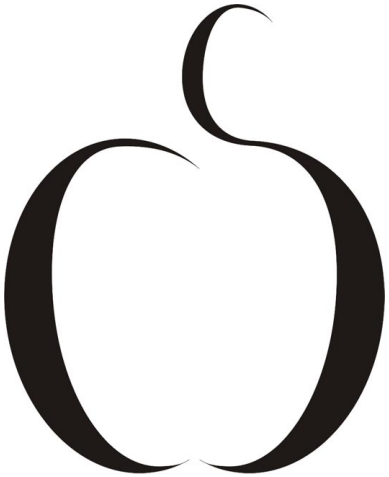
с/к “Эффективные алгоритмы”

Лекция 13: Подходы к решению NP-трудных задач

А. Куликов

Computer Science клуб при ПОМИ
http://logic.pdmi.ras.ru/∼infclub/



План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость
7. Случайное сведение к простой задаче
   - 3-раскрашиваемость

Мотивация

Представим, что у нас есть алгоритм сложности 1.7^n для некоторой задачи, который за "разумное" время позволяет решать примеры этой задачи размера не более n_0.

Мотивация

Представим, что у нас есть алгоритм сложности 1.7^n для некоторой задачи, который за "разумное" время позволяет решать примеры этой задачи размера не более n_0.

- The "hardware" approach: возьмем в 10 раз более быстрый компьютер. Теперь мы можем решать примеры размера $n_0 + 4$.

Мотивация

Представим, что у нас есть алгоритм сложности 1.7^n для некоторой задачи, который за "разумное" время позволяет решать примеры этой задачи размера не более n_0.

- The "hardware" approach: возьмем в 10 раз более быстрый компьютер. Теперь мы можем решать примеры размера $n_0 + 4$.
- The "brainware" approach: придумаем алгоритм сложности 1.3^n. Это позволит нам решать примеры размера $2 \cdot n_0$.

Мотивация

Представим, что у нас есть алгоритм сложности 1.7^n для некоторой задачи, который за "разумное" время позволяет решать примеры этой задачи размера не более n_0.

- The "hardware" approach: возьмем в 10 раз более быстрый компьютер. Теперь мы можем решать примеры размера $n_0 + 4$.
- The "brainware" approach: придумаем алгоритм сложности 1.3^n. Это позволит нам решать примеры размера $2 \cdot n_0$.

Другими словами, уменьшение основания экспоненты времени работы алгоритма увеличивает размер решаемых за данное время примеров в константное число раз, в то время как использование более быстрого компьютера способно увеличить размер лишь на константу.

Мотивация

Мотивация

- Многим приложениям требуется решать NP-трудные задачи, даже несмотря на то, что решения могут быть найдены только для весьма маленьких размеров входов.

Мотивация

- Многим приложениям требуется решать NP-трудные задачи, даже несмотря на то, что решения могут быть найдены только для весьма маленьких размеров входов.
- Лучшее понимание NP-трудных задач.

Мотивация

- Многим приложениям требуется решать NP-трудные задачи, даже несмотря на то, что решения могут быть найдены только для весьма маленьких размеров входов.
- Лучшее понимание NP-трудных задач.
- Новые интересные комбинаторные и алгоритмические задачи.

Мотивация

- Многим приложениям требуется решать NP-трудные задачи, даже несмотря на то, что решения могут быть найдены только для весьма маленьких размеров входов.
- Лучшее понимание NP-трудных задач.
- Новые интересные комбинаторные и алгоритмические задачи.
- Общая теория.

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость
7. Случайное сведение к простой задаче
   - 3-раскрашиваемость

Метод расщепления

Основная идея

Разбить входной пример задачи на несколько более простых примеров той же задачи, найти для них решения (рекурсивными вызовами) и построить по найденным решениям ответ для исходного примера.

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость
7. Случайное сведение к простой задаче
   - 3-раскрашиваемость

Анализ алгоритмов расщепления

Анализ алгоритмов расщепления

- как правило, анализ расщепляющего алгоритма состоит из длинного списка случаев

Анализ алгоритмов расщепления

- как правило, анализ расщепляющего алгоритма состоит из длинного списка случаев
- в каждом случае показывается, что входной пример можно разбить на несколько примеров, чьи размеры на нужную константу меньше, чем размер исходного примера

Анализ алгоритмов расщепления

- как правило, анализ расщепляющего алгоритма состоит из длинного списка случаев
- в каждом случае показывается, что входной пример можно разбить на несколько примеров, чьи размеры на нужную константу меньше, чем размер исходного примера
- доказательство для каждого случая представаляет собой простое комбинаторное рассуждение

Анализ алгоритмов расщепления

- как правило, анализ расщепляющего алгоритма состоит из длинного списка случаев
- в каждом случае показывается, что входной пример можно разбить на несколько примеров, чьи размеры на нужную константу меньше, чем размер исходного примера
- доказательство для каждого случая представаляет собой простое комбинаторное рассуждение
- такой разбор случаев можно проводить автоматически

Программа для автоматического доказательства верхних оценок для NP-трудных задач

Вход

Программа для автоматического доказательства верхних оценок для NP-трудных задач

Вход

- NP-трудную задачу, сформулированную в терминах КНФ формул

Программа для автоматического доказательства верхних оценок для NP-трудных задач

Вход

- NP-трудную задачу, сформулированную в терминах КНФ формул
- множество правил упрощения для данной задачи

Программа для автоматического доказательства верхних оценок для NP-трудных задач

Вход

- NP-трудную задачу, сформулированную в терминах КНФ формул
- множество правил упрощения для данной задачи
- верхнюю оценку

Программа для автоматического доказательства верхних оценок для NP-трудных задач

Вход

- NP-трудную задачу, сформулированную в терминах КНФ формул
- множество правил упрощения для данной задачи
- верхнюю оценку

По входным данным программа пытается найти

алгоритм расщепления для данной задачи, который использует данные правила упрощения и имеет время работы, удовлетворяющее данной оценке.

Пример входа

Пример входа

Допсутим, программа получила задание доказать, что существует алгоритм для выполнимости, который использует правила удаления единичных клозов и чистых литералов и имеет время работы не хуже 1.619^K, где K — количество клозов входной формулы.

Предобработка

Предобработка

Предобработка

Предобработка

- программа должна доказать, что для любой упрощенной формулы можно найти $(1, 2)$-расщепление (то есть расщепление, которому соответствует рекуррентное неравенство $T(K) \leq T(K-1) + T(K-2) + poly(K)$)

Предобработка

Предобработка

- программа должна доказать, что для любой упрощенной формулы можно найти $(1,2)$-расщепление (то есть расщепление, которому соответствует рекуррентное неравенство $T(K) \leq T(K-1) + T(K-2) + poly(K)$)
- каждый литерал упрощенной формулы входит в нее хотя бы один раз положительно и хотя бы один раз отрицательно (все остальные литералы являются чистыми и удаляются соответствующим правилом)

Предобработка

Предобработка

- программа должна доказать, что для любой упрощенной формулы можно найти $(1,2)$-расщепление (то есть расщепление, которому соответствует рекуррентное неравенство $T(K) \leq T(K-1) + T(K-2) + poly(K)$)
- каждый литерал упрощенной формулы входит в нее хотя бы один раз положительно и хотя бы один раз отрицательно (все остальные литералы являются чистыми и удаляются соответствующим правилом)
- более того, если есть литерал, который входит в формулу хотя бы дважды, то расщепление по нему дает требуемое неравенство:

$$\boxed{(x \vee a \vee \ldots) \wedge (x \vee b \vee \ldots) \wedge (\bar{x} \vee c \vee \ldots) \wedge \ldots}$$

$x = 1$ $\qquad$ $x = 0$

$$\boxed{(c \vee \ldots) \wedge \ldots} \qquad \boxed{(a \vee \ldots) \wedge (b \vee \ldots) \wedge \ldots}$$

Предобработка

Таким образом

Программе нужно доказать, что упрощенную формулу, состоящую только из $(1, 1)$-литералов, всегда можно хорошо расщепить.

Автоматический разбор случаев

упрощенные формулы

Автоматический разбор случаев

упрощенные формулы

$$\downarrow$$

$$(x \vee \ldots) \wedge (\bar{x} \vee \ldots) \wedge \ldots$$

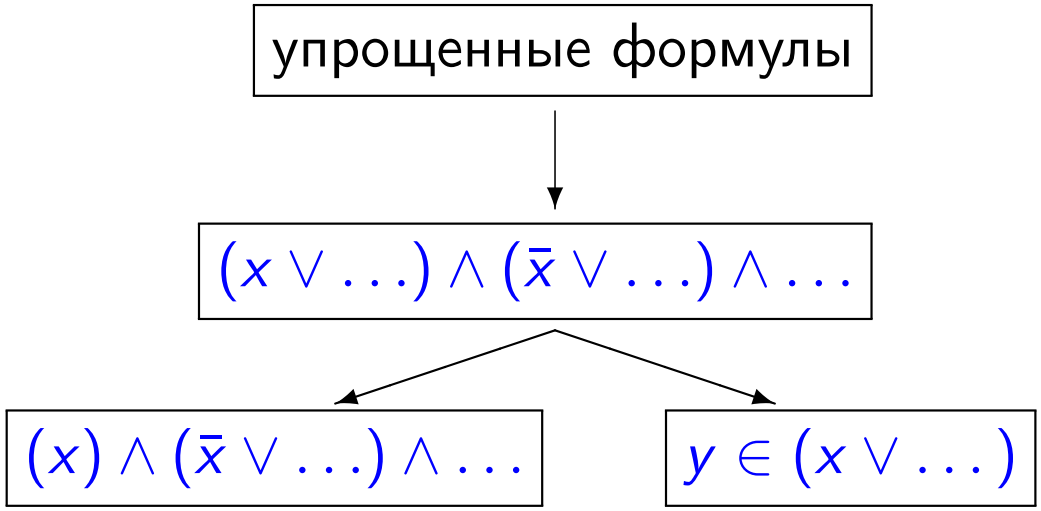
Автоматический разбор случаев

упрощенные формулы

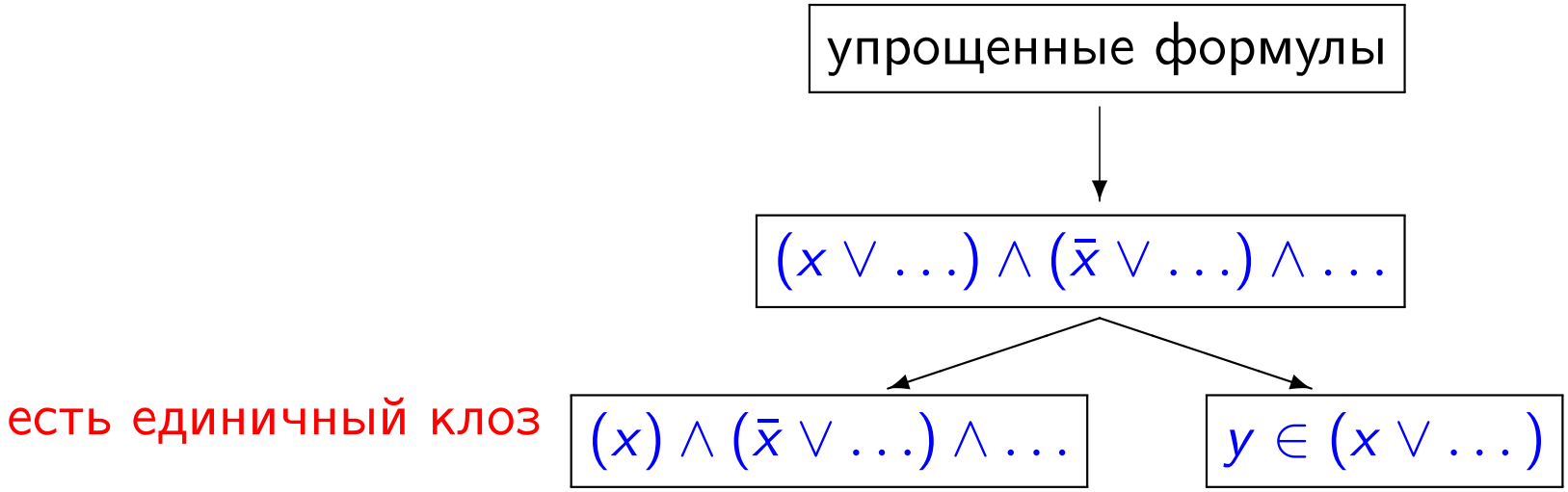
$$(x \vee \ldots) \wedge (\bar{x} \vee \ldots) \wedge \ldots$$

$$(x) \wedge (\bar{x} \vee \ldots) \wedge \ldots$$

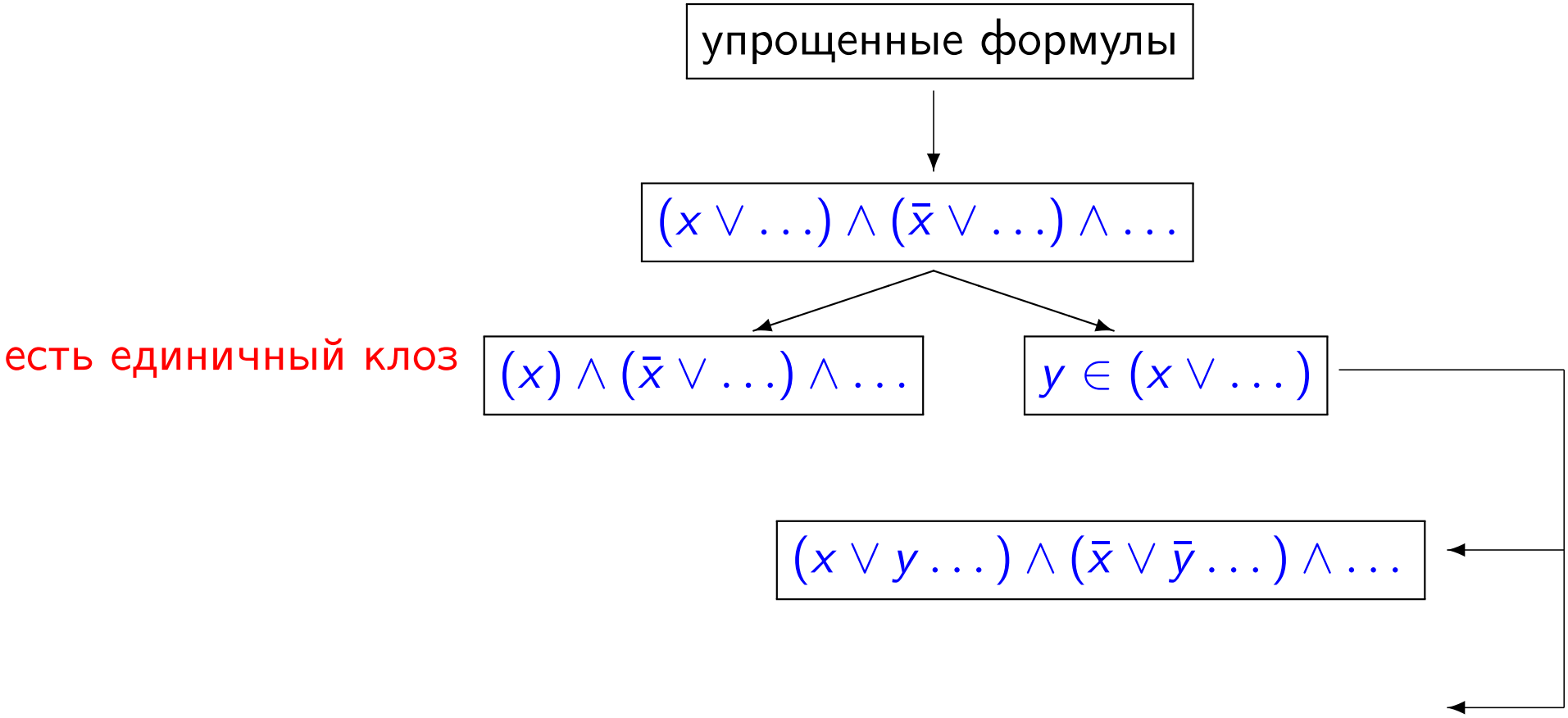
Автоматический разбор случаев



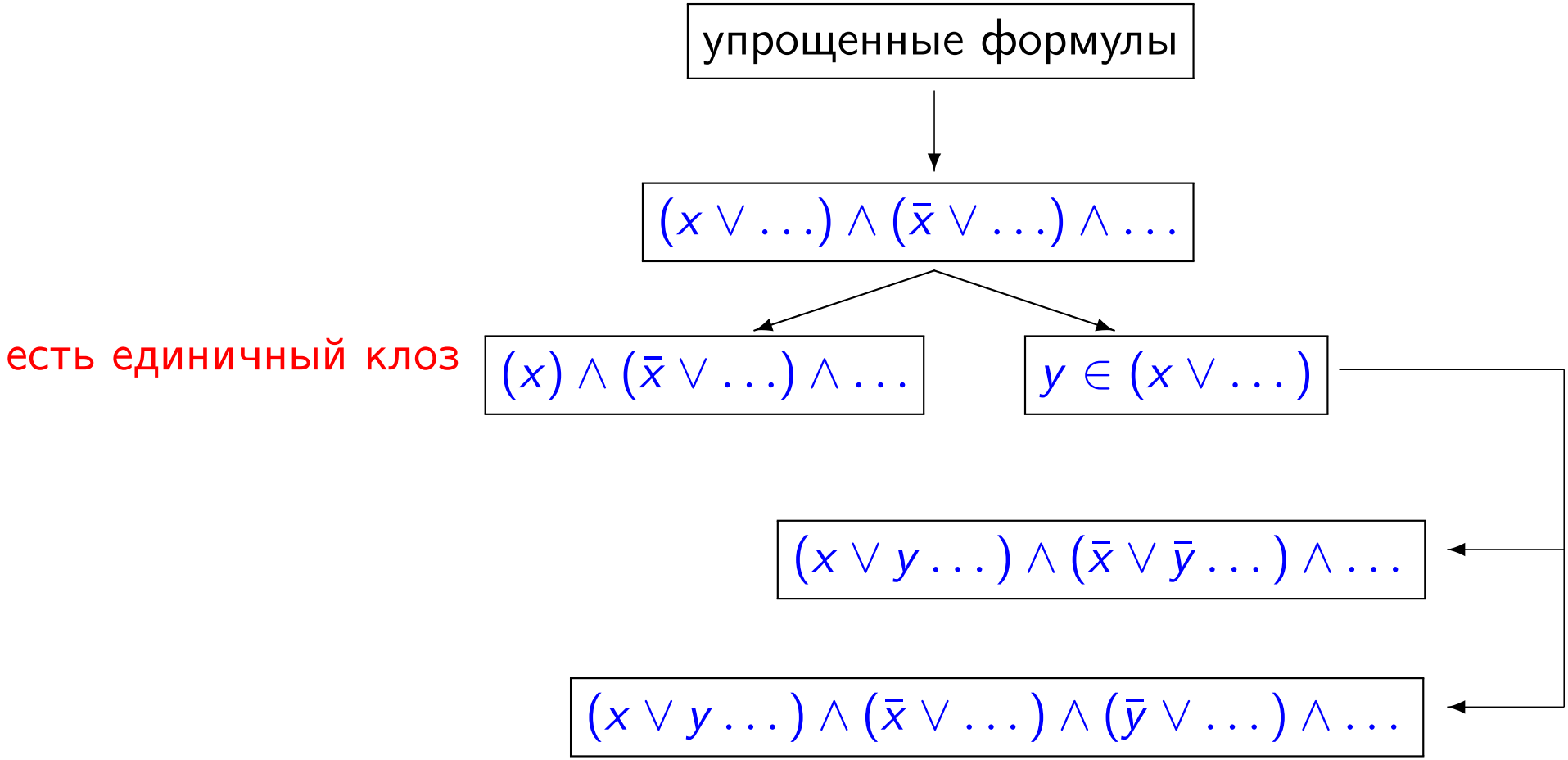
Автоматический разбор случаев



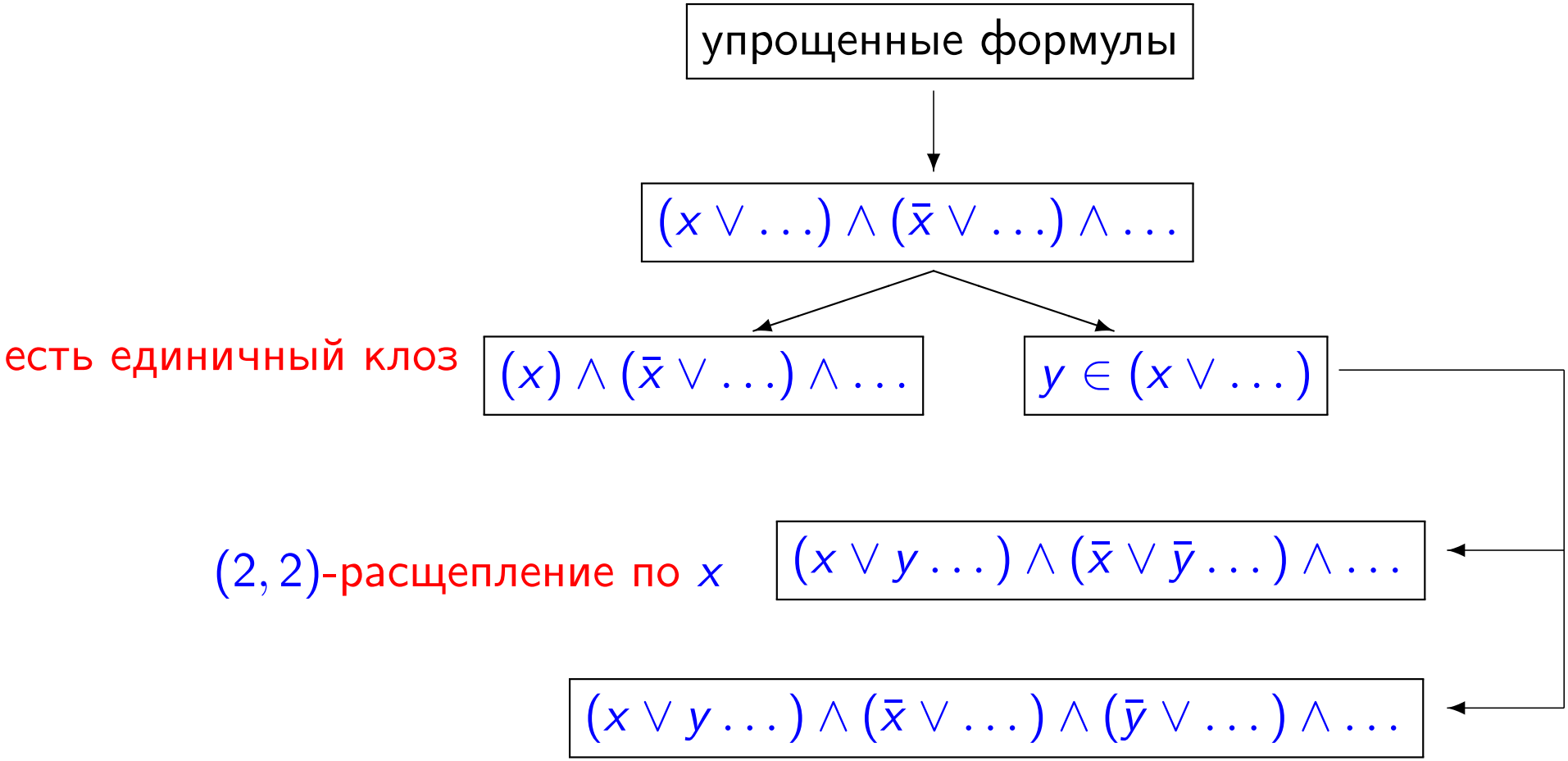
Автоматический разбор случаев



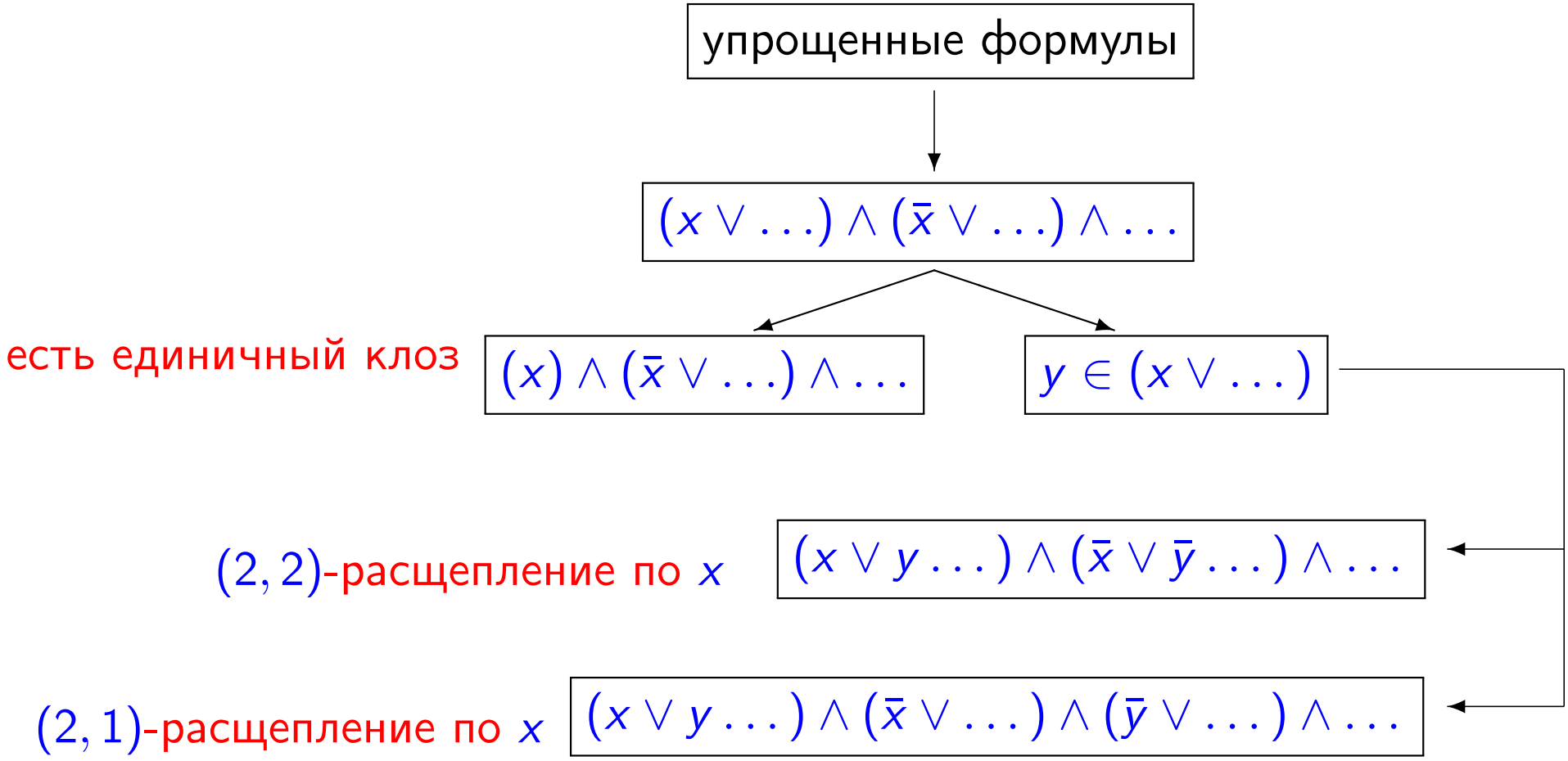
Автоматический разбор случаев



Автоматический разбор случаев



Автоматический разбор случаев



План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость
7. Случайное сведение к простой задаче
   - 3-раскрашиваемость

Случайный порядок перебора

Основная идея

Если для нахождения решения достаточно знать какую-то его часть, можно попытаться ее угадать, а потом достроить решение.

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость
7. Случайное сведение к простой задаче
   - 3-раскрашиваемость

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 $\{1, \ldots, n\}$

PPSZ-подобный алгоритм

Алгоритм

PPSZ-SAT(F)

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

PPSZ-подобный алгоритм

Алгоритм

PPSZ-SAT(F)

- выбрать случайным образом перестановку π из множества всех перестановок $\{1, \ldots, n\}$
- 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 входяющую в формулу, и расщепляемся по ней

PPSZ-подобный алгоритм

Алгоритм

PPSZ-SAT(F)

- выбрать случайным образом перестановку π из множества всех перестановок $\{1, \ldots, n\}$
- 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 входяющую 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 $\{1, \ldots, n\}$
- 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 входяющую в формулу, и расщепляемся по ней
- если на каком-то шаге выяснилось, что формула выполнима, выдать “выполнима”
- в противном случае выдать “невыполнима”

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость
7. Случайное сведение к простой задаче
   - 3-раскрашиваемость

Локальный поиск

Основная идея

Взять кандидата на решение и проверить, является ли он решением. Если нет, то локально модифицировать. Если через несколько итераций решение не найдено, выбрать другого кандидата.

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость
7. Случайное сведение к простой задаче
   - 3-раскрашиваемость

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 радуиса 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 выполянет F, возьмем произвольный невыполненный клоз $C = (x_1 \vee x_2 \vee x_3)$

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 выполянет F, возьмем произвольный невыполненный клоз $C = (x_1 \vee x_2 \vee x_3)$
- 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 выполянет F, возьмем произвольный невыполненный клоз $C = (x_1 \vee x_2 \vee x_3)$
- 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

$\textsc{Simple-3-SAT}(F)$

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

Лемма

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

$\sqrt{3}^n$-Алгоритм

Алгоритм

$\textsc{Simple-3-SAT}(F)$

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

Лемма

Время работы алгоритма $\textsc{Simple-3-SAT}$ 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.

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость
7. Случайное сведение к простой задаче
   - 3-раскрашиваемость

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: $(2 - 2/(k+1))^n$

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: $(2 - 2/(k+1))^n$

Открытый вопрос

Придумать алгоритм, работающий быстрее.

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость
7. Случайное сведение к простой задаче
   - 3-раскрашиваемость

Динамическое программирование

Основная идея

Последовательно находить и запоминать решения для подпримеров исходного примера.

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость
7. Случайное сведение к простой задаче
   - 3-раскрашиваемость

Задача о коммивояжере

Алгоритм

DYNAMIC-TSP(G)

Задача о коммивояжере

Алгоритм

DYNAMIC-TSP(G)

- для каждого непустого подмножества $S \subseteq \{2, \ldots, n\}$ и для каждой вершины $i \in S$ через $Opt[S, i]$ будем обозначать длину кратчайшего маршрута. начнинающегося в вершине 1, проходящему через все вершины $S - \{i\}$ и заканчивающегося в вершине i

Задача о коммивояжере

Алгоритм

DYNAMIC-TSP(G)

- для каждого непустого подмножества $S \subseteq \{2, \ldots, n\}$ и для каждой вершины $i \in S$ через $Opt[S, i]$ будем обозначать длину кратчайшего маршрута. начнинающегося в вершине 1, проходящему через все вершины $S - \{i\}$ и заканчивающегося в вершине i
- последовательно заполнить матрицу: $Opt[S, i] = \min\{Opt[S - \{i\}, j] + d(i, j) : j \in S - \{i\}\}$

Задача о коммивояжере

Алгоритм

DYNAMIC-TSP(G)

- для каждого непустого подмножества $S \subseteq \{2, \ldots, n\}$ и для каждой вершины $i \in S$ через $Opt[S, i]$ будем обозначать длину кратчайшего маршрута. начнинающегося в вершине 1, проходящему через все вершины $S - \{i\}$ и заканчивающегося в вершине i
- последовательно заполнить матрицу: $Opt[S, i] = \min\{Opt[S - \{i\}, j] + d(i, j) : j \in S - \{i\}\}$
- вернуть оптимальную стоимость маршрута: $\min\{Opt[\{2, \ldots, n\}, j] + d(j, 1) : 2 \leq j \leq n\}$

Открытые вопросы

Лемма

Сложность алгоритма DYNAMIC-TSP по времени и по памяти есть $\mathrm{poly}(n)2^n$.

Открытые вопросы

Лемма

Сложность алгоритма Dynamic-TSP по времени и по памяти есть $\mathrm{poly}(n)2^n$.

Факт

Данный теоретический алгоритм был представлен в 1962-м году и до сих пор является лучшим из известных.

Открытые вопросы

Лемма

Сложность алгоритма Dynamic-TSP по времени и по памяти есть $\mathrm{poly}(n)2^n$.

Факт

Данный теоретический алгоритм был представлен в 1962-м году и до сих пор является лучшим из известных.

Открытые вопросы

Открытые вопросы

Лемма

Сложность алгоритма Dynamic-TSP по времени и по памяти есть $\mathrm{poly}(n)2^n$.

Факт

Данный теоретический алгоритм был представлен в 1962-м году и до сих пор является лучшим из известных.

Открытые вопросы

- Придумать алгоритм, работающий за время 1.99^n.

Открытые вопросы

Лемма

Сложность алгоритма Dynamic-TSP по времени и по памяти есть $\mathrm{poly}(n)2^n$.

Факт

Данный теоретический алгоритм был представлен в 1962-м году и до сих пор является лучшим из известных.

Открытые вопросы

- Придумать алгоритм, работающий за время 1.99^n.
- Придумать алгоритм, работающий за время 2^n, но полиномиальный по памяти.

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость
7. Случайное сведение к простой задаче
   - 3-раскрашиваемость

Сведение к простой задаче

Основная идея

По входному примеру трудной задачи построить пример экспонециального размера для простой задачи и воспользоваться эффективным алгоритмом для этой задачи.

Табличная сумма

Определение

Задача о табличной k-сумме (table-k-SUM problem) заключается в проверке, можно ли из каждой строчки входной матрицы размера $k \times m$ выбрать по числу так, чтобы их сумма равнялась входному результату S.

Табличная сумма

Определение

Задача о табличной k-сумме (table-k-SUM problem) заключается в проверке, можно ли из каждой строчки входной матрицы размера $k \times m$ выбрать по числу так, чтобы их сумма равнялась входному результату S.

Перебор

На полный перебор тратится время m^k.

Табличная сумма

Определение

Задача о табличной k-сумме (table-k-SUM problem) заключается в проверке, можно ли из каждой строчки входной матрицы размера $k \times m$ выбрать по числу так, чтобы их сумма равнялась входному результату S.

Перебор

На полный перебор тратится время m^k.

Улучшение для табличной 2-суммы

Отсортируем первую строчку, после чего для каждого числа x второй строчки проверим бинарным поиском, нет ли в первой строчке числа $S - x$. Все это займет $O(m \log m)$ времени.

Улучшение для табличной k-суммы

Алгоритм

Table-k-SUM-Alg($A[k, m]$, S)

Улучшение для табличной k-суммы

Алгоритм

Table-k-SUM-Alg($A[k, m]$, S)

- построить матрицу $B[2, m^{\lceil k/2 \rceil}]$ следующим образом:

Улучшение для табличной k-суммы

Алгоритм

Table-k-SUM-Alg($A[k, m]$, S)

- построить матрицу $B[2, m^{\lceil k/2 \rceil}]$ следующим образом:
  - в первую строчку записываем все возможные суммы элементов из первых $\lfloor k/2 \rfloor$ строчек матрицы A (ровно по одному из каждой строчки)

Улучшение для табличной k-суммы

Алгоритм

Table-k-SUM-Alg($A[k, m]$, S)

- построить матрицу $B[2, m^{\lceil k/2 \rceil}]$ следующим образом:
  - в первую строчку записываем все возможные суммы элементов из первых $\lfloor k/2 \rfloor$ строчек матрицы A (ровно по одному из каждой строчки)
  - во вторую — из последних $\lceil k/2 \rceil$

Улучшение для табличной k-суммы

Алгоритм

Table-k-SUM-Alg($A[k, m]$, S)

- построить матрицу $B[2, m^{\lceil k/2 \rceil}]$ следующим образом:
  - в первую строчку записываем все возможные суммы элементов из первых $\lfloor k/2 \rfloor$ строчек матрицы A (ровно по одному из каждой строчки)
  - во вторую — из последних $\lceil k/2 \rceil$
- запускаем алгоритм для табличной 2-суммы на полученной матрице B

Улучшение для табличной k-суммы

Алгоритм

Table-k-SUM-Alg($A[k, m]$, S)

- построить матрицу $B[2, m^{\lceil k/2 \rceil}]$ следующим образом:
  - в первую строчку записываем все возможные суммы элементов из первых $\lfloor k/2 \rfloor$ строчек матрицы A (ровно по одному из каждой строчки)
  - во вторую — из последних $\lceil k/2 \rceil$
- запускаем алгоритм для табличной 2-суммы на полученной матрице B

Лемма

Алгоритм Table-k-SUM-Alg имеет $O(m^{\lceil k/2 \rceil} \log m)$ сложность по времени и $O(m^{\lceil k/2 \rceil})$ сложность по памяти.

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость
7. Случайное сведение к простой задаче
   - 3-раскрашиваемость

Сумма подмножества

Определение

Задача о сумме подмножества (subset-sum problem) заключается в проверке того, можно ли из заданного набора из n чисел выбрать несколько так, чтобы их сумма равнялась заданному числу B.

Сумма подмножества

Определение

Задача о сумме подмножества (subset-sum problem) заключается в проверке того, можно ли из заданного набора из n чисел выбрать несколько так, чтобы их сумма равнялась заданному числу B.

Алгоритм

Subset-Sum-Alg($\{b_i\}_{1 \le i \le n}$, B)

Сумма подмножества

Определение

Задача о сумме подмножества (subset-sum problem) заключается в проверке того, можно ли из заданного набора из n чисел выбрать несколько так, чтобы их сумма равнялась заданному числу B.

Алгоритм

Subset-Sum-Alg($\{b_i\}_{1 \le i \le n}$, B)

- разбить входные числа на две части: $b_1, \ldots, b_{\lceil n/2 \rceil}$ и $b_{\lceil n/2 \rceil + 1}, \ldots, b_n$

Сумма подмножества

Определение

Задача о сумме подмножества (subset-sum problem) заключается в проверке того, можно ли из заданного набора из n чисел выбрать несколько так, чтобы их сумма равнялась заданному числу B.

Алгоритм

Subset-Sum-Alg($\{b_i\}_{1 \le i \le n}$, B)

- разбить входные числа на две части: $b_1, \ldots, b_{\lceil n/2 \rceil}$ и $b_{\lceil n/2 \rceil + 1}, \ldots, b_n$
- построить таблицу из двух строчек, в первую из которых записать все возможные суммы чисел из первой части, во вторую — из второй

Сумма подмножества

Определение

Задача о сумме подмножества (subset-sum problem) заключается в проверке того, можно ли из заданного набора из n чисел выбрать несколько так, чтобы их сумма равнялась заданному числу B.

Алгоритм

Subset-Sum-Alg($\{b_i\}_{1 \le i \le n}$, B)

- разбить входные числа на две части: $b_1, \ldots, b_{\lceil n/2 \rceil}$ и $b_{\lceil n/2 \rceil + 1}, \ldots, b_n$
- построить таблицу из двух строчек, в первую из которых записать все возможные суммы чисел из первой части, во вторую — из второй
- запустить для полученной матрицы алгоритм для табличной 2-суммы

Анализ алгоритма

Лемма

Алгоритм Subset-Sum-Alg имеет сложность $2^{n/2}$ по времени и по памяти.

Анализ алгоритма

Лемма

Алгоритм Subset-Sum-Alg имеет сложность $2^{n/2}$ по времени и по памяти.

Открытые вопросы

Анализ алгоритма

Лемма

Алгоритм Subset-Sum-Alg имеет сложность $2^{n/2}$ по времени и по памяти.

Открытые вопросы

- Придумать алгоритм, работающий быстрее.

Анализ алгоритма

Лемма

Алгоритм Subset-Sum-Alg имеет сложность $2^{n/2}$ по времени и по памяти.

Открытые вопросы

- Придумать алгоритм, работающий быстрее.
- Придумать алгоритм, работающий за время 1.99^n, но использующий лишь полиномиальную память.

3-клика

Определение

Задача о 3-клике (3-clique problem) заключается в проверке наличия 3-клики (то есть полного подграфа на трех вершинах) во входном графе.

3-клика

Определение

Задача о 3-клике (3-clique problem) заключается в проверке наличия 3-клики (то есть полного подграфа на трех вершинах) во входном графе.

Алгоритм

Matrix-Clique(G)

3-клика

Определение

Задача о 3-клике (3-clique problem) заключается в проверке наличия 3-клики (то есть полного подграфа на трех вершинах) во входном графе.

Алгоритм

Matrix-Clique(G)

- построить матрицу смежности A графа G

3-клика

Определение

Задача о 3-клике (3-clique problem) заключается в проверке наличия 3-клики (то есть полного подграфа на трех вершинах) во входном графе.

Алгоритм

Matrix-Clique(G)

- построить матрицу смежности A графа G
- посчитать A^3

3-клика

Определение

Задача о 3-клике (3-clique problem) заключается в проверке наличия 3-клики (то есть полного подграфа на трех вершинах) во входном графе.

Алгоритм

Matrix-Clique(G)

- построить матрицу смежности A графа G
- посчитать A^3
- вернуть "да", если на диагонали построенной матрицы есть хотя бы одна единица

3-клика

Определение

Задача о 3-клике (3-clique problem) заключается в проверке наличия 3-клики (то есть полного подграфа на трех вершинах) во входном графе.

Алгоритм

Matrix-Clique(G)

- построить матрицу смежности A графа G
- посчитать A^3
- вернуть "да", если на диагонали построенной матрицы есть хотя бы одна единица
- вернуть "нет"

Анализ алгоритма

Лемма

Алгоритм выдает правильный ответ за время $O(n^w)$, где $w \approx 2.376$ — экспонента перемножения матриц (matrix multiplication exponent).

Анализ алгоритма

Лемма

Алгоритм выдает правильный ответ за время $O(n^w)$, где $w \approx 2.376$ — экспонента перемножения матриц (matrix multiplication exponent).

Доказательство

Анализ алгоритма

Лемма

Алгоритм выдает правильный ответ за время $O(n^w)$, где $w \approx 2.376$ — экспонента перемножения матриц (matrix multiplication exponent).

Доказательство

- важное свойство матрицы смежности: $A^k[i,j]$ есть количество путей длины k из вершины i в вершину j в графе G (легко доказать по индукции)

Анализ алгоритма

Лемма

Алгоритм выдает правильный ответ за время $O(n^w)$, где $w \approx 2.376$ — экспонента перемножения матриц (matrix multiplication exponent).

Доказательство

- важное свойство матрицы смежности: $A^k[i,j]$ есть количество путей длины k из вершины i в вершину j в графе G (легко доказать по индукции)
- 3-клика — это путь длины 3 из вершину в саму себя

Анализ алгоритма

Лемма

Алгоритм выдает правильный ответ за время $O(n^w)$, где $w \approx 2.376$ — экспонента перемножения матриц (matrix multiplication exponent).

Доказательство

- важное свойство матрицы смежности: $A^k[i,j]$ есть количество путей длины k из вершины i в вершину j в графе G (легко доказать по индукции)
- 3-клика — это путь длины 3 из вершину в саму себя
- для вычисления A^3 требуется два умножения матриц

Открытые вопросы

Открытые вопросы

Открытые вопросы

Открытые вопросы

- Придумать более быстрый алгоритм для 3-клики.

Открытые вопросы

Открытые вопросы

- Придумать более быстрый алгоритм для 3-клики.
- Является ли 3-клика такой же сложной, как и умножение булевых матриц?

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость
7. Случайное сведение к простой задаче
   - 3-раскрашиваемость

Основные идеи

Основные идеи сведения задачи о максимальном разрезе к задаче 3-клики

Дан граф G на n вершинах.

Основные идеи

Основные идеи сведения задачи о максимальном разрезе к задаче 3-клики

Дан граф G на n вершинах.

- Построим трехдольный граф H на $3 \cdot 2^{n/3}$ вершинах со следующим свойством: исходный граф G имеет разрез веса w тогда и только тогда, когда H содержит 3-клику веса w.

Основные идеи

Основные идеи сведения задачи о максимальном разрезе к задаче 3-клики

Дан граф G на n вершинах.

- Построим трехдольный граф H на $3 \cdot 2^{n/3}$ вершинах со следующим свойством: исходный граф G имеет разрез веса w тогда и только тогда, когда H содержит 3-клику веса w.
- Используем быстрое умножение матриц для поиска 3-клики.

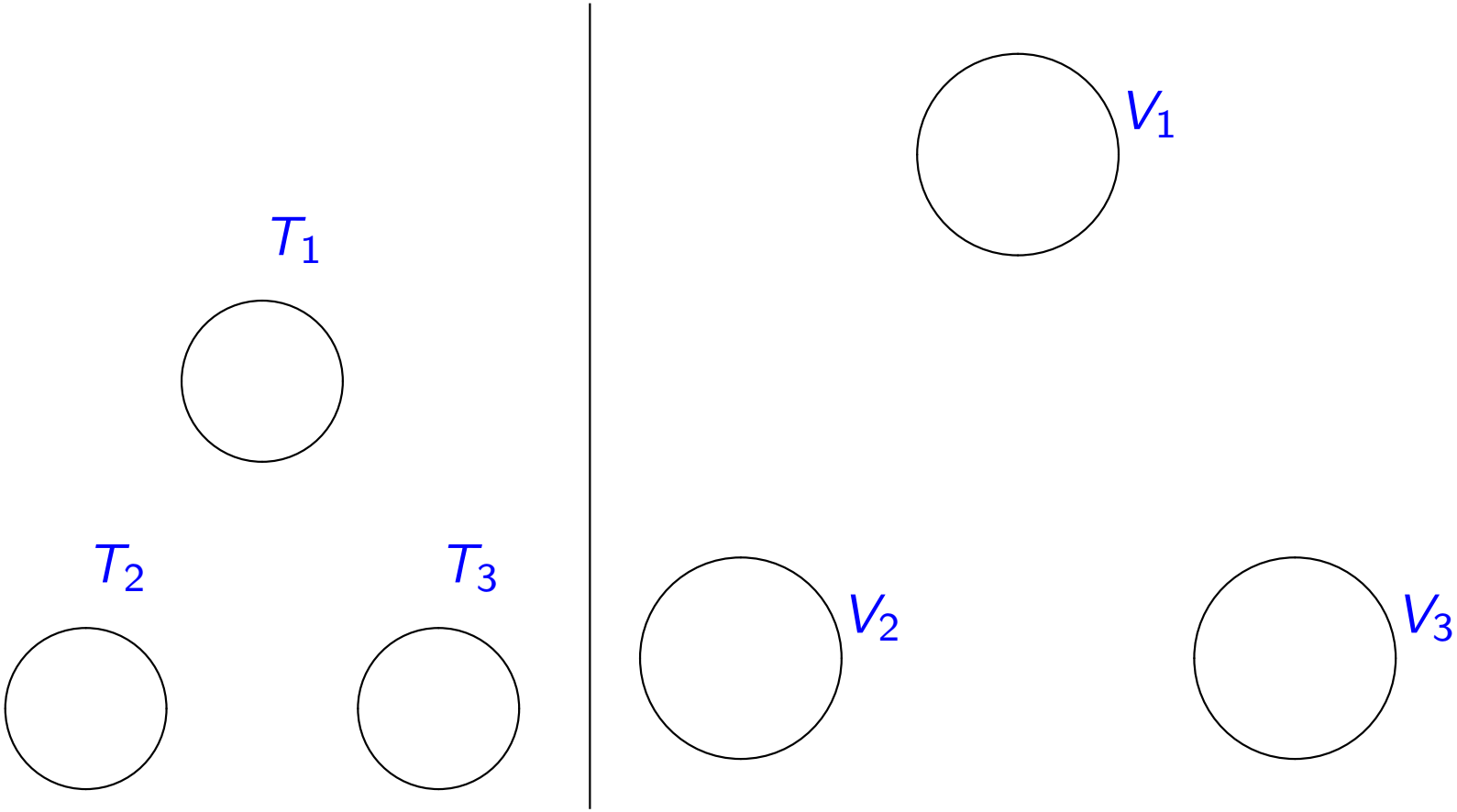
Основные идеи

Основные идеи сведения задачи о максимальном разрезе к задаче 3-клики

Дан граф G на n вершинах.

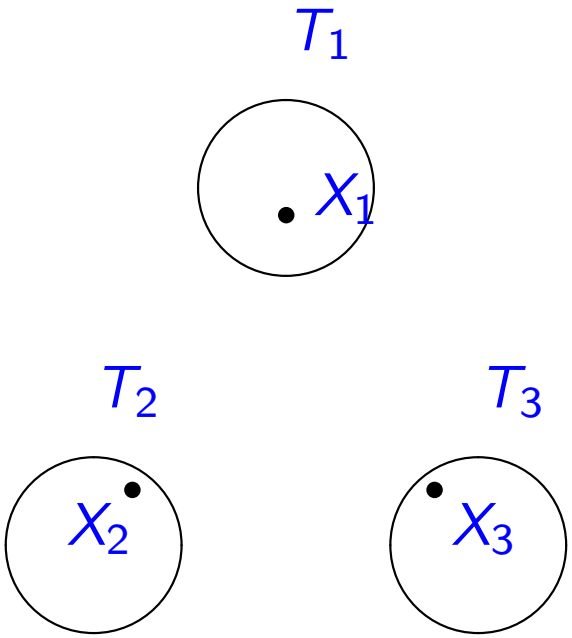
- Построим трехдольный граф H на $3 \cdot 2^{n/3}$ вершинах со следующим свойством: исходный граф G имеет разрез веса w тогда и только тогда, когда H содержит 3-клику веса w.
- Используем быстрое умножение матриц для поиска 3-клики.
- Сложность: $2^{wn/3} \approx 1.732^n$.

Вспомогательный граф

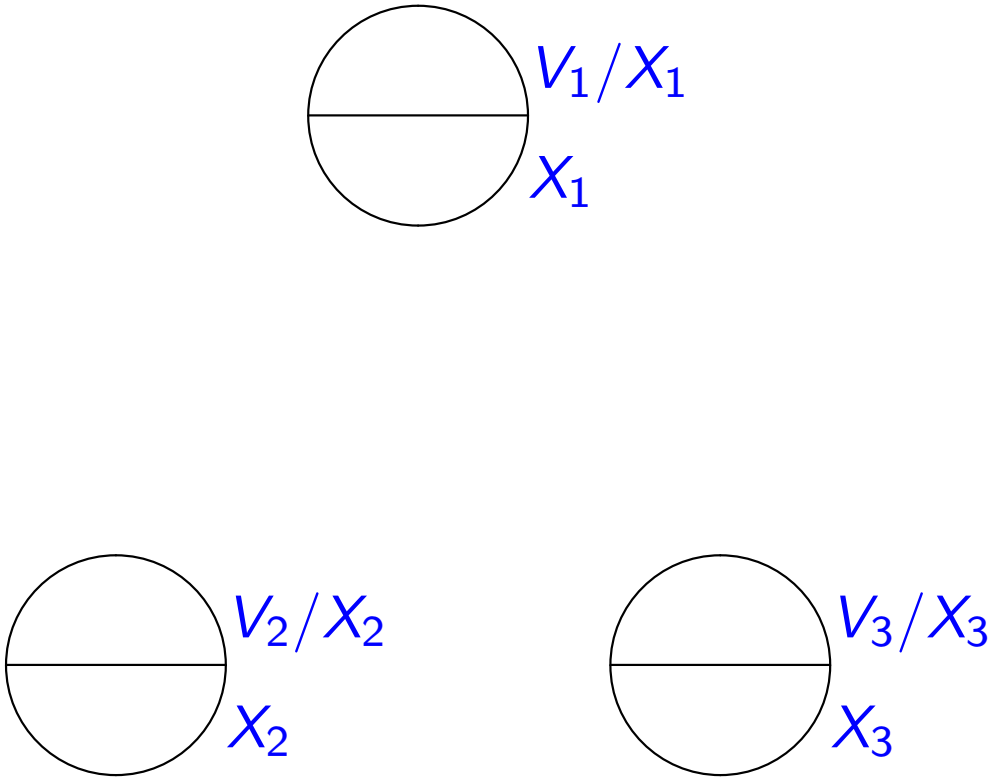


вспомогательный граф H входной граф G

Вспомогательный граф

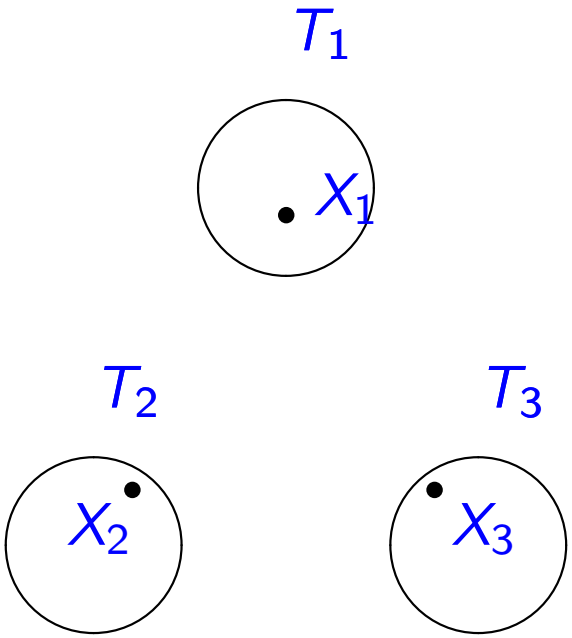


вспомогательный граф H

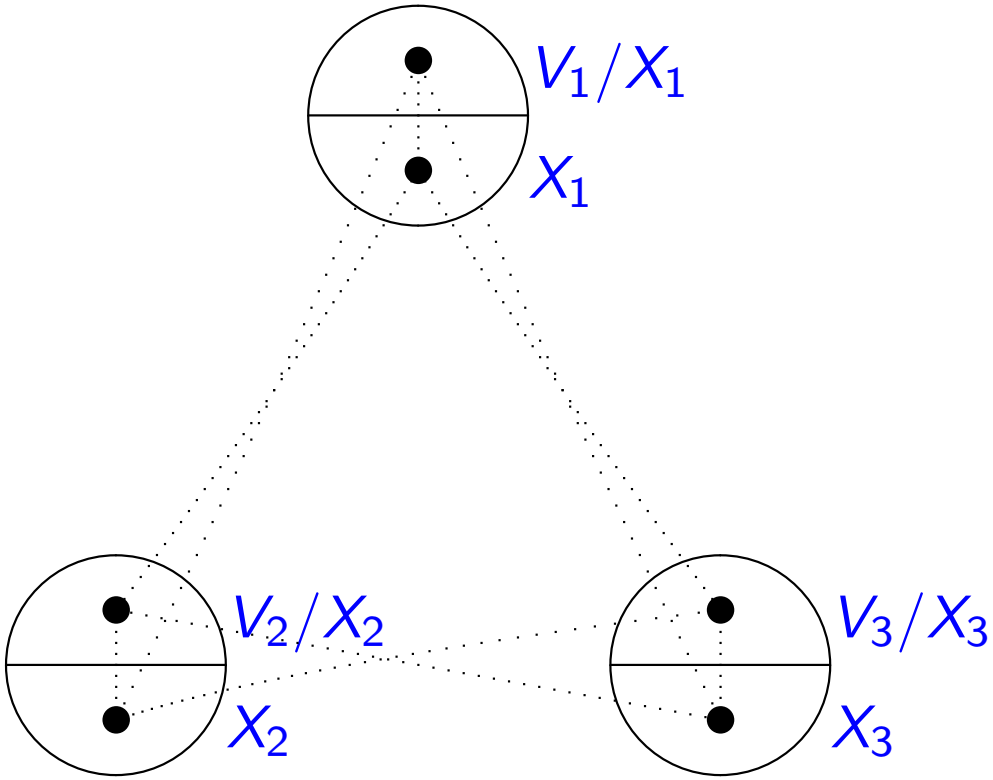


входной граф G

Вспомогательный граф

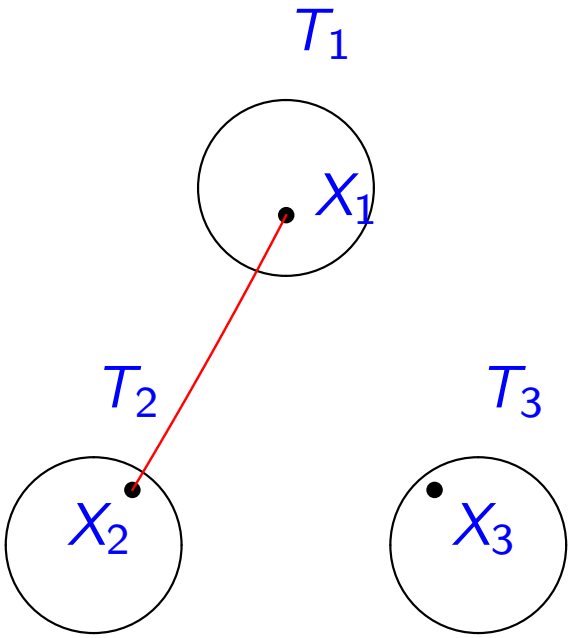


вспомогательный граф H

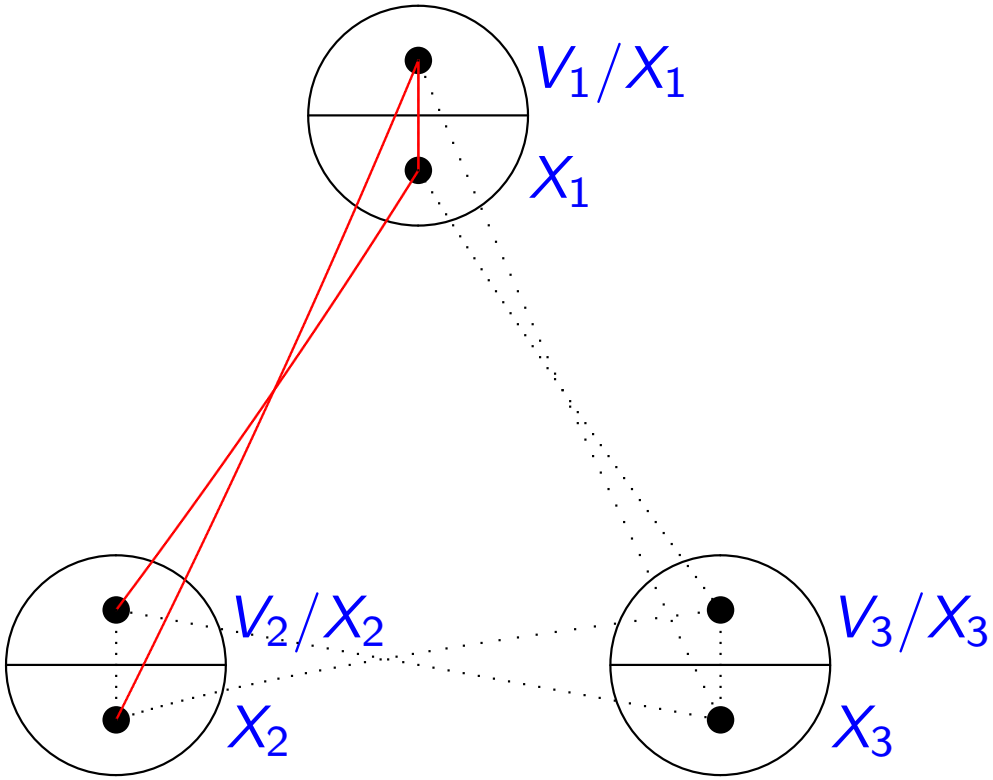


входной граф G

Вспомогательный граф

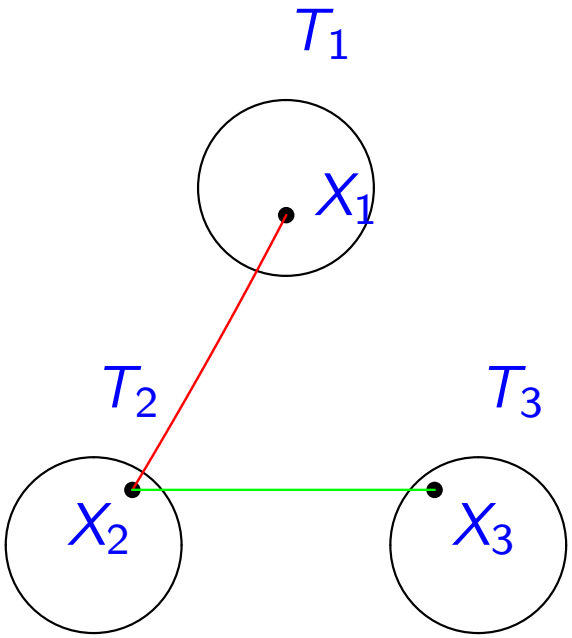


вспомогательный граф H

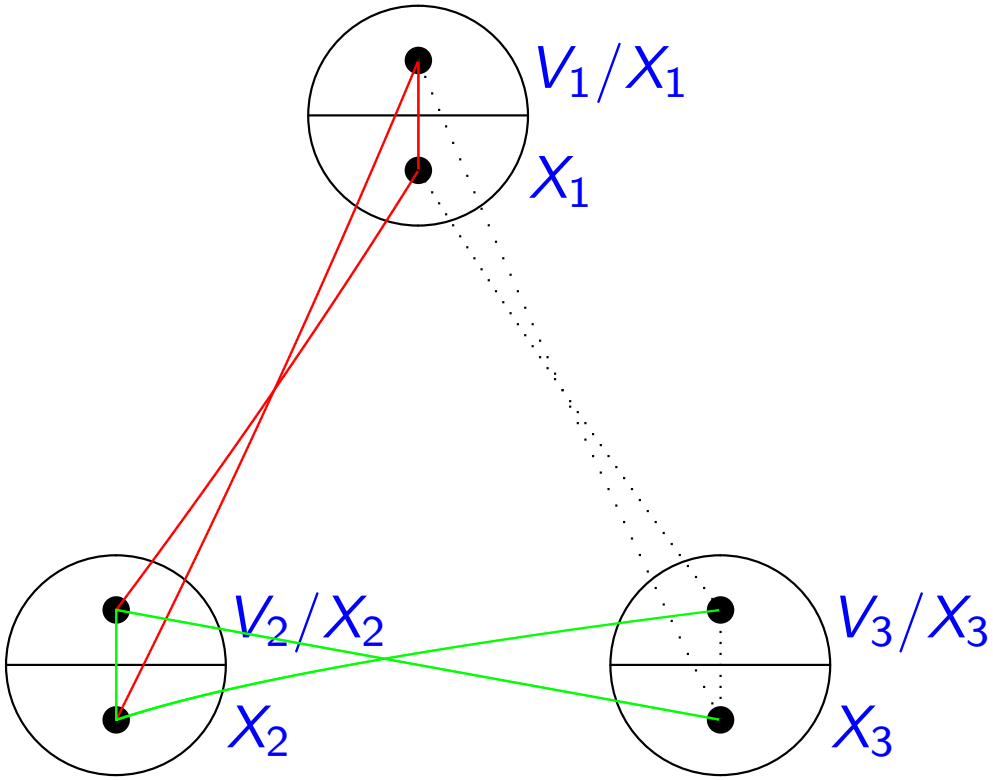


входной граф G

Вспомогательный граф

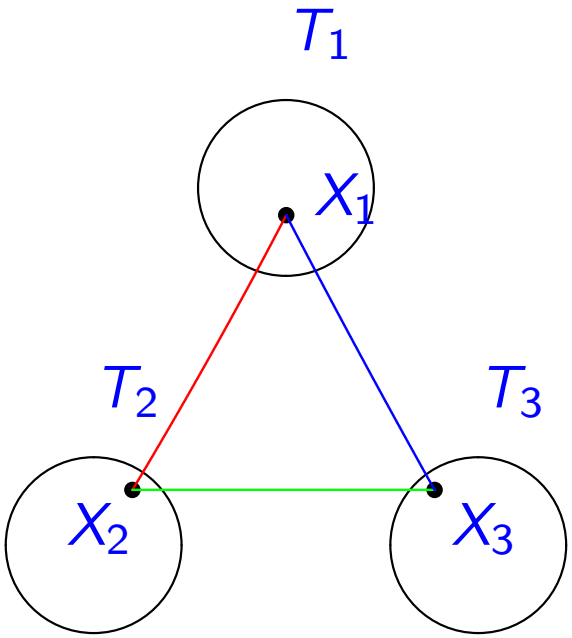


вспомогательный граф H

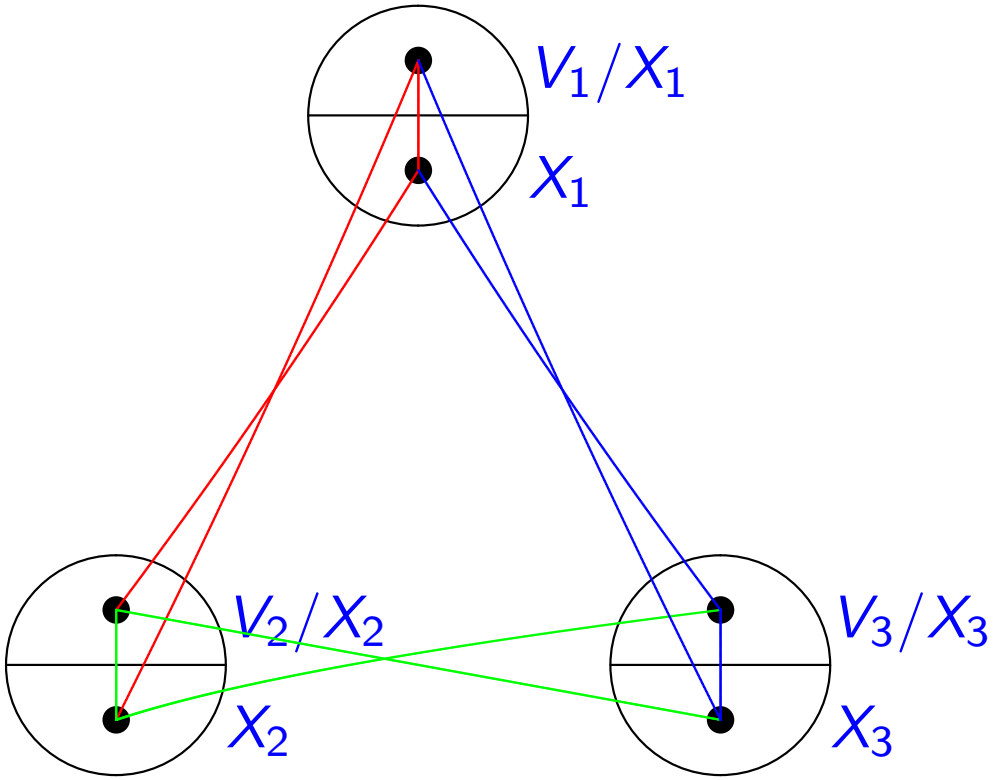


входной граф G

Вспомогательный граф



вспомогательный граф H



входной граф G

Алгоритм

Алгоритм

MATRIX-MAX-CUT(G)

Алгоритм

Алгоритм

MATRIX-MAX-CUT(G)

- разбить множество вершин V на три равные части V_1, V_2, V_3

Алгоритм

Алгоритм

Matrix-MAX-CUT(G)

- разбить множество вершин V на три равные части V_1, V_2, V_3
- построить вспомогательный трехдольный граф H: в i-я доля T_i содержит все возможные непустые подмножества V_i; вес ребра между X_1 и X_2 равен

$$w(V_2 \setminus X_2, X_1) + w(V_1 \setminus X_1, X_1) + w(V_1 \setminus X_1, X_2)$$

(для остальных пар долей — аналогично)

Алгоритм

Алгоритм

Matrix-MAX-CUT(G)

- разбить множество вершин V на три равные части V_1, V_2, V_3
- построить вспомогательный трехдольный граф H: в i-я доля T_i содержит все возможные непустые подмножества V_i; вес ребра между X_1 и X_2 равен

$$w(V_2 \setminus X_2, X_1) + w(V_1 \setminus X_1, X_1) + w(V_1 \setminus X_1, X_2)$$

  (для остальных пар долей — аналогично)
- для всех $1 \leq w_{12}, w_{13}, w_{23} \leq |E_G|$

Алгоритм

Алгоритм

Matrix-MAX-CUT(G)

- разбить множество вершин V на три равные части V_1, V_2, V_3
- построить вспомогательный трехдольный граф H: в i-я доля T_i содержит все возможные непустые подмножества V_i; вес ребра между X_1 и X_2 равен

$$w(V_2 \setminus X_2, X_1) + w(V_1 \setminus X_1, X_1) + w(V_1 \setminus X_1, X_2)$$

  (для остальных пар долей — аналогично)
- для всех $1 \leq w_{12}, w_{13}, w_{23} \leq |E_G|$
  - оставить только ребра веса w_{ij} между долями T_i и T_j

Алгоритм

Алгоритм

Matrix-MAX-CUT(G)

- разбить множество вершин V на три равные части V_1, V_2, V_3
- построить вспомогательный трехдольный граф H: в i-я доля T_i содержит все возможные непустые подмножества V_i; вес ребра между X_1 и X_2 равен

$$w(V_2 \setminus X_2, X_1) + w(V_1 \setminus X_1, X_1) + w(V_1 \setminus X_1, X_2)$$

  (для остальных пар долей — аналогично)
- для всех $1 \leq w_{12}, w_{13}, w_{23} \leq |E_G|$
  - оставить только ребра веса w_{ij} между долями T_i и T_j
  - проверить, есть ли в модифицированном графе H 3-клика

Алгоритм

Алгоритм

Matrix-MAX-CUT(G)

- разбить множество вершин V на три равные части V_1, V_2, V_3
- построить вспомогательный трехдольный граф H: в i-я доля T_i содержит все возможные непустые подмножества V_i; вес ребра между X_1 и X_2 равен

$$w(V_2 \setminus X_2, X_1) + w(V_1 \setminus X_1, X_1) + w(V_1 \setminus X_1, X_2)$$

  (для остальных пар долей — аналогично)
- для всех $1 \leq w_{12}, w_{13}, w_{23} \leq |E_G|$
  - оставить только ребра веса w_{ij} между долями T_i и T_j
  - проверить, есть ли в модифицированном графе H 3-клика
  - если да, то в G есть разрез веса $w_{12} + w_{13} + w_{23}$

Алгоритм

Алгоритм

Matrix-MAX-CUT(G)

- разбить множество вершин V на три равные части V_1, V_2, V_3
- построить вспомогательный трехдольный граф H: в i-я доля T_i содержит все возможные непустые подмножества V_i; вес ребра между X_1 и X_2 равен

$$w(V_2 \setminus X_2, X_1) + w(V_1 \setminus X_1, X_1) + w(V_1 \setminus X_1, X_2)$$

  (для остальных пар долей — аналогично)
- для всех $1 \leq w_{12}, w_{13}, w_{23} \leq |E_G|$
  - оставить только ребра веса w_{ij} между долями T_i и T_j
  - проверить, есть ли в модифицированном графе H 3-клика
  - если да, то в G есть разрез веса $w_{12} + w_{13} + w_{23}$
- вернуть максимальную найденную стоимость разреза

Открытые вопросы

Открытые вопросы

Открытые вопросы

Открытые вопросы

- Придумать алгоритм быстрее.

Открытые вопросы

Открытые вопросы

- Придумать алгоритм быстрее.
- Придумать алгоритм, работающий за время 1.99^n, но полиномиальный по памяти.

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость
7. Случайное сведение к простой задаче
   - 3-раскрашиваемость

Умный перебор

Основная идея

Используя свойства конкретной задачи, перебирать кандидатов на решения эффективно.

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость
7. Случайное сведение к простой задаче
   - 3-раскрашиваемость

3-раскрашиваемость

Определение

Задача 3-раскрашиваемости (3-coloring problem) заключается в проверке, можно ли раскрасить данный граф правильным образом в три цвета (смежные вершины не покрашены в один цвет).

3-раскрашиваемость

Определение

Задача 3-раскрашиваемости (3-coloring problem) заключается в проверке, можно ли раскрасить данный граф правильным образом в три цвета (смежные вершины не покрашены в один цвет).

Полный перебор

Всего кандидатов на решение — 3^n.

Умный перебор

Алгоритм

Simple-Clever-Enum(G)

Умный перебор

Алгоритм

Simple-Clever-Enum(G)

- НУО, граф связен

Умный перебор

Алгоритм

Simple-Clever-Enum(G)

- НУО, граф связен
- зафиксировать цвет 1 для какой-нибудь вершины

Умный перебор

Алгоритм

Simple-Clever-Enum(G)

- НУО, граф связен
- зафиксировать цвет 1 для какой-нибудь вершины
- последовательно выбирать вершины, у которых есть хотя бы один уже покрашенный сосед и рассматривать два варианта ее покраски

Умный перебор

Алгоритм

Simple-Clever-Enum(G)

- НУО, граф связен
- зафиксировать цвет 1 для какой-нибудь вершины
- последовательно выбирать вершины, у которых есть хотя бы один уже покрашенный сосед и рассматривать два варианта ее покраски

Лемма

Алгоритм Simple-Clever-Enum имеет время работы 2^n.

Еще один умный перебор

Алгоритм

Clever-Enum(G)

Еще один умный перебор

Алгоритм

Clever-Enum(G)

- для каждого $S \subseteq \{1, \ldots, n\}$ размера не более $n/3$

Еще один умный перебор

Алгоритм

Clever-Enum(G)

- для каждого $S \subseteq \{1, \ldots, n\}$ размера не более $n/3$
  - покрасить вершины из S в цвет 1

Еще один умный перебор

Алгоритм

Clever-Enum(G)

- для каждого $S \subseteq \{1, \ldots, n\}$ размера не более $n/3$
  - покрасить вершины из S в цвет 1
  - покрасить оставшиеся вершины в цвета 2 и 3

Еще один умный перебор

Алгоритм

Clever-Enum(G)

- для каждого $S \subseteq \{1, \ldots, n\}$ размера не более $n/3$
  - покрасить вершины из S в цвет 1
  - покрасить оставшиеся вершины в цвета 2 и 3
  - если покраска правильная, выдать “да”

Еще один умный перебор

Алгоритм

Clever-Enum(G)

- для каждого $S \subseteq \{1, \ldots, n\}$ размера не более $n/3$
  - покрасить вершины из S в цвет 1
  - покрасить оставшиеся вершины в цвета 2 и 3
  - если покраска правильная, выдать “да”
- выдать “нет”

Анализ алгоритма

Лемма

Алгоритм Clever-Enum корректно выдает решение за время 1.89^n.

Доказательство

Анализ алгоритма

Лемма

Алгоритм Clever-Enum корректно выдает решение за время 1.89^n.

Доказательство

- в любой раскраске один из цветов встречается не более $n/3$ раз

Анализ алгоритма

Лемма

Алгоритм Clever-Enum корректно выдает решение за время 1.89^n.

Доказательство

- в любой раскраске один из цветов встречается не более $n/3$ раз
- будем считать, что это всегда первый цвет

Анализ алгоритма

Лемма

Алгоритм Clever-Enum корректно выдает решение за время 1.89^n.

Доказательство

- в любой раскраске один из цветов встречается не более $n/3$ раз
- будем считать, что это всегда первый цвет
- 2-раскрашиваемость решается за линейное время

Анализ алгоритма

Лемма

Алгоритм Clever-Enum корректно выдает решение за время 1.89^n.

Доказательство

- в любой раскраске один из цветов встречается не более $n/3$ раз
- будем считать, что это всегда первый цвет
- 2-раскрашиваемость решается за линейное время
- $\binom{n}{n/3} \leq 2^{H(1/3)n} \leq 1.89^n$, где H — бинарная энтропия:

$$H(x) = x \log_2 \left(\frac{1}{x}\right) + (1-x) \log_2 \left(\frac{1}{1-x}\right)$$

□

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость
7. Случайное сведение к простой задаче
   - 3-раскрашиваемость

Случайное сведение к простой задаче

Основная идея

Используя случайные биты, так изменить входной пример задачи, чтобы для поиска решения понадобилось полиномиальное время.

План лекции

1. Расщепление
   - Выполнимость (автоматическое доказательство)
2. Случайный порядок перебора
   - Выполнимость
3. Локальный поиск
   - 3-выполнимость
   - k-выполнимость
4. Динамическое программирование
   - Задача о коммивояжере
5. Сведение к простой задаче
   - Сумма подмножества
   - Максимальный разрез
6. Умный перебор
   - 3-раскрашиваемость
7. Случайное сведение к простой задаче
   - 3-раскрашиваемость

3-раскрашиваемость

Алгоритм

3-раскрашиваемость

Алгоритм

- повторить $c \cdot 1.5^n$ раз:

3-раскрашиваемость

Алгоритм

- повторить $c \cdot 1.5^n$ раз:
  - для каждой вершины выбрать случайным образом один из трех цветов, в который данная вершина не покрашена

3-раскрашиваемость

Алгоритм

- повторить $c \cdot 1.5^n$ раз:
  - для каждой вершины выбрать случайным образом один из трех цветов, в который данная вершина не покрашена
  - записать формулу в 2-КНФ, которая выполнима тогда и только тогда, когда граф можно раскрасить с заданными ограничениями: для ребра (u, v), где u покрашена в цвет 1 или 2, а v — в цвет 2 или 3, запишем клозы

$$(u_1 \vee u_2) \wedge (v_2 \vee v_3) \wedge (\neg u_2 \vee \neg v_2)$$

3-раскрашиваемость

Алгоритм

- повторить $c \cdot 1.5^n$ раз:
  - для каждой вершины выбрать случайным образом один из трех цветов, в который данная вершина не покрашена
  - записать формулу в 2-КНФ, которая выполнима тогда и только тогда, когда граф можно раскрасить с заданными ограничениями: для ребра (u, v), где u покрашена в цвет 1 или 2, а v — в цвет 2 или 3, запишем клозы

$$(u_1 \vee u_2) \wedge (v_2 \vee v_3) \wedge (\neg u_2 \vee \neg v_2)$$

  - если полученная формула выполнима, выдать ответ “да”

3-раскрашиваемость

Алгоритм

- повторить $c \cdot 1.5^n$ раз:
  - для каждой вершины выбрать случайным образом один из трех цветов, в который данная вершина не покрашена
  - записать формулу в 2-КНФ, которая выполнима тогда и только тогда, когда граф можно раскрасить с заданными ограничениями: для ребра (u, v), где u покрашена в цвет 1 или 2, а v — в цвет 2 или 3, запишем клозы

$$(u_1 \vee u_2) \wedge (v_2 \vee v_3) \wedge (\neg u_2 \vee \neg v_2)$$

  - если полученная формула выполнима, выдать ответ “да”
- выдать ответ “нет”

Спасибо за внимание!