

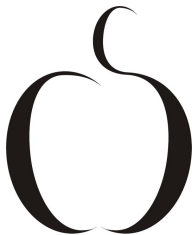
с/к “Эффективные алгоритмы”

Лекция 5: Алгоритмы для задачи выполнимости

А. Куликов

Computer Science клуб при ПОМИ

<http://logic.pdmi.ras.ru/~infclub/>



План лекции

- 1 Расщепление
 - Правила упрощения
 - Оценка времени работы
 - DPLL
 - PPSZ

План лекции

- 1 Расщепление
 - Правила упрощения
 - Оценка времени работы
 - DPLL
 - PPSZ
- 2 Случайное блуждание

Типы алгоритмов для SAT

Расщепляющие алгоритмы

Типы алгоритмов для SAT

Расщепляющие алгоритмы

- **расщепляющие алгоритмы** сводят задачу для входной формулы F к задаче для более простых формул F_1, \dots, F_j

Типы алгоритмов для SAT

Расщепляющие алгоритмы

- **расщепляющие алгоритмы** сводят задачу для входной формулы F к задаче для более простых формул F_1, \dots, F_j
 - ▶ **DPLL-подобные алгоритмы**, как правило, заменяют входную формулу F на две формулы $F[x = 1]$ и $F[x = 0]$; время работы оценивается из рекуррентных соотношений для количества листьев в дереве рекурсии

Типы алгоритмов для SAT

Расщепляющие алгоритмы

- **расщепляющие алгоритмы** сводят задачу для входной формулы F к задаче для более простых формул F_1, \dots, F_j
 - ▶ **DPLL-подобные алгоритмы**, как правило, заменяют входную формулу F на две формулы $F[x = 1]$ и $F[x = 0]$; время работы оценивается из рекуррентных соотношений для количества листьев в дереве рекурсии
 - ▶ **PPSZ-подобные алгоритмы** присваивают значения переменным случайно; время работы оценивается “глобально”

Типа алгоритмов для SAT (продолжение)

Локальный поиск

Типа алгоритмов для SAT (продолжение)

Локальный поиск

- **алгоритмы, основанные на локальном поиске** берут начальный набор и изменяют его шаг за шагом, пытаясь приблизиться к выполняющему набору; если через некоторое время выполняющий набор так и не был найден, порождается другой начальный набор

Типа алгоритмов для SAT (продолжение)

Локальный поиск

- **алгоритмы, основанные на локальном поиске** берут начальный набор и изменяют его шаг за шагом, пытаясь приблизиться к выполняющему набору; если через некоторое время выполняющий набор так и не был найден, порождается другой начальный набор
 - ▶ **жадные алгоритмы** выбирают переменную и изменяют ее значение на противоположное, так чтобы как можно больше увеличить значение некоторой функции от набора (к примеру, кол-во выполненных клозов)

Типа алгоритмов для SAT (продолжение)

Локальный поиск

- **алгоритмы, основанные на локальном поиске** берут начальный набор и изменяют его шаг за шагом, пытаясь приблизиться к выполняющему набору; если через некоторое время выполняющий набор так и не был найден, порождается другой начальный набор
 - ▶ **жадные алгоритмы** выбирают переменную и изменяют ее значение на противоположное, так чтобы как можно больше увеличить значение некоторой функции от набора (к примеру, кол-во выполненных клозов)
 - ▶ **алгоритмы, основанные на случайных блужданиях** меняют значение случайной переменной; время работы можно оценить, используя связь с одномерными случайными блужданиями

План лекции

- 1 Расщепление
 - Правила упрощения
 - Оценка времени работы
 - DPLL
 - PPSZ
- 2 Случайное блуждание

Пример работы алгоритма расщепления

$$(x \vee y \vee z) \wedge (\neg x \vee y) \wedge (\neg y \vee z) \wedge (\neg y \vee \neg z)$$

Пример работы алгоритма расщепления

$$(x \vee y \vee z) \wedge (\neg x \vee y) \wedge (\neg y \vee z) \wedge (\neg y \vee \neg z)$$

$$x = 1$$


$$(y) \wedge (\neg y \vee z) \wedge (\neg y \vee \neg z)$$

Пример работы алгоритма расщепления

$$(x \vee y \vee z) \wedge (\neg x \vee y) \wedge (\neg y \vee z) \wedge (\neg y \vee \neg z)$$

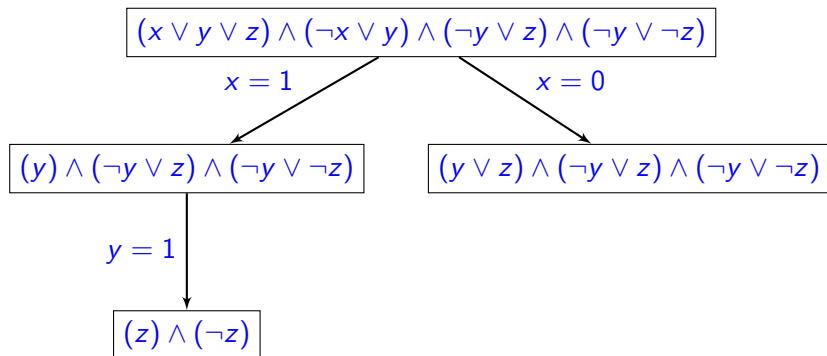
$$x = 1$$

$$(y) \wedge (\neg y \vee z) \wedge (\neg y \vee \neg z)$$

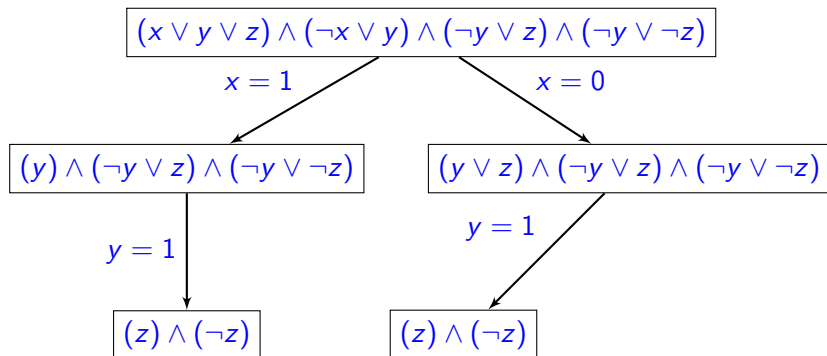
$$y = 1$$

$$(z) \wedge (\neg z)$$

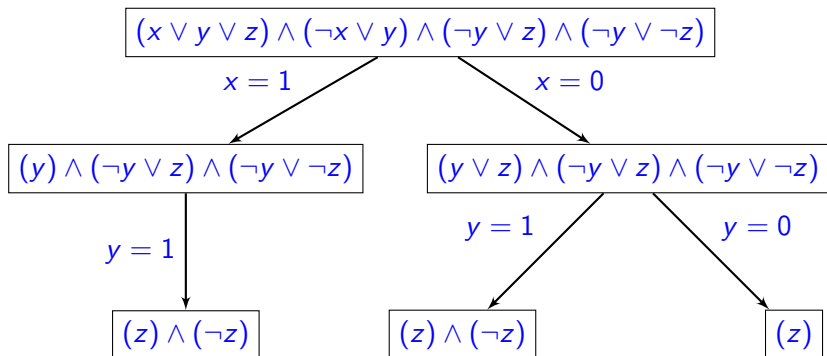
Пример работы алгоритма расщепления



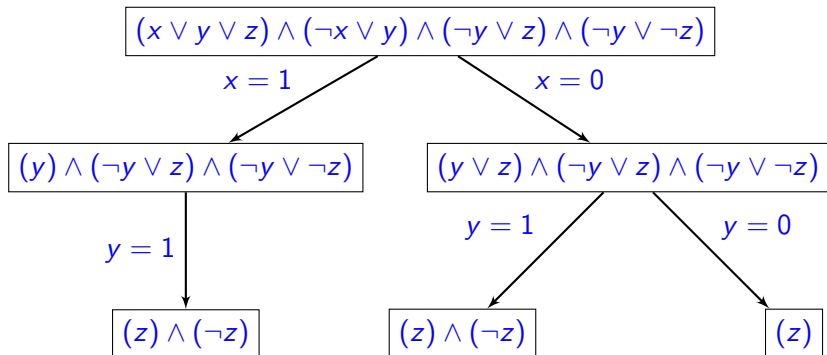
Пример работы алгоритма расщепления



Пример работы алгоритма расщепления

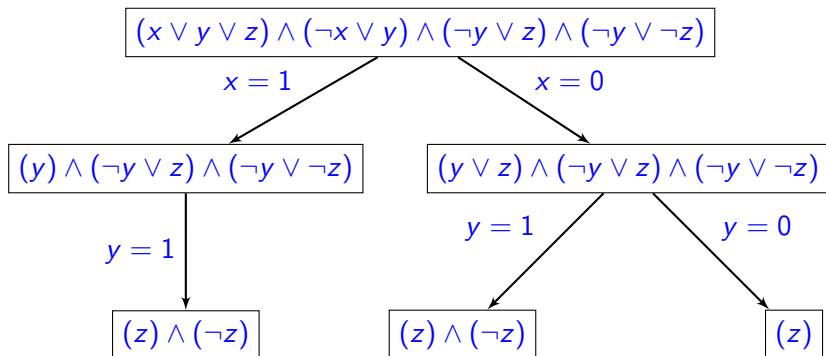


Пример работы алгоритма расщепления



$\{x = 0, y = 0, z = 1\}$ — выполняющий набор

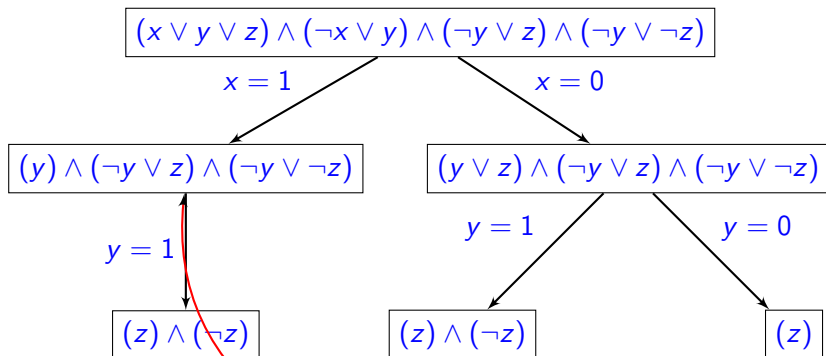
Пример работы алгоритма расщепления



как улучшить?

$\{x = 0, y = 0, z = 1\}$ — выполняющий набор

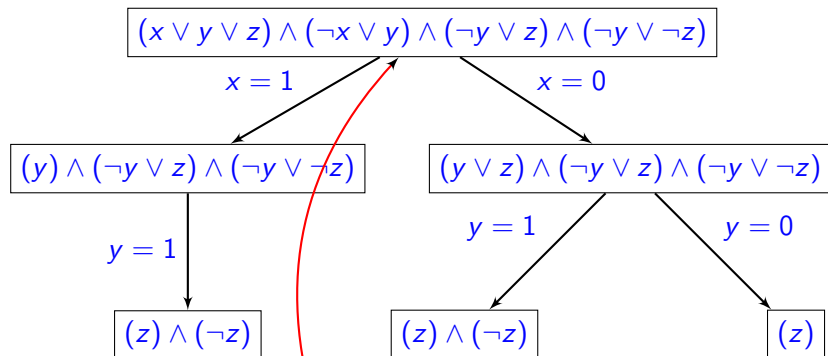
Пример работы алгоритма расщепления



упростить формулу перед расщеплением

$\{x = 0, y = 0, z = 1\}$ — выполняющий набор

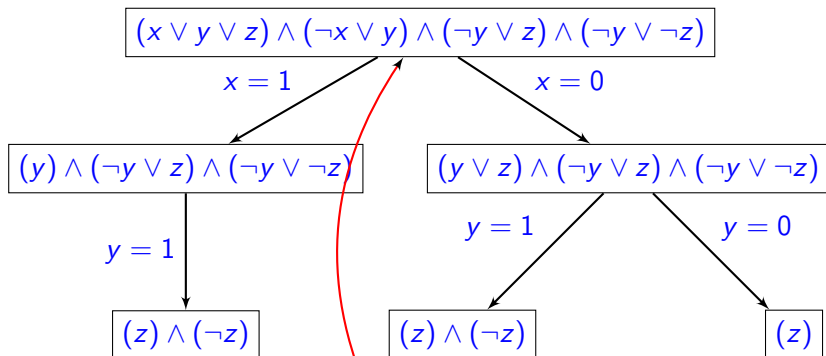
Пример работы алгоритма расщепления



мы могли бы рассмотреть сначала случай $x = 0$

$\{x = 0, y = 0, z = 1\}$ — выполняющий набор

Пример работы алгоритма расщепления



использование в правой ветке информации из левой

$\{x = 0, y = 0, z = 1\}$ — выполняющий набор

План лекции

1 Расщепление

- Правила упрощения
- Оценка времени работы
- DPLL
- PPSZ

2 Случайное блуждание

Правила упрощения

Определение

К формуле F применимо **правило упрощения** (simplification/reduction rule), если по этому правилу F можно заменить на полиномиальное время на F' , так что:

Правила упрощения

Определение

К формуле F применимо **правило упрощения** (simplification/reduction rule), если по этому правилу F можно заменить за полиномиальное время на F' , так что:

- сложность F' меньше сложности F ,

Правила упрощения

Определение

К формуле F применимо **правило упрощения** (simplification/reduction rule), если по этому правилу F можно заменить за полиномиальное время на F' , так что:

- сложность F' меньше сложности F ,
- выполняющий набор для F можно построить из выполняющего набора для F' за полиномиальное время.

Единичный кюз

Единичный кюз

Если формула содержит **единичный кюз** (unit clause), то входящему в него литералу можно присвоить значение 1.

Единичный кюз

Единичный кюз

Если формула содержит **единичный кюз** (unit clause), то входящему в него литералу можно присвоить значение 1.

Пример

$$F = (x \vee y \vee \neg z) \wedge (\neg x) \wedge (z \vee \neg y) \wedge (\neg y \vee \neg x \vee \neg z)$$

$$F[x = 0] = (y \vee \neg z) \wedge (z \vee \neg y)$$

Чистый литерал

Чистый литерал

Если формула содержит **чистый литерал** (pure literal), т. е. литерал, отрицание которого не входит в формулу, то ему можно присвоить значение **1**.

Чистый литерал

Чистый литерал

Если формула содержит **чистый литерал** (pure literal), т. е. литерал, отрицание которого не входит в формулу, то ему можно присвоить значение 1.

Пример

$$F = (x \vee y \vee z) \wedge (\neg x) \wedge (x \vee \neg y) \wedge (\neg y \vee \neg x \vee z)$$

$$F[z = 1] = (\neg x) \wedge (x \vee \neg y)$$

Резолюция

Резолюция

Если $F = F' \wedge (x \vee C) \wedge (\neg x \vee D)$ и F' не содержит ни x , ни $\neg x$,
заменить F на $F' \wedge (C \vee D)$.

Пример

Резолюция

Резолюция

Если $F = F' \wedge (x \vee C) \wedge (\neg x \vee D)$ и F' не содержит ни x , ни $\neg x$, заменить F на $F' \wedge (C \vee D)$.

Пример

- $(x \vee \neg y \vee z) \wedge (\neg z) \wedge (z \vee u) \wedge (\neg u \vee \bar{x}) \wedge (y \vee z)$
 $\leftrightarrow (\neg y \vee z \vee \neg u) \wedge (\neg z) \wedge (z \vee u) \wedge (y \vee z)$

Резолюция

Резолюция

Если $F = F' \wedge (x \vee C) \wedge (\neg x \vee D)$ и F' не содержит ни x , ни $\neg x$, заменить F на $F' \wedge (C \vee D)$.

Пример

- $(x \vee \neg y \vee z) \wedge (\neg z) \wedge (z \vee u) \wedge (\neg u \vee \bar{x}) \wedge (y \vee z)$
 $\leftrightarrow (\neg y \vee z \vee \neg u) \wedge (\neg z) \wedge (z \vee u) \wedge (y \vee z)$
- $(x \vee y \vee \neg z) \wedge (\neg x) \wedge (x \vee \neg y) \wedge (\neg y \vee \neg x \vee z)$
 $\leftrightarrow (\neg x) \wedge (x \vee \neg y)$

План лекции

1 Расщепление

- Правила упрощения
- Оценка времени работы
- DPLL
- PPSZ

2 Случайное блуждание

Оценка времени работы алгоритмов расщепления

Пусть K — некоторая мера сложности формул в КНФ (к примеру, кол-во переменных или кол-во кловов).

Оценка времени работы алгоритмов расщепления

Пусть K — некоторая мера сложности формул в КНФ (к примеру, кол-во переменных или кол-во кловов).

Формально

Если алгоритм всегда расщепляет с рекуррентным неравенством вида

$$T(K) \leq T(K - t_1) + \dots + T(K - t_j) + \text{poly}(K),$$

то его время работы в худшем случае составляет

$$\text{poly}(|F|)\alpha^{K(F)},$$

$\alpha = \tau(t_1, \dots, t_j)$ — единственный положительный корень уравнения $x^{-t_1} + \dots + x^{-t_j} = 1$.

Оценка времени работы алгоритмов расщепления

Пусть K — некоторая мера сложности формул в КНФ (к примеру, кол-во переменных или кол-во кловов).

Формально

Если алгоритм всегда расщепляет с рекуррентным неравенством вида

$$T(K) \leq T(K - t_1) + \dots + T(K - t_j) + \text{poly}(K),$$

то его время работы в худшем случае составляет

$$\text{poly}(|F|)\alpha^{K(F)},$$

$\alpha = \tau(t_1, \dots, t_j)$ — единственный положительный корень уравнения $x^{-t_1} + \dots + x^{-t_j} = 1$.

(t_1, \dots, t_j) называется **вектором расщепления**, а $\alpha = \tau(t_1, \dots, t_j)$ — числом расщепления.

Оценка времени работы алгоритмов расщепления

Неформально

Чем меньше сложности формул, на которые алгоритм расщепляет, тем быстрее алгоритм работает.

Оценка времени работы алгоритмов расщепления

Неформально

Чем меньше сложности формул, на которые алгоритм расщепляет, тем быстрее алгоритм работает.

Пример

Оценка времени работы алгоритмов расщепления

Неформально

Чем меньше сложности формул, на которые алгоритм расщепляет, тем быстрее алгоритм работает.

Пример

- $T(K) \leq T(K - 2) + T(K - 3) \leftrightarrow 1.325^K$

Оценка времени работы алгоритмов расщепления

Неформально

Чем меньше сложности формул, на которые алгоритм расщепляет, тем быстрее алгоритм работает.

Пример

- $T(K) \leq T(K - 2) + T(K - 3) \leftrightarrow 1.325^K$
- $T(N) \leq 2 \cdot T(N - 6) + 2 \cdot T(N - 7) \leftrightarrow 1.239^N$

План лекции

1 Расщепление

- Правила упрощения
- Оценка времени работы
- **DPLL**
- PPSZ

2 Случайное блуждание

DPLL-подобный алгоритм

Алгоритм

DPLL-SAT(F)

DPLL-подобный алгоритм

Алгоритм

DPLL-SAT(F)

- применять правила упрощения единичный клуз, чистый литерал и резолюция до тех пор, пока хотя бы одно из них применимо

DPLL-подобный алгоритм

Алгоритм

DPLL-SAT(F)

- применять правила упрощения единичный кюз, чистый литерал и резолюция до тех пор, пока хотя бы одно из них применимо
- если F не содержит ни одного кюза, выдать “выполнима”

DPLL-подобный алгоритм

Алгоритм

DPLL-SAT(F)

- применять правила упрощения единичный кюз, чистый литерал и резолюция до тех пор, пока хотя бы одно из них применимо
- если F не содержит ни одного кюза, выдать “выполнима”
- если F содержит пустой кюз, выдать “невыполнима”

DPLL-подобный алгоритм

Алгоритм

DPLL-SAT(F)

- применять правила упрощения единичный клоз, чистый литерал и резолюция до тех пор, пока хотя бы одно из них применимо
- если F не содержит ни одного клоза, выдать “выполнима”
- если F содержит пустой клоз, выдать “невыполнима”
- если F содержит литерал, входящий в нее хотя бы два раза положительно и хотя бы два раза отрицательно, положить x равным этому литералу; в противном случае взять в качестве x любой литерал

DPLL-подобный алгоритм

Алгоритм

DPLL-SAT(F)

- применять правила упрощения единичный кюз, чистый литерал и резолюция до тех пор, пока хотя бы одно из них применимо
- если F не содержит ни одного кюза, выдать “выполнима”
- если F содержит пустой кюз, выдать “невыполнима”
- если F содержит литерал, входящий в нее хотя бы два раза положительно и хотя бы два раза отрицательно, положить x равным этому литералу; в противном случае взять в качестве x любой литерал
- вызвать DPLL-SAT($F[x = 1]$) и DPLL-SAT($F[x = 0]$)

DPLL-подобный алгоритм

Алгоритм

DPLL-SAT(F)

- применять правила упрощения единичный кюз, чистый литерал и резолюция до тех пор, пока хотя бы одно из них применимо
- если F не содержит ни одного кюза, выдать “выполнима”
- если F содержит пустой кюз, выдать “невыполнима”
- если F содержит литерал, входящий в нее хотя бы два раза положительно и хотя бы два раза отрицательно, положить x равным этому литералу; в противном случае взять в качестве x любой литерал
- вызвать DPLL-SAT($F[x = 1]$) и DPLL-SAT($F[x = 0]$)
- если хотя бы один из рекурсивных вызовов вернул ответ “выполнима”, выдать “выполнима”; в противном случае выдать “невыполнима”

Анализ алгоритма

Лемма

Время работы алгоритма DPLL-SAT не превосходит 1.415^K , где K — кол-во кловов входной формулы.

Анализ алгоритма

Лемма

Время работы алгоритма DPLL-SAT не превосходит 1.415^K , где K — кол-во кловов входной формулы.

Доказательство

Анализ алгоритма

Лемма

Время работы алгоритма DPLL-SAT не превосходит 1.415^K , где K — кол-во кловов входной формулы.

Доказательство

- достаточно показать, что в каждой ветке удаляется хотя бы два клова: $\tau(2, 2) = \sqrt{2} = 1.414\dots$

Анализ алгоритма

Лемма

Время работы алгоритма DPLL-SAT не превосходит 1.415^K , где K — кол-во кловов входной формулы.

Доказательство

- достаточно показать, что в каждой ветке удаляется хотя бы два клова: $\tau(2, 2) = \sqrt{2} = 1.414\dots$
- это ясно, если нашелся литерал, который входит в формулу хотя бы дважды и положительно и отрицательно

Анализ алгоритма

Лемма

Время работы алгоритма DPLL-SAT не превосходит 1.415^K , где K — кол-во кловов входной формулы.

Доказательство

- достаточно показать, что в каждой ветке удаляется хотя бы два клова: $\tau(2, 2) = \sqrt{2} = 1.414\dots$
- это ясно, если нашелся литерал, который входит в формулу хотя бы дважды и положительно и отрицательно
- назовем (i, j) -литералом литерал, входящий в формулу ровно i раз положительно и ровно j раз отрицательно

Доказательство (продолжение)

Доказательство

Доказательство (продолжение)

Доказательство

- ясно, что упрощенная формула не содержит $(0, k)$ - и $(k, 0)$ -литералов (это как раз чистые литералы), а также $(1, 1)$ -литералов (они удаляются резольвированием)

Доказательство (продолжение)

Доказательство

- ясно, что упрощенная формула не содержит $(0, k)$ - и $(k, 0)$ -литералов (это как раз чистые литералы), а также $(1, 1)$ -литералов (они удаляются резольвированием)
- значит, осталось рассмотреть случай, когда формула состоит только из $(1, 2)$ - и $(2, 1)$ -литералов

Доказательство (продолжение)

Доказательство

Доказательство (продолжение)

Доказательство

- рассмотрим произвольный литерал x :

$$F = (x \vee \dots) \wedge (x \vee \dots) \wedge (\neg x \vee \dots) \wedge \dots$$

Доказательство (продолжение)

Доказательство

- рассмотрим произвольный литерал x :

$$F = (x \vee \dots) \wedge (x \vee \dots) \wedge (\neg x \vee \dots) \wedge \dots$$

- поскольку F не содержит единичных клозов, в клозе $(\neg x \vee \dots)$ есть еще хотя бы один литерал; назовем его y

Доказательство (продолжение)

Доказательство

- рассмотрим произвольный литерал x :

$$F = (x \vee \dots) \wedge (x \vee \dots) \wedge (\neg x \vee \dots) \wedge \dots$$

- поскольку F не содержит единичных клозов, в клозе $(\neg x \vee \dots)$ есть еще хотя бы один литерал; назовем его y
- в $F[x = 0]$ y будет $(2, 0)$ -, $(0, 2)$ - или $(1, 1)$ -литералом

Доказательство (продолжение)

Доказательство

- рассмотрим произвольный литерал x :

$$F = (x \vee \dots) \wedge (x \vee \dots) \wedge (\neg x \vee \dots) \wedge \dots$$

- поскольку F не содержит единичных кловов, в клозе $(\neg x \vee \dots)$ есть еще хотя бы один литерал; назовем его y
- в $F[x = 0]$ y будет $(2, 0)$ -, $(0, 2)$ - или $(1, 1)$ -литералом
- в любом из этих случаев y будет удален правилами упрощения, что повлечет за собой удаление хотя бы одного клова

Доказательство (продолжение)

Доказательство

- рассмотрим произвольный литерал x :

$$F = (x \vee \dots) \wedge (x \vee \dots) \wedge (\neg x \vee \dots) \wedge \dots$$

- поскольку F не содержит единичных кловов, в клове $(\neg x \vee \dots)$ есть еще хотя бы один литерал; назовем его y
- в $F[x = 0]$ y будет $(2, 0)$ -, $(0, 2)$ - или $(1, 1)$ -литералом
- в любом из этих случаев y будет удален правилами упрощения, что повлечет за собой удаление хотя бы одного клова
- итак, и $F[x = 1]$, и $F[x = 0]$ (после упрощения) содержат хотя бы на два клова меньше, чем F



Замечание

Замечание

Замечание

Замечание

- представленный алгоритм является одним из простейших; известные более хитрые правила упрощения и эвристики для расщеплений

Замечание

Замечание

- представленный алгоритм является одним из простейших; известные более хитрые правила упрощения и эвристики для расщеплений
- на методе расщепления основано большинство современных полных (т.е. всегда выдающих правильный ответ) SAT-солверов

Замечание

Замечание

- представленный алгоритм является одним из простейших; известные более хитрые правила упрощения и эвристики для расщеплений
- на методе расщепления основано большинство современных полных (т.е. всегда выдающих правильный ответ) SAT-солверов
- большинство теоретических верхних оценок также доказаны при помощи метода расщепления

План лекции

1 Расщепление

- Правила упрощения
- Оценка времени работы
- DPLL
- PPSZ

2 Случайное блуждание

Основные идеи PPSZ-подобного алгоритма

Основные идеи

Основные идеи PPSZ-подобного алгоритма

Основные идеи

- выберем случайную перестановку переменных и будем расщеплять по переменным в соответствующем порядке

Основные идеи PPSZ-подобного алгоритма

Основные идеи

- выберем случайную перестановку переменных и будем расщеплять по переменным в соответствующем порядке
- возможно, расщеплять нужно будет не по всем переменным, поскольку некоторые переменные могут получить значения в результате применения правил упрощения

Основные идеи PPSZ-подобного алгоритма

Основные идеи

- выберем случайную перестановку переменных и будем расщеплять по переменным в соответствующем порядке
- возможно, расщеплять нужно будет не по всем переменным, поскольку некоторые переменные могут получить значения в результате применения правил упрощения
- для оценки времени работы будем считать, по скольким переменным нам расщеплять не пришлось

PPSZ-подобный алгоритм

Алгоритм

PPSZ-SAT(F)

PPSZ-подобный алгоритм

Алгоритм

PPSZ-SAT(F)

- выбрать случайным образом перестановку π из множества всех перестановок $\{1, \dots, n\}$

PPSZ-подобный алгоритм

Алгоритм

PPSZ-SAT(F)

- выбрать случайным образом перестановку π из множества всех перестановок $\{1, \dots, n\}$
- построить дерево рекурсии глубины не более $2n/3$, где на каждом шаге

PPSZ-подобный алгоритм

Алгоритм

PPSZ-SAT(F)

- выбрать случайным образом перестановку π из множества всех перестановок $\{1, \dots, n\}$
- построить дерево рекурсии глубины не более $2n/3$, где на каждом шаге
 - ▶ сначала применяем правило удаления единичных кловов,

PPSZ-подобный алгоритм

Алгоритм

PPSZ-SAT(F)

- выбрать случайным образом перестановку π из множества всех перестановок $\{1, \dots, n\}$
- построить дерево рекурсии глубины не более $2n/3$, где на каждом шаге
 - ▶ сначала применяем правило удаления единичных клозов,
 - ▶ а потом выбираем первую переменную из перестановки, все еще входящую в формулу, и расщепляемся по ней

PPSZ-подобный алгоритм

Алгоритм

PPSZ-SAT(F)

- выбрать случайным образом перестановку π из множества всех перестановок $\{1, \dots, n\}$
- построить дерево рекурсии глубины не более $2n/3$, где на каждом шаге
 - ▶ сначала применяем правило удаления единичных клозов,
 - ▶ а потом выбираем первую переменную из перестановки, все еще входящую в формулу, и расщепляемся по ней
- если на каком-то шаге выяснилось, что формула выполнима, выдать “выполнима”

PPSZ-подобный алгоритм

Алгоритм

PPSZ-SAT(F)

- выбрать случайным образом перестановку π из множества всех перестановок $\{1, \dots, n\}$
- построить дерево рекурсии глубины не более $2n/3$, где на каждом шаге
 - ▶ сначала применяем правило удаления единичных кловов,
 - ▶ а потом выбираем первую переменную из перестановки, все еще входящую в формулу, и расщепляемся по ней
- если на каком-то шаге выяснилось, что формула выполнима, выдать “выполнима”
- в противном случае выдать “невыполнима”

Анализ алгоритма

Анализ алгоритма

Анализ алгоритма

Анализ алгоритма

- очевидно, время работы этого алгоритма равно $\text{poly}(|F|) \cdot 2^{2n/3}$

Анализ алгоритма

Анализ алгоритма

- очевидно, время работы этого алгоритма равно $\text{poly}(|F|) \cdot 2^{2n/3}$
- мы покажем, что PPSZ-SAT с достаточно большой вероятностью находит правильное решение для формулы, у которой не более одного выполняющего набора

Анализ алгоритма

Анализ алгоритма

- очевидно, время работы этого алгоритма равно $\text{poly}(|F|) \cdot 2^{2n/3}$
- мы покажем, что PPSZ-SAT с достаточно большой вероятностью находит правильное решение для формулы, у которой не более одного выполняющего набора
- рассмотрим дерево расщепления алгоритма, однако без ограничения $2n/3$ на его глубину

Анализ алгоритма

Анализ алгоритма

- очевидно, время работы этого алгоритма равно $\text{poly}(|F|) \cdot 2^{2n/3}$
- мы покажем, что PPSZ-SAT с достаточно большой вероятностью находит правильное решение для формулы, у которой не более одного выполняющего набора
- рассмотрим дерево расщепления алгоритма, однако без ограничения $2n/3$ на его глубину
- если S — выполняющий набор для F , то это дерево содержит ветвь, ведущую в S

Анализ алгоритма

Анализ алгоритма

- очевидно, время работы этого алгоритма равно $\text{poly}(|F|) \cdot 2^{2n/3}$
- мы покажем, что PPSZ-SAT с достаточно большой вероятностью находит правильное решение для формулы, у которой не более одного выполняющего набора
- рассмотрим дерево расщепления алгоритма, 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

Анализ алгоритма

Анализ алгоритма

- очевидно, время работы этого алгоритма равно $\text{poly}(|F|) \cdot 2^{2n/3}$
- мы покажем, что PPSZ-SAT с достаточно большой вероятностью находит правильное решение для формулы, у которой не более одного выполняющего набора
- рассмотрим дерево расщепления алгоритма, 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
- некоторым переменным значения также присваиваются правилами упрощения

Анализ алгоритма (продолжение)

Анализ алгоритма

Анализ алгоритма (продолжение)

Анализ алгоритма

- под **описанием** набора S будем понимать набор, отвечающий всем присваиваниям во время расщеплений, а под **длиной** описания — кол-во переменных в описании

Анализ алгоритма (продолжение)

Анализ алгоритма

- под **описанием** набора S будем понимать набор, отвечающий всем присваиваниям во время расщеплений, а под **длиной** описания — кол-во переменных в описании
- ясно, что по описанию выполняющего набора легко восстановить и сам выполняющий набор: присваиваем переменным значения из описания, при появлении единичных клозов — присваиваем значения их литералам

Лемма о кодировании выполняющего набора

Лемма

Пусть F — одновыполнимая формула в 3-КНФ, а S — ее выполняющий набор. Рассмотрим описание относительно перестановки, случайно и равновероятно выбранной из множества всех перестановок. Тогда математическое ожидание длины описания S не превосходит $2n/3$.

Доказательство

Лемма о кодировании выполняющего набора

Лемма

Пусть F — одновыполнимая формула в 3-КНФ, а S — ее выполняющий набор. Рассмотрим описание относительно перестановки, случайно и равновероятно выбранной из множества всех перестановок. Тогда математическое ожидание длины описания S не превосходит $2n/3$.

Доказательство

- заметим, что для каждой переменной x из F найдется хотя бы один клуз, в котором S выполняет только литерал, соответствующий x :

Лемма о кодировании выполняющего набора

Лемма

Пусть F — одновыполнимая формула в 3-КНФ, а S — ее выполняющий набор. Рассмотрим описание относительно перестановки, случайно и равновероятно выбранной из множества всех перестановок. Тогда математическое ожидание длины описания S не превосходит $2n/3$.

Доказательство

- заметим, что для каждой переменной x из F найдется хотя бы один клоз, в котором S выполняет только литерал, соответствующий x : если бы такого клоза не нашлось, то, изменив значение x в S , мы получили бы другой выполняющий набор

Лемма о кодировании выполняющего набора

Лемма

Пусть F — одновыполнимая формула в 3-КНФ, а S — ее выполняющий набор. Рассмотрим описание относительно перестановки, случайно и равномерно выбранной из множества всех перестановок. Тогда математическое ожидание длины описания S не превосходит $2n/3$.

Доказательство

- заметим, что для каждой переменной x из F найдется хотя бы один клоз, в котором S выполняет только литерал, соответствующий x : если бы такого клоза не нашлось, то, изменив значение x в S , мы получили бы другой выполняющий набор
- клоз, удовлетворяющий такому условию, назовем **критическим** для переменной x

Доказательство (продолжение)

Доказательство

Доказательство (продолжение)

Доказательство

- поскольку π выбирается случайно и равновероятно, переменная x окажется в этой перестановке последней переменной критического клона для x с вероятностью не менее $1/3$

Доказательство (продолжение)

Доказательство

- поскольку π выбирается случайно и равновероятно, переменная x окажется в этой перестановке последней переменной критического клона для x с вероятностью не менее $1/3$
- ясно, что в таком случае x не появится в описании S

Доказательство (продолжение)

Доказательство

- поскольку π выбирается случайно и равновероятно, переменная x окажется в этой перестановке последней переменной критического клона для x с вероятностью не менее $1/3$
- ясно, что в таком случае x не появится в описании S
- таким образом, с вероятностью хотя бы $1/3$ x не попадет в описание S

Доказательство (продолжение)

Доказательство

- поскольку π выбирается случайно и равновероятно, переменная x окажется в этой перестановке последней переменной критического клона для x с вероятностью не менее $1/3$
- ясно, что в таком случае x не появится в описании S
- таким образом, с вероятностью хотя бы $1/3$ x не попадет в описание S
- осталось воспользоваться линейностью математического ожидания



Анализ алгоритма (продолжение)

Анализ алгоритма (продолжение)

- пусть p_l — вероятность того, что длина описания выполняющего набора в точности равна l

Анализ алгоритма (продолжение)

- пусть p_l — вероятность того, что длина описания выполняющего набора в точности равна l
- по только что доказанной лемме

$$\frac{2n}{3} \geq \sum_{l=0}^n p_l l \geq \left(\left\lfloor \frac{2n}{3} \right\rfloor + 1 \right) \sum_{l=\lfloor \frac{2n}{3} \rfloor + 1}^n p_l$$

Анализ алгоритма (продолжение)

- пусть p_l — вероятность того, что длина описания выполняющего набора в точности равна l
- по только что доказанной лемме

$$\frac{2n}{3} \geq \sum_{l=0}^n p_l l \geq \left(\left\lfloor \frac{2n}{3} \right\rfloor + 1 \right) \sum_{l=\left\lfloor \frac{2n}{3} \right\rfloor + 1}^n p_l$$

- следовательно,

$$\sum_{l=\left\lfloor \frac{2n}{3} \right\rfloor + 1}^n p_l \leq \frac{2n}{3} \left(\left\lfloor \frac{2n}{3} \right\rfloor + 1 \right)^{-1} \leq \frac{2n}{3} \cdot \frac{3}{2n+1} = 1 - \frac{1}{2n+1}$$

Анализ алгоритма (продолжение)

Анализ алгоритма (продолжение)

$$\sum_{l=0}^{\lfloor \frac{2n}{3} \rfloor + 1} p_l \geq \frac{1}{2n+1}$$

Анализ алгоритма (продолжение)

-

$$\sum_{l=0}^{\lfloor \frac{2n}{3} \rfloor + 1} p_l \geq \frac{1}{2n+1}$$

- итак, для случайной перестановки длина описания не превосходит $2n/3$ с вероятностью хотя бы $1/(2n+1)$

Анализ алгоритма (продолжение)

-

$$\sum_{l=0}^{\lfloor \frac{2n}{3} \rfloor + 1} p_l \geq \frac{1}{2n+1}$$

- итак, для случайной перестановки длина описания не превосходит $2n/3$ с вероятностью хотя бы $1/(2n+1)$
- если запустить алгоритм $(2n+1)$ раз, то алгоритм не найдет набор с вероятностью не более $(1 - \frac{1}{2n+1})^{2n+1} < \frac{1}{e}$

Анализ алгоритма (продолжение)

•

$$\sum_{l=0}^{\lfloor \frac{2n}{3} \rfloor + 1} p_l \geq \frac{1}{2n+1}$$

- итак, для случайной перестановки длина описания не превосходит $2n/3$ с вероятностью хотя бы $1/(2n+1)$
- если запустить алгоритм $(2n+1)$ раз, то алгоритм не найдет набор с вероятностью не более $(1 - \frac{1}{2n+1})^{2n+1} < \frac{1}{e}$
- а если запустить $c(2n+1)$ раз, то вероятность ошибки будет не более $\frac{1}{e^c}$



Основные идеи PPSZ-подобного алгоритма еще раз

Основные идеи еще раз

Основные идеи PPSZ-подобного алгоритма еще раз

Основные идеи еще раз

- случайно выбираем порядок переменных и расщепляем, периодически удаляя единичные клозы

Основные идеи PPSZ-подобного алгоритма еще раз

Основные идеи еще раз

- случайно выбираем порядок переменных и расщепляем, периодически удаляя единичные клозы
- если у F есть ровно один выполняющий набор S , то для любой переменной x найдется клоз $(x \vee y \vee z)$, такой что S выполняет только литерал x в этом клозе (то есть S присваивает x, y, z значения $1, 0, 0$, соответственно)

Основные идеи PPSZ-подобного алгоритма еще раз

Основные идеи еще раз

- случайно выбираем порядок переменных и расщепляем, периодически удаляя единичные клозы
- если у F есть ровно один выполняющий набор S , то для любой переменной x найдется клоз $(x \vee y \vee z)$, такой что S выполняет только литерал x в этом клозе (то есть S присваивает x, y, z значения $1, 0, 0$, соответственно)
- если в перестановке π x идет после y и z , то в описание S она не попадет

Основные идеи PPSZ-подобного алгоритма еще раз

Основные идеи еще раз

- случайно выбираем порядок переменных и расщепляем, периодически удаляя единичные клозы
- если у F есть ровно один выполняющий набор S , то для любой переменной x найдется клоз $(x \vee y \vee z)$, такой что S выполняет только литерал x в этом клозе (то есть S присваивает x, y, z значения $1, 0, 0$, соответственно)
- если в перестановке π x идет после y и z , то в описание S она не попадет
- в среднем, примерно треть всех переменных не попадет в описание

Основные идеи PPSZ-подобного алгоритма еще раз

Основные идеи еще раз

- случайно выбираем порядок переменных и расщепляем, периодически удаляя единичные клозы
- если у F есть ровно один выполняющий набор S , то для любой переменной x найдется клоз $(x \vee y \vee z)$, такой что S выполняет только литерал x в этом клозе (то есть S присваивает x, y, z значения $1, 0, 0$, соответственно)
- если в перестановке π x идет после y и z , то в описание S она не попадет
- в среднем, примерно треть всех переменных не попадет в описание
- таким образом, если мы переберем все описания длины не более $2n/3$, то с хорошей вероятностью мы найдем описание выполняющего набора

План лекции

- 1 Расщепление
 - Правила упрощения
 - Оценка времени работы
 - DPLL
 - PPSZ
- 2 Случайное блуждание

Основные идеи алгоритмов, основанных на случайных блужданиях

Основные идеи

Основные идеи алгоритмов, основанных на случайных блужданиях

Основные идеи

- выберем случайный набор переменных и проверим, выполняет ли он формулу

Основные идеи алгоритмов, основанных на случайных блужданиях

Основные идеи

- выберем случайный набор переменных и проверим, выполняет ли он формулу
- если да, то нам повезло

Основные идеи алгоритмов, основанных на случайных блужданиях

Основные идеи

- выберем случайный набор переменных и проверим, выполняет ли он формулу
- если да, то нам повезло
- если нет, то возьмем произвольный невыполненный клоз и заметим, что для какой-то переменной этого клоза ее значения в текущем и выполняющем наборе различаются (так как выполняющий набор выполняет этот клоз)

Основные идеи алгоритмов, основанных на случайных блужданиях

Основные идеи

- выберем случайный набор переменных и проверим, выполняет ли он формулу
- если да, то нам повезло
- если нет, то возьмем произвольный невыполненный клоз и заметим, что для какой-то переменной этого клоза ее значения в текущем и выполняющем наборе различаются (так как выполняющий набор выполняет этот клоз)
- выберем случайную переменную из этого клоза и изменим ее значение в текущем наборе

Основные идеи алгоритмов, основанных на случайных блужданиях

Основные идеи

- выберем случайный набор переменных и проверим, выполняет ли он формулу
- если да, то нам повезло
- если нет, то возьмем произвольный невыполненный клоз и заметим, что для какой-то переменной этого клоза ее значения в текущем и выполняющем наборе различаются (так как выполняющий набор выполняет этот клоз)
- выберем случайную переменную из этого клоза и изменим ее значение в текущем наборе
- с некоторой вероятностью мы стали ближе к выполняющему набору

Общая схема

Алгоритм

RANDOM-WALK(F, α, τ)

Общая схема

Алгоритм

RANDOM-WALK(F, α, τ)

- Повторить $\alpha(n)$ раз:

Общая схема

Алгоритм

RANDOM-WALK(F, α, τ)

- Повторить $\alpha(n)$ раз:
 - ▶ Выбрать случайным образом набор A .

Алгоритм

RANDOM-WALK(F, α, τ)

- Повторить $\alpha(n)$ раз:
 - ▶ Выбрать случайным образом набор A .
 - ▶ Если A выполняет F , выдать его. В противном случае повторить $\tau(n)$ раз:

Алгоритм

RANDOM-WALK(F, α, τ)

- Повторить $\alpha(n)$ раз:
 - ▶ Выбрать случайным образом набор A .
 - ▶ Если A выполняет F , выдать его. В противном случае повторить $\tau(n)$ раз:
 - ★ Взять произвольный невыполненный клон C в F .

Алгоритм

RANDOM-WALK(F, α, τ)

- Повторить $\alpha(n)$ раз:
 - ▶ Выбрать случайным образом набор A .
 - ▶ Если A выполняет F , выдать его. В противном случае повторить $\tau(n)$ раз:
 - ★ Взять произвольный невыполненный кюз C в F .
 - ★ Выбрать случайную переменную x из C .

Алгоритм

RANDOM-WALK(F, α, τ)

- Повторить $\alpha(n)$ раз:
 - ▶ Выбрать случайным образом набор A .
 - ▶ Если A выполняет F , выдать его. В противном случае повторить $\tau(n)$ раз:
 - ★ Взять произвольный невыполненный клоз C в F .
 - ★ Выбрать случайную переменную x из C .
 - ★ Изменить значение x в A .

Алгоритм

RANDOM-WALK(F, α, τ)

- Повторить $\alpha(n)$ раз:
 - ▶ Выбрать случайным образом набор A .
 - ▶ Если A выполняет F , выдать его. В противном случае повторить $\tau(n)$ раз:
 - ★ Взять произвольный невыполненный клоз C в F .
 - ★ Выбрать случайную переменную x из C .
 - ★ Изменить значение x в A .
 - ★ Если A выполняет F , выдать его.

Алгоритм

RANDOM-WALK(F, α, τ)

- Повторить $\alpha(n)$ раз:
 - ▶ Выбрать случайным образом набор A .
 - ▶ Если A выполняет F , выдать его. В противном случае повторить $\tau(n)$ раз:
 - ★ Взять произвольный невыполненный кюз C в F .
 - ★ Выбрать случайную переменную x из C .
 - ★ Изменить значение x в A .
 - ★ Если A выполняет F , выдать его.
- Выдать ответ “невыполнима”.

Анализ алгоритма

Анализ алгоритма

Анализ алгоритма

Анализ алгоритма

- рассмотрим работу алгоритма на формуле F в k -КНФ

Анализ алгоритма

Анализ алгоритма

- рассмотрим работу алгоритма на формуле F в k -КНФ
- предположим, что у F есть выполняющий набор S , который отличается от A значениями ровно i переменных

Анализ алгоритма

Анализ алгоритма

- рассмотрим работу алгоритма на формуле F в k -КНФ
- предположим, что у F есть выполняющий набор S , который отличается от A значениями ровно i переменных
- на каждом шаге алгоритм приближается к S с вероятностью хотя бы $1/k$

Анализ алгоритма

Анализ алгоритма

- рассмотрим работу алгоритма на формуле F в k -КНФ
- предположим, что у F есть выполняющий набор S , который отличается от A значениями ровно i переменных
- на каждом шаге алгоритм приближается к S с вероятностью хотя бы $1/k$
- таким образом, процесс модификации A связан со следующим одномерным случайным блужданием:

Анализ алгоритма

Анализ алгоритма

- рассмотрим работу алгоритма на формуле F в k -КНФ
- предположим, что у F есть выполняющий набор S , который отличается от A значениями ровно i переменных
- на каждом шаге алгоритм приближается к S с вероятностью хотя бы $1/k$
- таким образом, процесс модификации A связан со следующим одномерным случайным блужданием:
 - ▶ частица блуждает на интервале $[0..n]$

Анализ алгоритма

Анализ алгоритма

- рассмотрим работу алгоритма на формуле F в k -КНФ
- предположим, что у F есть выполняющий набор S , который отличается от A значениями ровно i переменных
- на каждом шаге алгоритм приближается к S с вероятностью хотя бы $1/k$
- таким образом, процесс модификации A связан со следующим одномерным случайным блужданием:
 - ▶ частица блуждает на интервале $[0..n]$
 - ▶ стартует с позиции j в момент времени $t = 0$ и совершает $\tau(n)$ шагов

Анализ алгоритма

Анализ алгоритма

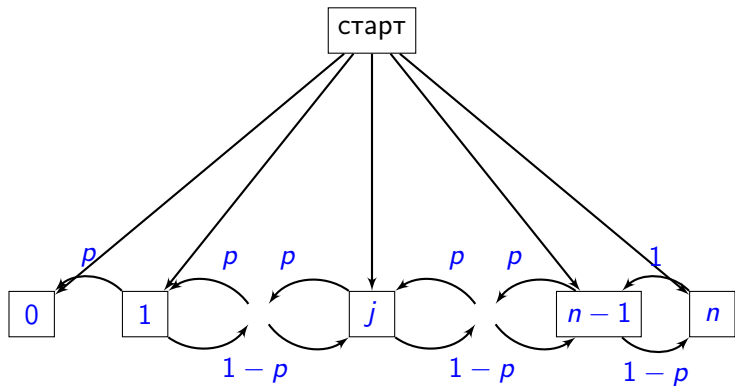
- рассмотрим работу алгоритма на формуле F в k -КНФ
- предположим, что у F есть выполняющий набор S , который отличается от A значениями ровно i переменных
- на каждом шаге алгоритм приближается к S с вероятностью хотя бы $1/k$
- таким образом, процесс модификации A связан со следующим одномерным случайным блужданием:
 - ▶ частица блуждает на интервале $[0..n]$
 - ▶ стартует с позиции j в момент времени $t = 0$ и совершает $\tau(n)$ шагов
 - ▶ с вероятностью $1/k$ переходит в $j - 1$, с вероятностью $1 - 1/k$ — в $j + 1$

Анализ алгоритма

Анализ алгоритма

- рассмотрим работу алгоритма на формуле F в k -КНФ
- предположим, что у F есть выполняющий набор S , который отличается от A значениями ровно i переменных
- на каждом шаге алгоритм приближается к S с вероятностью хотя бы $1/k$
- таким образом, процесс модификации A связан со следующим одномерным случайным блужданием:
 - ▶ частица блуждает на интервале $[0..n]$
 - ▶ стартует с позиции j в момент времени $t = 0$ и совершает $\tau(n)$ шагов
 - ▶ с вероятностью $1/k$ переходит в $j - 1$, с вероятностью $1 - 1/k$ — в $j + 1$
 - ▶ в 0 останавливается, из n обязательно переходит в $n - 1$

Случайные блуждания визуально



Анализ алгоритма неформально

Неформально

Анализ алгоритма неформально

Неформально

- для оценки вероятности ошибки такого алгоритма оценивается вероятность того, что частица, вышедшая из состояния i , достигнет состояния 0 за t шагов

Анализ алгоритма неформально

Неформально

- для оценки вероятности ошибки такого алгоритма оценивается вероятность того, что частица, вышедшая из состояния i , достигнет состояния 0 за t шагов
- мы рассмотрим упрощенные варианты такого алгоритма, в которых вместо выбора случайной переменной из невыполненного клоза, рассматриваются все три изменения

Хэммингово расстояние

Определение

Хэммингово расстояние

Определение

- **Хэммингово расстояние** двух наборов — количество переменных, которым эти наборы присваивают разные значения.

Хэммингово расстояние

Определение

- **Хэммингово расстояние** двух наборов — количество переменных, которым эти наборы присваивают разные значения.
- Для набора t и числа d под **Хэмминговым шаром** $\mathcal{H}(t, d)$ (с центром в t и радиуса d) будем понимать множество всех наборов, находящихся на расстоянии не более чем d от t .

Поиск в шаре

Поиск в шаре

Для формулы F в 3-КНФ поиск выполняющего набора в шаре $\mathcal{H}(t, d)$ может быть осуществлен за время 3^d :

Поиск в шаре

Поиск в шаре

Для формулы F в 3-КНФ поиск выполняющего набора в шаре $\mathcal{H}(t, d)$ может быть осуществлен за время 3^d :

- если t не выполняет F , возьмем произвольный невыполненный клон $C = (x_1 \vee x_2 \vee x_3)$

Поиск в шаре

Поиск в шаре

Для формулы F в 3-КНФ поиск выполняющего набора в шаре $\mathcal{H}(t, d)$ может быть осуществлен за время 3^d :

- если t не выполняет F , возьмем произвольный невыполненный клз $C = (x_1 \vee x_2 \vee x_3)$
- рассмотрим три набора, полученных из t изменением значений переменных x_1 , x_2 и x_3

Поиск в шаре

Поиск в шаре

Для формулы F в 3-КНФ поиск выполняющего набора в шаре $\mathcal{H}(t, d)$ может быть осуществлен за время 3^d :

- если t не выполняет F , возьмем произвольный невыполненный клуз $C = (x_1 \vee x_2 \vee x_3)$
- рассмотрим три набора, полученных из t изменением значений переменных x_1 , x_2 и x_3
- хотя бы один из них будет ближе к выполняющему набору

$\sqrt{3}^n$ -Алгоритм

Алгоритм

SIMPLE-3-SAT(F)

$\sqrt{3}^n$ -Алгоритм

Алгоритм

SIMPLE-3-SAT(F)

- проверить, есть ли выполняющий набор в шарах $\mathcal{H}(0^n, n/2)$, $\mathcal{H}(1^n, n/2)$

$\sqrt{3}^n$ -Алгоритм

Алгоритм

SIMPLE-3-SAT(F)

- проверить, есть ли выполняющий набор в шарах $\mathcal{H}(0^n, n/2)$, $\mathcal{H}(1^n, n/2)$

Лемма

Время работы алгоритма SIMPLE-3-SAT есть $\sqrt{3}^n$, где $n = n(F)$ — число переменных формулы F .

$\sqrt{3}^n$ -Алгоритм

Алгоритм

SIMPLE-3-SAT(F)

- проверить, есть ли выполняющий набор в шарах $\mathcal{H}(0^n, n/2)$, $\mathcal{H}(1^n, n/2)$

Лемма

Время работы алгоритма SIMPLE-3-SAT есть $\sqrt{3}^n$, где $n = n(F)$ — число переменных формулы F .

Доказательство

Понятно, что если выполняющий набор есть, то он содержится в одном из рассмотренных двух шаров. □

1.5^n -Алгоритм

Алгоритм

IMPROVED-3-SAT(F)

1.5ⁿ-Алгоритм

Алгоритм

IMPROVED-3-SAT(F)

- повторить $T = c \cdot 2^n / |\mathcal{H}(0^n, n/4)|$ раз:

1.5ⁿ-Алгоритм

Алгоритм

IMPROVED-3-SAT(F)

- повторить $T = c \cdot 2^n / |\mathcal{H}(0^n, n/4)|$ раз:
 - ▶ выбрать случайно набор t

1.5ⁿ-Алгоритм

Алгоритм

IMPROVED-3-SAT(F)

- повторить $T = c \cdot 2^n / |\mathcal{H}(0^n, n/4)|$ раз:
 - ▶ выбрать случайно набор t
 - ▶ проверить шар $\mathcal{H}(t, n/4)$

1.5ⁿ-Алгоритм

Алгоритм

IMPROVED-3-SAT(F)

- повторить $T = c \cdot 2^n / |\mathcal{H}(0^n, n/4)|$ раз:
 - ▶ выбрать случайно набор t
 - ▶ проверить шар $\mathcal{H}(t, n/4)$
 - ▶ если выполняющий набор найден, выдать ответ “выполнима”

1.5ⁿ-Алгоритм

Алгоритм

IMPROVED-3-SAT(F)

- повторить $T = c \cdot 2^n / |\mathcal{H}(0^n, n/4)|$ раз:
 - ▶ выбрать случайно набор t
 - ▶ проверить шар $\mathcal{H}(t, n/4)$
 - ▶ если выполняющий набор найден, выдать ответ “выполнима”
- выдать ответ “невыполнима”

1.5ⁿ-Алгоритм

Алгоритм

IMPROVED-3-SAT(F)

- повторить $T = c \cdot 2^n / |\mathcal{H}(0^n, n/4)|$ раз:
 - ▶ выбрать случайно набор t
 - ▶ проверить шар $\mathcal{H}(t, n/4)$
 - ▶ если выполняющий набор найден, выдать ответ “выполнима”
- выдать ответ “невыполнима”

Лемма

Алгоритм IMPROVED-3-SAT имеет время работы 1.5^n и вероятность ошибки не более e^{-c} .

Доказательство

Доказательство

Доказательство

Доказательство

- время работы: $T \cdot 3^{n/4} = c \cdot 2^n \cdot 3^{n/4} / |\mathcal{H}(0^n, n/4)| \leq 1.5^n$

Доказательство

Доказательство

- время работы: $T \cdot 3^{n/4} = c \cdot 2^n \cdot 3^{n/4} / |\mathcal{H}(0^n, n/4)| \leq 1.5^n$
- вероятность ошибки:

Доказательство

Доказательство

- время работы: $T \cdot 3^{n/4} = c \cdot 2^n \cdot 3^{n/4} / |\mathcal{H}(0^n, n/4)| \leq 1.5^n$
- вероятность ошибки:
 - ▶ ошибка может возникнуть только в случае, когда формула выполнима и алгоритм всегда выбирает “плохие центры”

Доказательство

Доказательство

- время работы: $T \cdot 3^{n/4} = c \cdot 2^n \cdot 3^{n/4} / |\mathcal{H}(0^n, n/4)| \leq 1.5^n$
- вероятность ошибки:
 - ▶ ошибка может возникнуть только в случае, когда формула выполнима и алгоритм всегда выбирает “плохие центры”
 - ▶ вероятность того, что случайно выбранный набор является плохим, не превосходит $1 - |\mathcal{H}(0^n, 2^{n/4})|/2^n = 1 - c/T$

Доказательство

Доказательство

- время работы: $T \cdot 3^{n/4} = c \cdot 2^n \cdot 3^{n/4} / |\mathcal{H}(0^n, n/4)| \leq 1.5^n$
- вероятность ошибки:
 - ▶ ошибка может возникнуть только в случае, когда формула выполнима и алгоритм всегда выбирает “плохие центры”
 - ▶ вероятность того, что случайно выбранный набор является плохим, не превосходит $1 - |\mathcal{H}(0^n, 2^{n/4})|/2^n = 1 - c/T$
 - ▶ таким образом, вероятность ошибки не превосходит $(1 - c/T)^T < e^{-c}$



Обобщение для k -SAT

Обобщение для k -SAT

Обобщение для k -SAT

Обобщение для k -SAT

- покрыть множество всех наборов шарами равного радиуса (покрывающий код, covering code)

Обобщение для k -SAT

Обобщение для k -SAT

- покрыть множество всех наборов шарами равного радиуса (покрывающий код, covering code)
- проверить каждый шар

Обобщение для k -SAT

Обобщение для k -SAT

- покрыть множество всех наборов шарами равного радиуса (покрывающий код, covering code)
- проверить каждый шар
- время работы алгоритма: $(2 - 2/(k + 1))^n$

Что мы узнали за сегодня?

Что мы узнали за сегодня?

основные типы алгоритмов для SAT:

Что мы узнали за сегодня?

Что мы узнали за сегодня?

основные типы алгоритмов для SAT:

- расщепляющие алгоритмы сводят задачу для входной формулы F к задаче для более простых формул F_1, \dots, F_j

Что мы узнали за сегодня?

Что мы узнали за сегодня?

основные типы алгоритмов для SAT:

- расщепляющие алгоритмы сводят задачу для входной формулы F к задаче для более простых формул F_1, \dots, F_j
 - ▶ DPLL-подобные алгоритмы заменяют входную формулу F на две формулы $F[x = 1]$ и $F[x = 0]$; время работы оценивается из рекуррентных соотношений для количества листьев в дереве рекурсии

Что мы узнали за сегодня?

Что мы узнали за сегодня?

основные типы алгоритмов для SAT:

- расщепляющие алгоритмы сводят задачу для входной формулы F к задаче для более простых формул F_1, \dots, F_j
 - ▶ DPLL-подобные алгоритмы заменяют входную формулу F на две формулы $F[x = 1]$ и $F[x = 0]$; время работы оценивается из рекуррентных соотношений для количества листьев в дереве рекурсии
 - ▶ PPSZ-подобные алгоритмы присваивают значения переменным случайно; время работы оценивается “глобально”

Что мы узнали за сегодня?

Что мы узнали за сегодня?

основные типы алгоритмов для SAT:

- расщепляющие алгоритмы сводят задачу для входной формулы F к задаче для более простых формул F_1, \dots, F_j
 - ▶ DPLL-подобные алгоритмы заменяют входную формулу F на две формулы $F[x = 1]$ и $F[x = 0]$; время работы оценивается из рекуррентных соотношений для количества листьев в дереве рекурсии
 - ▶ PPSZ-подобные алгоритмы присваивают значения переменным случайно; время работы оценивается “глобально”
- алгоритмы, основанные на локальном поиске берут начальный набор и изменяют его шаг за шагом, пытаясь приблизиться к выполняющему набору; если через некоторое время выполняющий набор так и не был найден, порождается другой начальный набор

Спасибо за внимание!