Записки научных
семинаров ПОМИ
Том 293, 2002 г.

С. И. Николенко

ТРУДНЫЕ ВЫПОЛНИМЫЕ ФОРМУЛЫ ДЛЯ DPLL–ПОДОБНЫХ АЛГОРИТМОВ

1. Введение

Выполнимость – одна из наиболее хорошо изученных NP-полных задач. Для решения задачи выполнимости существует два основных типа алгоритмов: алгоритмы локального поиска (ссылки см., например, в обзоре [11]) и DPLL-подобные алгоритмы (берущие начало от работ Дэвиса и Патнема [4] и Дэвиса, Лоджмана и Лавлэнда [3]). Известно большое количество результатов о меньших, чем 2^N, верхних оценках времени работы этих алгоритмов. В настоящей статье мы докажем экспоненциальные нижние оценки для двух DPLL-подобных алгоритмов: GUC (Generalized Unit Clause heuristic; впервые описан в [2]) и Randomized GUC.

DPLL-подобные алгоритмы были первыми, для которых были получены верхние оценки, меньшие, чем 2^N (равно как и экспоненциальные нижние оценки). На вход такой алгоритм получает формулу F в КНФ с переменными $x_1, \ldots, x_N$. Затем DPLL-подобный алгоритм упрощает входную формулу согласно определенному набору *правил преобразования*. Если теперь ответ на вопрос о выполнимости очевиден (упрощенная формула либо пуста, либо содержит пару противоречащих друг другу клозов, состоящих из одной переменной), алгоритм возвращает ответ. В противном случае он выбирает литерал l из формулы согласно некоторой эвристике, которая является одной из основных характеристик такого алгоритма. Затем алгоритм порождает две формулы, одну – для случая $l := true$, а другую – в предположении $l := false$, и рекурсивно вызывает себя от этих двух формул (отметим, что, так как мы подсчитываем общее время работы алгоритма, порядок, в котором он обрабатывает полученные формулы, значения не имеет). Если любой из вызовов выдает ответ "Выполнима", алгоритм также выдает этот ответ. В противном случае алгоритм выдает "Невыполнима". Таким образом, алгоритмы могут отличаться друг от друга в двух местах: либо в правилах упрощения формулы, либо в эвристике выбора следующего литерала.

Суперполиномиальные нижние оценки для метода резолюций (и, как следствие, для DPLL-подобных алгоритмов) были впервые получены в [9]. В [1] авторы рассмотрели вероятностное распределение булевых формул и доказали экспоненциальные нижние оценки для двух DPLL-подобных алгоритмов: GUC и UC (фактически было доказано, что вероятность появления экспоненциально сложной формулы отделена от нуля). Среди них предположительно должны существовать выполнимые формулы, но этот вопрос остается открытым. Экспоненциально сложные формулы были найдены и для алгоритмов локального поиска (см. [7]). Однако до сих пор не были опубликованы результаты об экспоненциально сложных *доказуемо выполнимых* формулах для DPLL-подобных алгоритмов. В настоящей статье мы рассмотрим такие примеры. Раздел 2 посвящен основным определениям, в разделе 3 рассмотрены примеры трудных невыполнимых формул, разделы 4 и 5 содержат доказательства нижних оценок времени работы алгоритмов, а в разделе 6 сформулированы открытые вопросы.

2. Основные определения

Обозначим через X множество булевых переменных. Отрицание переменной x обозначается через $\overline{x}$. Если $U \subseteq X$, то $\overline{U} = \{\overline{x} \mid x \in U\}$. *Литералы* – это элементы множества $X \cup \overline{X}$. *Клоз* – это множество литералов, в котором одновременно не встречается литерал и его отрицание. *Формула в КНФ* – это конечное множество клозов. Клоз мы будем называть *единичным*, если он состоит из одного литерала. Литерал назовем *чистым* по отношению к формуле, если эта формула содержит сам литерал, но не содержит его отрицание. Будем обозначать через $PL(F)$ множество всех чистых литералов формулы F.

Набор – это конечное подмножество $I \subseteq X \cup \overline{X}$, которое не может содержать литерала вместе со своим отрицанием. Будем обозначать через $F[I]$ формулу, которая получается из формулы F и набора $I = \{x_1, x_2, \ldots, x_n\}$ после удаления из нее всех клозов, содержащих литералы x_i и удаления из оставшихся клозов всех вхождений литералов $\overline{x}_i$. Набор I называется *выполняющим* формулу F, если $F[I]$ – пустая формула (т.е. $F[I]$ не содержит ни одного клоза).

Для формулы $F(x_1, \dots, x_n)$ построим его бинарное *дерево наборов*. Узлы дерева – частичные наборы для F, состоящие из литералов $x_1, \dots, x_n$ или их отрицаний, а потомки узла $I = \{l_1, \dots, l_i\}$, где $l_j \in \{x_j, \overline{x_j}\}$ – наборы $I_1 = \{l_1, \dots, l_i, x_{i+1}\}$ и $I_2 = \{l_1, \dots, l_i, \overline{x_{i+1}}\}$. Следуя [2], будем обозначать через C_i^F множество клозов в F, содержащих в точности i литералов каждый (верхний индекс будем опускать, если он очевиден из контекста).

3. Трудные невыполнимые формулы

Первые примеры невыполнимых формул, требующих суперполиномиальной длины вывода методом резолюций (который, как показано в [5], со сложностной точки зрения эквивалентен процедуре Дэвиса–Патнема) были приведены Цейтиным в [9]. Примеры в статье Цейтина были построены с использованием булевских формул, основанных на графах. Цейтин использовал довольно простые графы, и его результаты были улучшены Галилом в [5]. В [10], используя результаты о графах-расширителях, оценки были улучшены до 2^{cN}, где N – число переменных в формуле (в будущем c будет обозначать именно эту константу). Отметим, что доказанные в этой статье результаты зависят от известных оценок для невыполнимых формул и, таким образом, будут автоматически улучшены, если будет увеличена вышеупомянутая константа c.

Приведем следующую теорему из [10] (S_m – построенная ранее в той же статье формула, существование которой доказано для всех m):

Теорема 1 ([10], 5.7). *Существует такая константа $c > 1$, что для достаточно больших m всякое доказательство невыполнимости формулы S_m методом резолюций содержит c^n различных клозов, где длина S_m составляет $O(n), n = m^2$.*

В [8], используя обобщение цейтинских тавтологий, был получен следующий результат: для всякого $k \geqslant 3$ существует константа $c_k > 0$, $c_k = O(1/k^{1/8})$, такая, что время работы всякого DPLL-подобного алгоритма для k-SAT в наихудшем случае – по меньшей мере $\Omega(2^{N(1-c_k)})$, где N – число переменных в формуле.

Также стоит упомянуть, что формулы в [8] и [10] имеют линейное число клозов, т.е. существует такая константа b, что эти формулы имеют менее чем bN клозов, где N – число переменных в них.

В дальнейшем будем обозначать через $G_k(y_1, \ldots, y_N)$ сложную формулу в k-КНФ из [8] с N переменными $y_1, \ldots, y_N$.

4. Сложные формулы для GUC

Выбор литерала улучшенным алгоритмом GUC
Вход: формула F в КНФ.
Действия:

1. $m := min\{i: \ C_i^F \neq \emptyset\}$.
2. Если $m = 1$, выбрать C из C_1^F случайным образом и положить l равным единственному литералу из C.
3. Иначе, если $PL(F) \neq \emptyset$, выбрать l случайно из $PL(F)$.
4. Иначе случайно выбрать C из C_m^F, затем выбрать l из C случайным образом.
5. Выдать l.

Рис. 1. Один шаг улучшенного алгоритма GUC.

Алгоритм GUC описан в [2], и его эвристика выбора следующего литерала показана на рис. 1. Этот алгоритм выбирает случайный литерал, выполняющий случайный клоз наименьшей длины. По сравнению с алгоритмом из [2], мы добавили в эвристику правило чистых литералов, т.е., если отрицание литерала в формуле не встречается, алгоритм автоматически устанавливает этому литералу истинное значение. Очевидно, проверка на чистые литералы может быть выполнена за полиномиальное время (по отношению к числу переменных в формуле). Также очевидно, что применение правила чистых литералов не может сделать текущий набор противоречивым. Стоит отметить, что наши оценки также верны для исходного алгоритма GUC (и, в следующем разделе, алгоритма Randomized GUC без правила чистых литералов вместо описанной ниже версии); на самом деле правило чистых литералов может только ухудшить наши оценки.

В этой статье мы используем реализацию этого алгоритма, описанную, например, в [1]. Каждый раз, когда алгоритм расщепляет набор по какой-либо переменной, он делает выбор, и в данном случае именно число таких выборов является мерой эффективности алгоритма. Когда алгоритм впервые достигает какого-либо узла, он отмечает свой выбор как *искусственный*, если он был сделан при помощи правил преобразования. В нашем случае, такой выбор имеет место, если формула содержит либо единичный клоз, либо чистый литерал. В противном случае выбор называется *свободным*. Данная реализация GUC будет "спускаться по дереву выборов", пока не найдет противоречие, а затем вернется к последнему свободному выбору. После этого алгоритм изменит сделанный выбор на противоположный, отметит этот выбор как искусственный и продолжит работу. Мы будем измерять сложность работы алгоритма как количество выборов (как искусственных, так и свободных), которые ему придется сделать, прежде чем он найдет выполняющий набор.

Перейдем теперь к доказательству экспоненциальных нижних оценок времени работы алгоритма на выполнимых формулах. Рассмотрим следующую формулу (обозначив через $x \vee E$ множество клозов, полученных после добавления x ко всем клозам в E):

$$\begin{aligned} F = (x_1 \vee G_k(x_{M+1}, \ldots, x_{\lceil \frac{1+c_k}{c_k} M \rceil}))\wedge \\ \wedge(\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4}) \wedge \cdots \\ \wedge(\overline{x_1} \vee x_{M-1} \vee \overline{x_M}) \wedge (\overline{x_1} \vee x_M \vee \overline{x_2}) \end{aligned}$$

Отметим, что вторая часть формулы соответствует $\overline{x_1} \vee H$, где H – формула, вынуждающая переменные $x_2, \ldots, x_M$ принимать одинаковые значения. Отметим также, что если G_k – формула в k-КНФ, то F – формула в $(k+1)$-КНФ (и ее первая часть – также формула в $(k+1)$-КНФ). На первом шаге GUC выполняет случайный литерал из случайного клоза минимальной длины. С вероятностью $\frac{1}{3}$ этот литерал – $\overline{x_1}$. В таком случае формула F принимает вид $G_k(x_{M+1}, \ldots, x_{\lceil \frac{1+c_k}{c_k} M \rceil})$, и алгоритм, по [10, Теорема 5.7], должен будет сделать по крайней мере $poly(M)2^M$ выборов для того, чтобы выбраться из этой части дерева выборов.

С вероятностью $\frac{2}{3}$ GUC выберет другой литерал l. Пусть это будет $l = x_2$ (вследствие симметричности формулы не важно, какой именно литерал выберет алгоритм).

$$F[x_2] = (x_1 \vee G(x_{M+1}, \dots, x_{\lceil \frac{1+c_k}{c_k} M \rceil})) \wedge$$
$$\wedge (\overline{x_1} \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_4 \vee \overline{x_5}) \wedge \dots \wedge (\overline{x_1} \vee x_{M-1} \vee \overline{x_M}) \wedge (\overline{x_1} \vee x_M)$$

Теперь в формуле есть 2-клоз, и на следующем шаге GUC либо, с вероятностью $\frac{1}{2}$, выполнит $\overline{x_1}$, породив таким образом трудную невыполнимую формулу, либо выполнит x_M, и формула примет вид

$$F[x_2, x_M] = (x_1 \vee G(x_{M+1}, \dots, x_{\lceil \frac{1+c_k}{c_k} M \rceil})) \wedge$$
$$\wedge (\overline{x_1} \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_4 \vee \overline{x_5}) \wedge \dots \wedge (\overline{x_1} \vee x_{M-1})$$

Только когда 3-клозов больше не останется, последний оставшийся литерал станет чистым, и последний 2-клоз разрешится автоматически (выбор будет искусственным). Значит, по индукции, вероятность того, что алгоритм установит $x_1 = false$ (и будет вынужден работать в течение $poly(M)2^M$) равна

$$P(x_1 = false) = 1 - \frac{2}{3} 2^{-M+3},$$

что стремится к 1 экспоненциально быстро при $M \longrightarrow \infty$.

Если теперь обозначить через N общее количество переменных в формуле, все вышесказанное доказывает следующую теорему.

Теорема 2. *Для всякого $k \geqslant 4$ существует такое множество выполнимых формул F_N^k в k-КНФ, что улучшенный алгоритм GUC сделает по меньшей мере $poly(N)2^{\frac{c_{k-1}}{1+c_{k-1}}N}$ выборов, прежде чем найдет выполняющий набор для F_N^k, причем F_N^k содержит N переменных и не более чем aN клозов, где a – константа, не зависящая от N, и $c_k = O(1/k^{1/8})$.*

5. Трудные формулы для Randomized GUC

Могло показаться, что мы достигли успеха с алгоритмом GUC лишь вследствие его строго определенного поведения. Например, проблема могла быть в обязательном *выполнении* кратчайшего клоза. Наша формула из предыдущего раздела "заманивает" GUC в неверное дерево именно из-за этой особенности его

Выбор литерала улучшенным алгоритмом Randomized GUC

Вход: формула F в КНФ.

Действия:

1. $m := min\{i : C_i^F \neq \emptyset\}$.
2. Если $m = 1$, выбрать C из C_1^F случайным образом и положить l равным единственному литералу из C.
3. Иначе, если $PL(F) \neq \emptyset$, выбрать l случайно из $PL(F)$.
4. Иначе случайно выбрать C из C_m^F, затем выбрать l из $C \cup \overline{C}$ случайным образом.
5. Вынести l.

Рис. 2: Один шаг алгоритма Randomized GUC.

поведения. В данном разделе, мы приведем пример трудной выполнимой формулы для модификации алгоритма GUC, а именно алгоритма Randomized GUC. Один шаг этого алгоритма показан на рис. 2. Он выбирает литерал случайным образом из кратчайшего клоза, но также случайно выбирает, выполнять его или нет. Например, если кратчайший клоз имеет вид $a \vee b$, Randomized GUC может выбрать любой из литералов $\{a, b, \overline{a}, \overline{b}\}$.

Randomized GUC легко расправился бы с примером из предыдущего раздела. Действительно, на самом первом шаге он с вероятностью $\frac{1}{6}$ установил бы $x_1 = true$, сводя таким образом формулу к очень простому виду. И, перезапуская Randomized GUC, можно было бы достичь сколь угодно близкой к 1 вероятности успеха.

Рассмотрим следующую формулу (обозначая $G := G_k(x_{M+1}, \ldots, x_{\lceil \frac{3c_k+2}{3c_k} M \rceil})$ и предположив $3 \mid M$ без потери общности):

$$F = (x_1 \vee G) \wedge (x_2 \vee G) \wedge \ldots \wedge (x_M \vee G) \wedge$$
$$\wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_1}) \wedge (x_3 \vee \overline{x_1} \vee \overline{x_2}) \wedge$$
$$\wedge (x_4 \vee \overline{x_5} \vee \overline{x_6}) \wedge (x_5 \vee \overline{x_6} \vee \overline{x_4}) \wedge (x_6 \vee \overline{x_4} \vee \overline{x_5}) \wedge \ldots$$
$$\ldots \wedge (x_{M-2} \vee \overline{x_{M-1}} \vee \overline{x_M}) \wedge (x_{M-1} \vee \overline{x_M} \vee \overline{x_{M-2}})$$
$$\wedge (x_{M-1} \vee \overline{x_M} \vee \overline{x_{M-2}}).$$

Как и в случае из предыдущего раздела, набор является выполняющим для F тогда и только тогда, когда значения переменных $x_1, x_2, \ldots, x_M$ в нем равны $true$.

Алгоритм Randomized GUC сначала выбирает случайный клоз среди кратчайших, т.е. среди второй части нашей формулы, а затем случайный литерал из выбранного клоза. Так как все литералы входят в формулу симметрично, на самом деле, как нетрудно проверить, он выбирает случайный литерал среди $x_1, \ldots, x_M, \overline{x_1}, \ldots, \overline{x_M}$ равновероятным образом. Прежде всего отметим, что, если он выберет любое из отрицаний переменных, то $F[\overline{x_i}]$ будет содержать G в качестве независимой подформулы. Алгоритму потребуется экспоненциально большое время для доказательства ее невыполнимости, так как противоречие может быть достигнуто только в ней (легко видеть, что остальные кластеры по три клоза не могут быть сведены алгоритмом Randomized GUC к пустому клозу, и, таким образом, там противоречие невозможно). Итак, с вероятностью $\frac{1}{2}$, желаемый результат достигнут. Предположим, что алгоритм избрал x_1 (без потери общности вследствие симметрии формулы относительно первых M переменных). Теперь формула содержит два 2-клоза, $(\overline{x_2} \vee x_3)$ и $(\overline{x_3} \vee x_2)$. Алгоритм должен теперь сделать свободный выбор с вероятностью успеха $\frac{1}{2}$ (такова вероятность того, что он выберет x_2 или x_3, а не их отрицания). Если алгоритм делает правильный выбор, в формуле образуется единичный клоз, и выбор значения последней переменной автоматически верен.

Короче говоря, в каждом кластере из трех 3-клозов с одинаковыми переменными алгоритм с вероятностью $\frac{1}{4}$ установит верные значения переменных, и алгоритм рассматривает эти кластеры один за другим, по одному.

Таким образом, общая вероятность успеха равна

$$P(\forall i : 1 \leqslant i \leqslant M \;\; x_i = true) = \frac{1}{2} 2^{-\frac{2}{3}M}.$$

И, в случае неудачи, алгоритму Randomized GUC придется для доказательства невыполимости G работать в течение $poly(M) 2^{\frac{2}{3}M}$. Все вышесказанное доказывает следующую теорему.

Теорема 3. *Для всякого $k \geqslant 4$ существует множество выполнимых формул F_N^k в k-КНФ, такой, что алгоритм Randomized GUC сделает по меньшей мере $poly(N) 2^{\frac{2c_{k-1}}{2+3c_{k-1}}N}$ выборов, прежде чем* *найдет выполняющий набор для* F_N^k, *причем* F_N^k *содержат* N *переменных и не более чем* aN^2 *клозов, где* a *– константа, не зависящая от* N, *и* $c_k = O(1/k^{1/8})$.

6. Дальнейшая работа

В этой статье мы доказали экспоненциальную нижнюю оценку на время работы двух DPLL-подобных алгоритмов на выполнимых формулах. Однако количество клозов в "трудных" формулах для алгоритма Randomized GUC имеет квадратичную зависимость от количества переменных. Было бы интересно привести аналогичные формулы линейной длины.

Кроме того, помимо эвристик выбора единичного клоза и чистого литерала, современные DPLL-подобные алгоритмы используют и другие эвристики, такие, как правило резолюции, принцип "черных и белых литералов" и т.п. (подробный обзор приведен в [11]). Аналогичные оценки для этих эвристик еще не доказаны.

Благодарности

Я хотел бы поблагодарить Э. А. Гирша за постановку проблемы и поддержку при ее решении.

Литература

1. D. Achlioptas, P. Beame, M. Molloy, *A sharp threshold in proof complexity.* — In Proc. 33th ACM Symp. on Theory of Computing, (2001).
2. M.-N. Chao, J. Franco, *Probabilistic analysis of a generalization of the unit clause literal selection heuristic for the k-satisfiability problem.* — Information Science, **51** (1990), 289–314.
3. M. Davis, G. Logemann, D. Loveland, *A machine program for theorem-proving.* — Communications of the ACM **5(7)** (1962), 394–397.
4. M. Davis, H. Putnam, *A computing procedure for quantification theory.* — Journal of the ACM, **7(3)** (1960), 201–215.
5. Z. Galil, *On the complexity of regular resolution and the Davis-Putnam procedure.* Theoretical Computer Science, **4** (1977), 27–46.
6. E. A. Hirsch, *New worst-case upper bounds for SAT.* — Journal of Automated Reasoning, **24** (2000), 397–420.
7. E. A. Hirsch, *SAT local search algorithms: worst-case study.* — Journal of Automated Reasoning, **24** (2000), 127–143.
8. P. Pudlak, R. Impagliazzo, *A lower bound for DLL-algorithms for k-SAT.* — Proc. SODA-2000. (2000).

9. G. S. Tseitin, *On the complexity of derivation in propositional calculus.* — Studies in Constructive Mathematics and Matematical Logic, Part II. Consultants Bureau, New York–London, 1968, 115–125.
10. A. Urquhart, *Hard Examples for Resolution.* — The Bulletin of Symbolic Logic, Vol. 1, No. 4, Dec. 1995.
11. М. А. Всемирнов, Э. А. Гирш, Е. Я. Данцин, С. В. Иванов, *Алгоритмы для пропозициональной выполнимости и верхние оценки их сложности.* — Запис. научн. семин. ПОМИ, **277** (2001), 14–46.

Nikolenko S. I. Hard satisfiable formulas for DPLL-type algorithms.

We address lower bounds on the time complexity of algorithms solving the propositional satisfiability problem. Namely, we consider two DPLL-type algorithms, enhanced with the unit clause and pure literal heuristics. Exponential lower bounds for solving satisfiability on provably *satisfiable* formulas are proven.

Санкт-Петербургский
государственный университет
E-mail: smartnik@inbox.ru