Лекция 13 — SAT

Содержание

1. Постановка задачи
2. Сложность задачи выполнимости
    1. Классы сложности алгоритмов
    2. Теорема Кука — Левина
3. 2-SAT
4. Алгоритмы решения SAT
    1. Упрощение выражений
    2. DPLL
    3. CDCL
    4. Эвристики

Постановка задачи

Определение

Булево выражение φ называется **выполнимым**, если существует хотя бы один набор булевых переменных, при котором оно обращается в истину. Формально,

$$\exists (x_1, \dots, x_n) \in \{0,1\}^n : \varphi(x_1, \dots, x_n) = 1$$

Удовлетворяющий булеву выражению набор переменных называется *моделью*.

Задача выполнимости булева выражения имеет две формулировки:

1. Проверить выполнимость произвольной КНФ (*decision SAT*);
2. Найти модель выполнимой КНФ (*search SAT*).

Определение

Будем называть булево выражение φ **валидным**, если оно является *тавтологией*:

$$\forall (x_1, \dots, x_n) \in \{0,1\}^n : \varphi(x_1, \dots, x_n) = 1$$

Двойственность SAT-TAUT

Булево выражение φ валидно тогда и только тогда, когда $\neg\varphi$ невыполнимо!

Сложность задачи выполнимости

Классы сложности алгоритмов

1. P (*polynomial*)

Класс сложности P состоит из задач, которые можно решить за полиномиальное время при помощи детерминированного алгоритма.

2. NP (*non-deterministic polynomial*)

Класс сложности NP состоит из задач, ответ на которые может быть проверен за полиномиальное время при помощи детерминированного алгоритма.
Эквивалентно, задачи из класса NP допускают полиномиальное решение при помощи недетерминированного алгоритма.

2.1 NP-hard

Задача считается NP-сложной, если любая задача из класса NP сводится к ней за полиномиальное время.

2.2 NP-complete

NP-сложная задача, принадлежащая классу NP, считается NP-полной.

Теорема Кука — Левина

Теорема Кука — Левина

Задача выполнимости является NP-полной.

Идея доказательства

Легко показать, что SAT лежит в классе NP: мы можем проверить каждый набор переменных за линейное время.

Чтобы показать, что SAT — NP-сложная задача, т.е. что любая задача из класса NP сводится к задаче выполнимости, можно представить процесс выполнения любого

алгоритма в качестве набора переменных, задающих состояние выполняющей машины, и объединить их в дизъюнкты, задав тем самым все необходимые для решения исходной задачи условия. Таким образом мы можем проверить любой набор входных данных: если он удовлетворяет полученной формуле, то он является решением задачи.

2-SAT

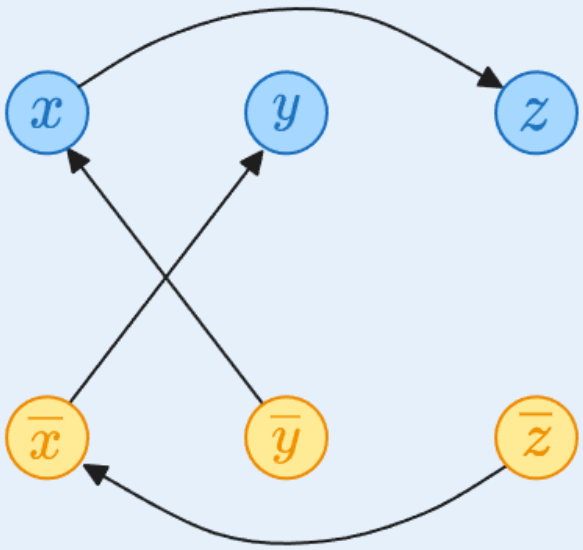
Определение

Задачей **k-SAT** будем называть задачу выполнимости выражения, в котором каждый дизъюнкт содержит ровно k литералов.

Определение

Графом импликаций для 2-SAT формулы φ называется ориентированный граф G_φ, где:

- вершинами являются литералы x и $\neg x$ для каждой переменной x
- для каждого дизъюнкта $(a \vee b)$ проводятся рёбра $\neg a \to b$ и $\neg b \to a$



Граф импликаций для формулы $(x \vee y) \wedge (\neg x \vee z)$

Теорема

2-SAT формула невыполнима тогда и только тогда, когда она содержит такую переменную x, что x и $\neg x$ содержатся в одной компоненте сильной связности.

Идея доказательства

Ребро $\neg u \to v$ в графе импликаций означает, что если значение переменной u равняется 0, то переменная v *обязана* принять значение 1. Отсюда легко видеть, почему x и $\neg x$ не могут быть сильно связаны.

Вторая часть утверждения — достаточность условия — доказывается конструктивно путём конденсации графа импликаций и его последующего обхода.

⚠ **Важно!**

Из теоремы следует, что задача 2-SAT имеет **полиномиальное** решение!
Это один из немногих видов задачи выполнимости, который относится к классу P.

Алгоритмы решения SAT

Упрощение выражений

Распространение единичных литералов (*Unit propogation*)

Если формула содержит единичный дизъюнкт (l), то легко видеть, что в любой её модели он будет принимать только истинное значение. В силу этого можно проделать следующие преобразования без потери выполнимости:

- исключить все дизъюнкты, которые содержат литерал l (они автоматически удовлетворены);
- исключить литерал $\neg l$ из всех дизъюнктов.

Исключение чистых литералов (*Pure literal elimination*)

Назовём литерал *чистым*, если формула не содержит его отрицания.
Легко заметить, что тогда ему можно присвоить истинное значение без потери выполнимости: если дизъюнкция была истинна при $x = 0$, она останется таковой при $x = 1$. Именно потому, что литерал чистый, это правило распространяется на всё выражение, т.е. ни один дизъюнкт при такой замене не становится ложным, если не был им первоначально.
Итак, если литерал l чистый, мы можем:

- присвоить l истинное значение ($l = 0$, если встречается только $\neg l$, и $l = 1$ иначе);
- исключить l из выражения.

DPLL

DPLL (Davis-Putnam-Logmeann-Loveland) — классический алгоритм решения задачи выполнимости. Он проделывает следующие шаги:

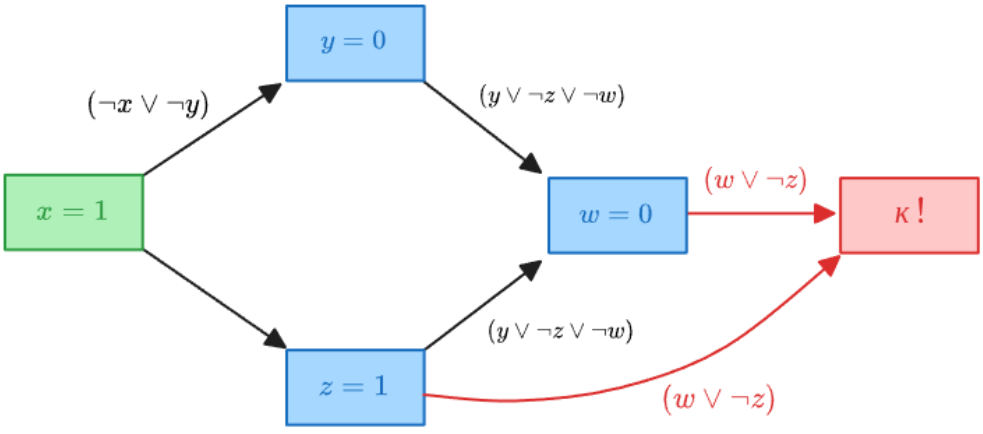
> 1. **Упрощение выражения**
>
> На первом шаге DPLL упрощает выражение, используя два описанные выше правила. Если при этом дизъюнктов не остаётся, выражение выполнимо. Если при этом в результатате распространения единичных литералов возникает пустой дизъюнкт, выражение невыполнимо.

> 2. **Перебор с возвратами** (*Backtracking*)
>
> На втором шаге DPLL выбирает произвольную переменную, присваивает ей значение 1 и рекурсивно пытается решить получившуюся упрощённую формулу. Если формула выполнима, он возвращает SAT; иначе он возвращается и пытается присвоить ей значение 0, после чего возвращает результат.

CDCL

CDCL (Conflict-Driven Clause Learning) — улучшенная версия алгоритма DPLL, которая анализирует причину каждого конфликта, приведшего к неудаче на втором шаге DPLL, и использует это знание для отсечения больших кусков дерева поиска.

> **Механизм работы CDCL**
>
> CDCL использует *граф импликаций*, чтобы отследить, почему каждой из переменных было присвоено то или иное значение:



> CDCL использует правило *резолюции*, которое позволяет объединить конфликтный дизъюнкт с его причиной по принципу
>
> $$(A \vee B) \wedge (\neg B \vee C) \implies (A \vee C)$$

Поскольку процесс происходит «назад», резолюция позволяет шаг за шагом исключать посредников и прийти к исходной причине конфликта. Как только она была найдена, CDCL добавляет к исходному выражению получившийся конъюнкт, запрещая конфликтную комбинацию значений, и продолжает свою работу.

Схемы обучения

Классическая реализация CDCL продолжает процесс резолюции до тех пор, пока он не дойдёт до такой вершины в графе, через которую проходят все пути, приводящие к конфликту. Такой подход называется *first UIP* (*unique implication point*) и является наиболее оптимальным, так как даёт минимальный возможный дизъюнкт.

Существуют также схемы *all UIP* (поиск всех UIP на текущем уровне), *last UIP* (поиск дизъюнкта, содержащего только переменные, выбранные на последнем шаге) и *RelSAT* (поиск дизъюнкта, содержащего все конфликтующие переменные).

Примечание. Стоит проверить правдивость утверждений про альтернативные схемы, прежде чем ими пользоваться — глубокого их разбора не проводилось.

Эвристики

Phase saving

Идея этой эвристики — сохранять последнее успешное присвоение этой переменной и предпочитать его при следующем обращении к ней.

Two-watched literals

Легко заметить, что дизъюнкт не может стать единичным до тех пор, пока хотя бы два литерала в нём не станут однозначно ложны. Мы можем выбрать для каждого дизъюнкта два таких произвольных литерала, и обращаться к нему только тогда, когда произойдут манипуляции хотя бы с одним из них.

VSIDS (*Variable State Independent Decaying Sum*)

Эта эвристика работает следующим образом:

- у каждой переменной есть счётчик активности
- когда переменная участвует в конфликте, счётчик активности увеличивается
- счётчик активности каждой переменной периодически уменьшается
- на каждом шаге выбираются переменные с наибольшим счётом

Идея в том, чтобы обращаться к наиболее конфликтным переменным и, как следствие, наиболее оптимально разрешать эти конфликты. При этом старые

конфликты имеют меньший вес — это достигается постепенным уменьшением счётчиков.

Перезапуски

Иногда CDCL «застревает» в неудачной части дерева решений, поэтому бывает полезно перезапускать его, сохраняя при этом все выученные дизъюнкты. Политика, определяющая условие перезапуска, при этом может разниться.