятность больше $1/\mathrm{poly}(t)$. Напомним, что $Pr\{l(H_{t^2}) = \lambda\} = 0$ при $\lambda > t^2$. Следовательно, существует такое λ, что $t \leq \lambda \leq t^2$ и $Pr\{l(H_{t^2}) = \lambda\} \geqslant Pr\{l(H_{t^2}) \geqslant t\}/t^2 \geqslant 1/\mathrm{poly}(t)$. Поскольку при $\lambda \geq t$ имеет место $p_{i,\lambda} \geq p_{i,t}$, теорема доказана. •

Открытые вопросы.

1. Заметим, что в то время как существование критических клозов улучшает работу PPSZ-подобных алгоритмов, оно является узким местом для алгоритмов случайного блуждания: если невыполненный клоз, который мы выбираем, не является критическим, то вероятность продвижения в "правильном" направлении составляет, по крайней мере, $2/k$, а не $1/k$. Было бы интересно использовать взаимодополняемость этих двух подходов, комбинируя их в одном алгоритме.
2. Мы показали, что RWB аналогичен RW, если блуждания имеют полиномиальную длину. Вопрос о том, имеет ли ме-

сто аналогичное утверждение для экспоненциально длинных блужданий, остается открытым.

Литература

1. S. Arora, C. Lund, *Hardness of approximation.* Approximation algorithms for NP-hard problems (D. Hochbaum, ed.), Boston, PWS Publishing Company, 1997.
2. R. Beigel, *Finding maximum independent sets in sparse and general graphs.* Proceedings of SODA'99. 1999, pp. 856–857.
3. R. Beigel, D. Eppstein, *3-coloring in time $O(1.3446^n)$: a no-MIS algorithm.* Proceedings of FOCS'95. 1995, pp. 444–452.
4. A. Broder, M. Charikar, A. Frieze, M. Mitzenmacher, *Min-wise independent permutations.* Proceedings of STOC'98. 1998, pp. 327–336.
5. H. Buhrman, L. Fortnow, *Resource-bounded Kolmogorov complexity revisited.* Proceedings of STACS'97. LNCS **1200**, Springer-Verlag, pp. 105–116.
6. S. Chari, *Randomness as a computational resource: issues in efficient computation.* PhD thesis, 1994.
7. S. Chari, P. Rohatgi, A. Srinivasan, *Improved algorithms via approximations of probability distributions.* — Journal of Computer and System Sciences **61** (2000), 81–107.
8. П. Л. Чебышев, *О простых числах.* — В кн.: П. Л. Чебышев, *Избранные труды*, М., Изд-во АН СССР, 1955, с. 33–54.
9. Е. Я. Данцин, *Две системы доказательства тавтологичности, основанные на методе расщеплений.* — Зап. науч. семин. ЛОМИ **105** (1981), 24–44; English translation: E. Ya. Dantsin, *Two propositional proof systems based on the splitting method*, Journal of Soviet Mathematics, 22(3):1293–1305, 1983.
10. E. Dantsin, M. Gavrilovich, E. A. Hirsch, B. Konev, *Approximation algorithms for MAX SAT: a better performance ratio at the cost of a longer running time.* — Ann. Pure App. Logic (2001) (to appear).
11. E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, U. Schöning, *A deterministic $(2-\frac{2}{k+1})^n$ algorithm for k-SAT based on local search.* — Theor. Comp. Sci. (2001), (to appear).
12. E. Dantsin, A. Goerdt, E. A. Hirsch, U. Schöning, *Deterministic algorithms for k-SAT based on covering codes and local search.* Proceedings of ICALP'2000 (U. Montanari, J. D. P. Rolim, E. Welzl, eds.), LNCS **1853**, Springer-Verlag, 2000.
13. M. Davis, G. Logemann, D. Loveland, *A machine program for theorem-proving.* — Communications of the ACM **5(7)** (1962), 394–397.
14. M. Davis, H. Putnam, *A computing procedure for quantification theory.* — Journal of the ACM **7(3)** (1960), 201–215.
15. R. Fagin, A. Karlin, J. Kleinberg, P. Raghavan, S. Rajagopalan, R. Rubinfeld, M. Sudan, A. Tomkins, *Random walks with "back buttons".* Proceedings of STOC'2000. 2000, pp. 484–493.
16. W. Feller, *An Introduction to Probability Theory and Its Applications.* vol. 1, Wiley, 1968; имеется перевод: У. Феллер, *Введение в теорию вероятностей и ее приложения*, т. 1, М., Мир, 1984.

17. J. Franco, A. Van Gelder, *A perspective on certain polynomial time solvable classes of satisfiability*. submitted to *Discrete Applied Mathematics*.
18. I. P. Gent, T. Walsh, *Towards an understanding of hill-climbing procedures for SAT*. Proceedings of AAAI-93. AAAI Press / MIT Press, 1993, pp. 28–33.
19. J. Gramm, E. A. Hirsch, R. Niedermeier, P. Rossmanith, *New worst-case upper bounds for MAX-2-SAT with application to MAX-CUT*. Electronic Colloquium on Computational Complexity. Technical Report **00-037**, 2000, Available from ftp://ftp.eccc.uni-trier.de/pub/eccc/reports/2000/TR00-037/index.html. To appear in *Discrete Applied Mathematics*..
20. J. Gu, P. Purdom, J. Franco, B. W. Wah, *Algorithms for the satisfiability problem*. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2000.
21. S. Gupta, *On bounded-probability operators and $C_=P$*. — Information Processing Letters **48** (1993), 93–98.
22. E. A. Hirsch, *New worst-case upper bounds for SAT*. — Journal of Automated Reasoning **24(4)** (2000), 397–420.
23. E. A. Hirsch, *Worst-case time bounds for MAX-k-SAT w.r.t. the number of variables using local search*. Proceedings of RANDOM 2000, ICALP Workshops 2000 (J. D. P. Rolim et al., eds.), Proceedings in Informatics **8**, Carleton Scientific, 2000, pp. 69–76.
24. K. Ireland, M. Rosen, *A Classical Introduction to Modern Number Theory*. Graduate Texts in Mathematics, **87**, Springer-Verlag, 1982; имеется перевод: К. Айерленд, М. Роузен, *Классическое введение в современную теорию чисел*, М., Мир, 1987.
25. T. Itoh, Y. Takei, J. Tarui, *On permutations with limited independence*. Proceedings of SODA 2000. 2000, pp. 137–146.
26. А. И. Кострикин, Ю. И. Манин, *Линейная алгебра и геометрия*. М., Изд-во МГУ, 1980; English translation: A. Kostrikin, Yu. Manin, *Linear Algebra and Geometry*, Gordon & Breach, 1989.
27. E. Koutsoupias, C. H. Papadimitriou, *On the greedy algorithm for satisfiability*. — Information Processing Letters **43(1)** (1992), 53–55.
28. O. Kullmann, *Heuristics for SAT algorithms: Searching for some foundations*. — Discrete Applied Mathematics (submitted) (1998).
29. O. Kullmann, *New methods for 3-SAT decision and worst-case analysis*. — Theoretical Computer Science **223 (1-2)** (1999), 1–72.
30. O. Kullmann, *Investigations on autark assignments*. — Discrete Applied Mathematics (2000) (to appear).
31. O. Kullmann and H. Luckhardt, *Deciding propositional tautologies: Algorithms and their complexity*. Preprint, http://www.cs.toronto.edu/~kullmann.
32. H. R. Lewis, *Renaming a set of clauses as a Horn set*. — Journal of the ACM **25(1)** (1978), 134–135.
33. B. Monien, E. Speckenmeyer, *Solving satisfiability in less then 2^n steps*. — Discrete Applied Mathematics **10** (1985), 287–295.
34. K. Mulmuley, U. V. Vazirani, V. V. Vazirani, *Matching is as easy as matrix inversion*. — Combinatorica **7(1)** (1987), 105–113.
35. R. Niedermeier, P. Rossmanith, *New upper bounds for MaxSat*. — Journal of Algorithms (2000) (to appear).

36. A. V. Naik, K. W. Regan, D. Sivakumar, *Quasilinear Time Complexity Theory.* — Theoretical Computer Science **148** (1995), 325-349.
37. R. Paturi, P. Pudlák, M. E. Saks, F. Zane, *An improved exponential-time algorithm for k-SAT.* Proceedings of FOCS'98. 1998, pp. 628–637.
38. R. Paturi, P. Pudlák, F. Zane, *Satisfiability coding lemma.* Proceedings of FOCS'97. 1997, pp. 566–574.
39. C. H. Papadimitriou, *On selecting a satisfying truth assignment.* Proceedings of FOCS'91. 1991, pp. 163–169.
40. C. H. Papadimitriou, *Computational Complexity.* Addison-Wesley, 1994.
41. J. M. Robson, *Algorithms for maximum independent set.* — Journal of Algorithms **7** (1986), 425–440.
42. J. S. Schlipf, F. S. Annexstein, J. V. Franco, R. P. Swaminathan, *On finding solutions for extended Horn formulas.* — Information Processing Letters **54 (3)** (1995), 133–137.
43. U. Schöning, *A probabilistic algorithm for k-SAT and constraint satisfaction problems.* Proceedings of FOCS'99. 1999, pp. 410–414.
44. B. Selman, H. Levesque, D. Mitchell, *A new method for solving hard satisfiability problems.* Proceedings of AAAI-92 (W. Swartout, ed.), MIT Press, 1992, pp. 440–446.
45. S. Toda, *PP is as hard as the polynomial-time hierarchy.* — SIAM Journal on Computing **20(5)** (1991), 865–877.
46. L. Valiant, V. Vazirani, *NP is as easy as detecting unique solutions.* — Theoretical Computer Science **47** (1986), 85–93.
47. M. Vsemirnov, *Automorphisms of projective spaces and min-wise independent families of permutations.* Manuscript in preparation. 2000.

Dantsin E. Ya., Hirsch E. A., Ivanov S. V., Vsemirnov M. A. Algorithms for SAT and upper bounds on their complexity.

We survey recent algorithms for the propositional satisfiability problem, in particular algorithms that have the best current worst-case upper bounds on their complexity. We also discuss some related issues: the derandomization of the algorithm of Paturi, Pudlák, Saks and Zane, the Valiant-Vazirani Lemma, and random walk algorithms with the "back button".

С.-Петербургское отделение
Математического института
им. В. А. Стеклова РАН

Поступило 15 января 2001