Московский физико-технический институт
Факультет инноваций и высоких технологий
Математическая логика и теория алгоритмов, осень 2015
Лекции 3–4: **NP**-полнота

Краткое содержание

Полиномиальная сводимость. Её свойства: рефлексивность, транзитивность, связь с **P** и **NP**. Понятия **NP**-трудности и **NP**-полноты. **NP**-полная задача общего вида. Задачи SAT и 3SAT, сводимость второй к первой. Теорема Кука–Левина об **NP**-полноте задачи SAT. **NP**-полнота задач о клике, независимом множестве, вершинном покрытии, раскрасках, ориентированном и неориентированном гамильтоновом пути, рюкзаке, некоторых задач о выполнимости булевых формул, целочисленном и квадратичном линейном программировании. Задачи поиска для языков из **NP**. Принцип самосводимости. Если $\mathbf{P} = \mathbf{NP}$, то все задачи поиска решаются за полиномиальное время. Возможные подходы к практическому решению **NP**-полных задач: частные случаи, эвристики, приближения.

Теория **NP**-полноты — краеугольный камень структурной теории сложности, с которого началась эта наука в 1970-х годах. **NP**-полные задачи — в некотором роде самые сложные в классе **NP**. Если про какую-то задачу доказана **NP**-полнота, то можно не надеяться найти её точное и полное решение, лучше искать другие пути.

1 Полиномиальная сводимость по Карпу

В теории алгоритмов известно два основных вида сводимости одних задач к другим: m-сводимость и T-сводимость, или сводимость по Тьюрингу. В теории сложности вычислений им соответствуют сводимость по Карпу и сводимость по Куку. Для нас наиболее важной является полиномиальная сводимость по Карпу.

Определение 1. Пусть A и B суть два языка. Тогда A сводится по Карпу к B, если существует всюду определённая функция $f\colon \{0,1\}^* \to \{0,1\}^*$, вычислимая за полиномиальное время, такая что $x \in A \Leftrightarrow f(x) \in B$. Обозначение: $A \leqslant_p B$. (Индекс p означает полиномиальность).

Можно заметить несколько простых свойств такой сводимости:

Утверждение 2. *Имеют место следующие факты:*

а) *Полиномиальная сводимость рефлексивна:* $A \leqslant_p A$;
б) *Полиномиальная сводимость транзитивна: если* $A \leqslant_p B$ *и* $B \leqslant_p C$, *то* $A \leqslant_p C$;
в) *Если* $A \in \mathbf{P}$, *а* $B \neq \varnothing$ *и* $B \neq \{0,1\}^*$, *то* $A \leqslant_p B$;
г) *Если* $B \in \mathbf{P}$ *и* $A \leqslant_p B$, *то* $A \in \mathbf{P}$;
д) *Если* $B \in \mathbf{NP}$ *и* $A \leqslant_p B$, *то* $A \in \mathbf{NP}$.

Доказательство. Рефлексивность очевидна: достаточно рассмотреть функцию $f(x) = x$. Для транзитивности нужно рассмотреть композицию: если f сводит A к B, а g сводит B к C, то $g \circ f$ сводит A к C: $x \in A \Leftrightarrow f(x) \in B \Leftrightarrow g(f(x)) \in C$. При этом если и f, и g вычисляются за полиномиальное время, то $g \circ f$ также вычисляется за полиномиальное время.

Третье утверждение также несложно: если $B \neq \varnothing$ и $B \neq \{0,1\}^*$, то можно зафиксировать $b_1 \in B$ и $b_2 \notin B$ и рассмотреть функцию $f(x) = \begin{cases} b_1, & x \in A \\ b_2, & x \notin A \end{cases}$. В силу разрешимости A эта функция будет вычислимой.

Четвёртое утверждение самое важное и полезное, но также несложно. Достаточно заметить, что характеристическая функция χ_A представляется как композиция $\chi_B \circ f$. Если $B \in P$, то χ_B вычисляется за полиномиальное время, а если $A \leqslant_p B$, то соответствующая f также вычисляется за полиномиальное время. Значит, их композиция, т.е. χ_A, тоже вычисляется за полиномиальное время, что и означает $A \in \mathbf{P}$.

Наконец, последнее утверждение докажем двумя способами. Во-первых, для определения через недетерминированные машины достаточно заметить, что если χ_B вычисляется недетерминированной машиной за полиномиальное время, а f (детерминированно) вычисляется за полиномиальное время, то $\chi_A = \chi_B \circ f$ также вычисляется недетерминированной машиной за полиномиальное время. Во-вторых, для определения через сертификаты: если $W(y,s)$ будет верификатором принадлежности y к B, то $V(x,s) = W(f(x),s)$ будет верификатором принадлежности x к A: если $x \in A$, то для $f(x)$ будет существовать сертификат и V его примет, а если $x \notin A$, то для $f(x)$ сертификата существовать не будет. □

2 NP-полнота

Неформально говоря, задача трудна для некоторого класса, если её решение позволяет легко решить все задачи из этого класса, и полна для этого же класса, если она и сама лежит в нём. Это определение уточняется следующим образом для сводимости по Карпу:

Определение 3. Язык B является **NP**-*трудным*, если для любого $A \in \mathbf{NP}$ выполнено $A \leqslant_p B$. Язык B является **NP**-*полным*, если он **NP**-трудный и лежит в **NP**.

Имеют место следующие простые утверждения:

Утверждение 4. *Если B является* **NP**-*трудным и $B \in \mathbf{P}$, то* $\mathbf{P} = \mathbf{NP}$.

Доказательство. Действительно, если $A \in \mathbf{NP}$, то $A \leqslant_p B$. А раз $B \in \mathbf{P}$, то и $A \in \mathbf{P}$. Значит, $\mathbf{NP} \subset \mathbf{P}$, т.е. $\mathbf{P} = \mathbf{NP}$. □

Утверждение 5. *Если A является* **NP**-*трудным и $A \leqslant_p B$, то B тоже является* **NP**-*трудным.*

Доказательство. Действительно, любой язык C из **NP** сводится к A, а по транзитивности и к B. □

Несмотря на свою простоту, второе утверждение очень важно, поскольку позволяет получать новые **NP**-трудные задачи: достаточно свести к новой задаче уже известную.

Тривиальным примером **NP**-полного языка является язык $\mathsf{TMSAT} = \{(M, x, 1^t) \mid \exists y M(x, y) = 1$ и $M(x, y)$ заканчивает работу не более чем за t шагов$\}$.

Теорема 6. *Язык* TMSAT *является* **NP***-полным.*

Доказательство. Во-первых, докажем, что $\mathsf{TMSAT} \in \mathbf{NP}$. В качестве сертификата выступит тот самый y, существование которого утверждается в формулировке. При помощи универсальной машины Тьюринга можно проверить, что $M(x, y) = 1$, а если при запуске считать шаги, то и условие на остановку также проверяется. При этом, если $M(x, y)$ сделало больше t шагов, но не остановилось, то нужно остановить её работу, а сертификат отвергнуть. При тривиальной реализации универсальной машины общее число шагов не превысит $O(t^2)$: каждый шаг машины M моделируется $O(t)$ шагами на универсальной машине Тьюринга,[1] а поддержка счётчика увеличит число щагов ещё в $O(\log t)$ раз. Более хитрая реализация позволяет сократить число шагов до $O(t \log t)$.[2] В любом случае время работы будет полиномом от длины входа, т.е. $|M|+|x|+t$. Заметим, что в этом месте важна именно унарная, а не двоичная запись t в списке аргументов.

Во-вторых, докажем, что если $L \in \mathbf{NP}$, то $L \leqslant_p \mathsf{TMSAT}$. Действительно, пусть $V(x.s)$ есть верификатор принадлежности x к L, причём V вычисляется машиной M, работающей не дольше $p(n)$ шагов. Тогда искомой совдимостью будет отображение $x \mapsto (M, x, 1^{p(|x|)})$. Действительно, определение TMSAT как раз и говорит о существовании сертификата. □

Однако задача TMSAT слишком искусственна. По сути, определение **NP** попросту зашито в определение этого языка. Гораздо интереснее содержательные **NP**-полные задачи, возникающие в различных областях логики, дискретной математики, алгебры и других дисциплин.

3 Теорема Кука–Левина

Типичным способом доказательства **NP**-трудности данной задачи является сведение к ней какой-нибудь уже известной **NP**-трудной задачи. Действительно, если B является **NP**-полным, $B \leqslant_p C$ и $C \in \mathbf{NP}$, то C также **NP**-полон: любой другой язык сводится в нему по транзитивности. Во многих случаях в качестве известной задачи выступает задача 3SAT. В этом разделе мы докажем её **NP**-полноту.

Определение 7. Пропозициональная формула называется *выполнимой*, если она истинна хотя бы на каком-то наборе значений переменных. Языком SAT называется множество выполнимых пропозициональных формул.

Определение 8. Пропозициональная формула записана в *3-КНФ*, если она представлена в конъюнктивной нормальной форме, в которой любой дизъюнкт включает ровно три литерала. Языком 3SAT называется множество выполнимых 3-КНФ.

[1] У уинверасльной машины некоторое фиксированное число лент, а у M — произвольное

[2] См. Arora, Barak, раздел 1.7

Вначале докажем несложное

Утверждение 9. $\mathsf{SAT} \leqslant_p \mathsf{3SAT}$.

Доказательство. Пусть φ — пропозициональная формула. Введём переменную для каждой её подформулы (включая исходные переменные и всю формулу). Каждая подформула, составленная из двух, задаёт утверждение вида $q = r \wedge s$. Его можно представить как 3-КНФ (в данном случае $(q \vee \neg r \vee \neg s) \wedge (\neg q \vee r \vee \neg s) \wedge (\neg q \vee \neg r \vee s) \wedge (\neg q \vee r \vee s)$). Взяв конъюнкцию всех таких формул, мы получим 3-КНФ, выполнимость которой эквивалентна выполнимости исходной формулы. Действительно, выполняющий набор исходной формулы задаст значения всех подформул, которые будут выполняющим набором полученной формулы. И наоборот, из выполняющего набора новой формулы можно выделить выполняющий набор исходной.

Осталось пояснить, почему этот алгоритм работает полиномиальное время. Ясно, что достаточно построить за полиномиальное время дерево синтаксического разбора формулы, дальнейшие операции занимают константное время для каждой его вершины. Построение дерева также несложно: путём подсчёта скобочного итога можно выделить из формулы две подформулы, из которых она получена, и повторить процедуру с ними рекурсивно. Это займёт линейное время для каждой подформулы, т.е. всего квадратичное время. □

Теперь докажем важнейшую теорему:

Теорема 10 (Теорема Кука–Левина)**.** *Задача* SAT *является* **NP***-полной.*

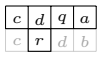
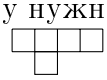
Доказательство. Пусть $L \in \mathbf{NP}$. Рассмотрим верификатор V, работающий с сертификатами длины $q(n)$. Будем считать, что и вход, и сертификат заданы в двоичном алфавите. Будем также считать, что V задан машиной Тьюринга с одной лентой, бесконечной вправо, и работающей не дольше $p(n)$ шагов. Идея состоит в записи булевой формулы, которая моделирует работу этой машины.

Напомним, что конфигурацией называется слово вида aqb, где $q \in Q$, а $a, b \in \Gamma^*$. Имеется в виду, что машина находится в состоянии q и указывает на первый символ слова b, а левее записано слово a. Все остальные ячейки заполнены бланками. Вначале машина находится в конфигурации $q_0 x \# s$, а в конце должна быть в состоянии q_a, если сертификат s верный. Поскольку за 1 шаг машина сдвигается не больше, чем на 1 ячейку вправо, максимальная длина любой конфигурации не больше $p(n)$. Введём служебные символы $\blacktriangleright$ и $\blacktriangleleft$, ограничивающие рабочую часть ленты. В новых обозначениях начальная конфигурация запишется как $\blacktriangleright\ q_0 x \# s \#^{p(n)-n-q(n)-2}\ \blacktriangleleft$. Все возможные символы (элементы Γ, элементы Q, $\blacktriangleright$ и $\blacktriangleleft$), закодируем в двоичной записи, пусть на это потребуется k битов. В таком случае для кодирования всех конфигураций нам потребуется $k(p(n)+2)(p(n)+1)$ битов: k битов на каждый символ, $(p(n)+2)$ символов в каждой конфигурации и $(p(n)+1)$ конфигураций за $p(n)$ шагов, включая начальную и конечную.

Теперь нужно построить формулу, гарантирующую корректность протокола. Она будет конъюнкцией трёх формул, гарантирующих корректность начала, окончания и каждого шага. Для корректности начала нужно гарантировать, что первый символ равен $\blacktriangleright$, следующий — q_0, следующие n символов совпадают с битами x, далее идёт бланк,

далее $q(n)$ битов (т.е. не служебных символов), далее бланки в количестве $p(n) - n - q(n) - 2$ и, наконец, ◀. Каждое такое равенство выражается простой формулой, содержащей k эквиваленций (в случае битов сертификата — дизъюнкций двух наборов из k эквиваленций), таким образом, общая длина конъюнкции всех формул будет порядка $p(n)$.

Для корректности окончания нужно построить формулу, проверяющую, что среди символов последней строки есть q_a. Это делается при помощи дизъюнкции эквиваленций.

Наконец, нужно построить формулу, проверяющую корректность каждого шага. Ключевой идеей здесь является то, что вычисления на машине Тьюринга локальны, т.е. на каждом шаге изменяется небольшая часть ленты, и это изменение зависит от небольшой части ленты на предыдущем шаге. Более точно, значение символа в каждой ячейке зависит от значений символов в 4 ячейках на предыдущем шаге:  (зависимость от самой правой ячейки появляется, когда машина сдвигается налево: например,  для команды $qa \to rbL$). Вид этой зависимости полностью определяется программой машиной Тьюринга, а значит, может быть описан некоторой фиксированной пропозициональной формулой. Эту формулу нужно скопировать $p(n)(p(n)-1)$ раз для каждого из возможных положений фигуры  и взять конъюнкцию всех формул.

Таким образом, мы построили формулу, размер которой есть полином от исходных данных. Её выполнимость равносильна тому, что для некоторого сертификата s машина на входе x остановится в принимающем состоянии, т.е. $x \in L$. Таким образом, L сведён к SAT и теорема доказана. □

4 Примеры NP-полных задач

В этом разделе мы изучим несколько стандартных примеров **NP**-полных задач. Главное, что нужно помнить: для доказательства **NP**-полноты некоторой задачи нужно сводить не её, а к ней. Как правило, мы будем сводить к нашим задачам язык 3SAT. Общая идея заключается в построении специальных «гаджетов» для переменных и для дизъюнктов.

4.1 Клика, независимое множество, вершинное покрытие

Определение 11. Пусть дан неориентированный граф G. *Кликой* называется любой его полный подграф. *Независимым множеством* называется любое множество вершин, между которыми нет рёбер. *Вершинным покрытием* называется множество вершин, в котором лежит хотя бы один конец любого ребра.

С каждым из указанных понятий связана задача оптимизации: найти самую большую клику, самое большое независимое множеств и самое маленькое вершинное покрытие. Соответствующие задачи распознавания формулируются следующим образом:

- CLIQUE $= \{(G, k) \mid$ в графе G есть клика из k вершин$\}$;

- $\mathsf{INDSET} = \{(G, k) \mid$ в графе G есть независимое множество из k вершин$\}$;
- $\mathsf{VERTEXCOVER} = \{(G, k) \mid$в графе G есть вершинное покрытие из k вершин$\}$.

Вначале покажем, что все эти задачи сводятся друг к другу, потом — **NP**-полноту одной из них.

Утверждение 12. *Задачи* CLIQUE, INDSET *и* VERTEXCOVER *полиномиально сводятся друг к другу.*

Доказательство. Задачи CLIQUE и INDSET полиномиально сводятся друг к другу переходом к дополнению графа, т.е. графу $\overline{G}$, в котором те и только те рёбра, которых нет в G.

Задачи INDSET и VERTEXCOVER сводятся друг к другу заменой k на $n - k$, где n — число вершин. Дело в том, что дополнение к любому вершинному покрытию есть независимое множество: если вершинное покрытие задело все рёбра, то между оставшимися вершинами рёбер быть не может, иначе эти рёбра не были бы задеты. И наоборот, если между оставшимися рёбер нет, то все рёбра покрыты взятыми. □

Теорема 13. *Задача* INDSET *является* **NP**-*полной.*

Доказательство. Докажем, что $\mathsf{3SAT} \leqslant_p \mathsf{INDSET}$. Построим граф следующим образом: каждому вхождению литерала сопоставим вершину графа. Таким образом, всего будет $3k$ вершин, где k — число дизъюнктов в 3-КНФ. Вершины, соответствующие литералам из одного дизъюнкта, соединим рёбрами. Также соединим все противоположные литералы, например p и $\neg p$. На этом построение графа заканчивается, размер независимого множества возьмём равным k.

Если у формулы есть выполняющий набор, то в каждом дизъюнкте есть хотя бы один истинный литерал. Соответствующие вершины образуют независимое множество, поскольку эти литералы из разных дизъюнктов, а литералы вида p и $\neg p$ не могут быть истинны одновременно. Этих вершин как раз k.

Если, наоборот, есть независимое множество из k вершин, то эти вершины обязаны соответствовать различным дизъюнктам. А поскольку среди этих вершин нет одновременно литералов вида p и $\neg p$, то можно взять такие значения переменных, при которых все задействованные литералы истинны. Эти значения и будут выполняющим набором. □

4.2 Раскраски

Лес, точно терем расписной,
Лиловый, золотой, багряный...
И.А.Бунин, *Листопад (1900)*

Важнейшей характеристикой графа является его хроматическое число, т.е. минимальное число цветов, в которые можно покрасить вершины графа, так чтобы вершины одного цвета не были соединены ребром. Соответствующей задачей распознавания будет задача $\mathsf{COL} = \{(G, k) \mid$ вершины графа G можно правильно раскрасить в k цветов$\}$.

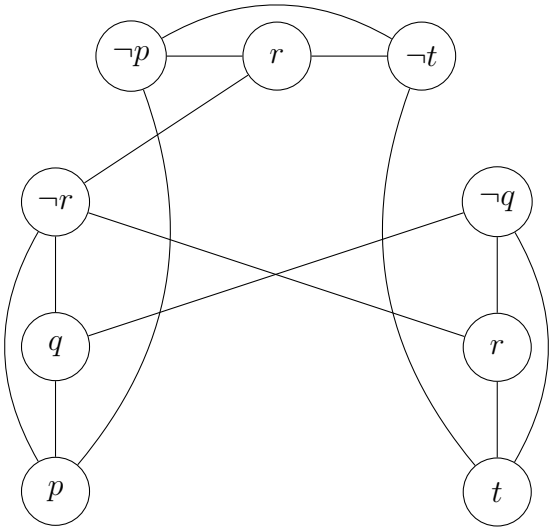
$(p \vee q \vee \neg r) \wedge (\neg p \vee r \vee \neg t) \wedge (\neg q \vee r \vee t)$

(а) Формула в виде 3-КНФ

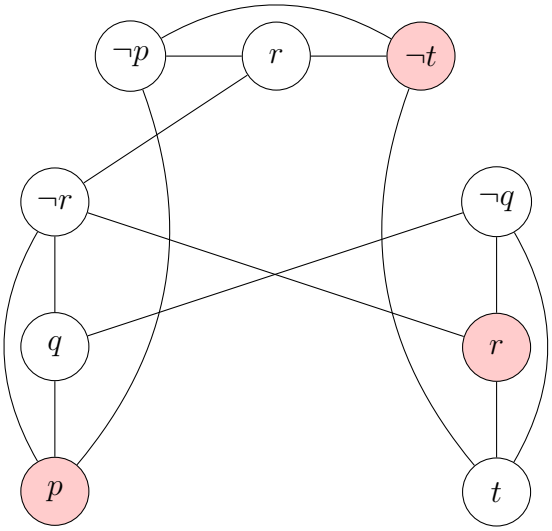
$p = 1,\ q = 1,\ r = 1,\ t = 0$

$(\mathbf{p} \vee q \vee \neg r) \wedge (\neg p \vee r \vee \neg \mathbf{t}) \wedge (\neg q \vee \mathbf{r} \vee t)$

(б) Выполняющий набор



(в) Граф, построенный по формуле

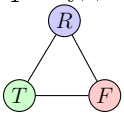


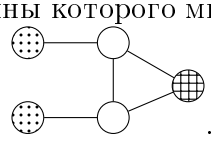
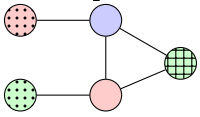
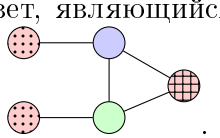
(г) Независимое множество в графе

Рис. 1: Сводимость 3SAT к INDSET.

Если рассмотреть частные случаи для $k = 1$ и $k = 2$, то задача будет, очевидно, полиномиальной: в первом случае достаточно проверить пустоту графа, во втором — двудольность. А вот задача 3COL о раскраске в 3 цвета уже будет **NP**-полной. Как следствие, **NP**-полной будет и общая задача COL.

Теорема 14. *Задача* 3COL *является* **NP***-полной.*

Доказательство. 3 цвета, в которые нужно будет красить граф, назовём истинным (обозначать будем зелёным), ложным (обозначать будем красным) и синим. Частью графа будет«палитра» — треугольник, вершины которого мы покрасим в указанные цвета: 

. Далее рассмотрим такой гаджет: 
. Непосредственным перебором проверяется такое утверждение: если гаджет правильно раскрашен в 3 цвета, при этом выделенные вершины покрашены в цвета T или F, то клетчатая вершина покрашена в цвет, являющийся дизъюнкцией цветов точечных вершин, например: 
или 
. Состыковав два таких гаджета между собой, можно получить ана-

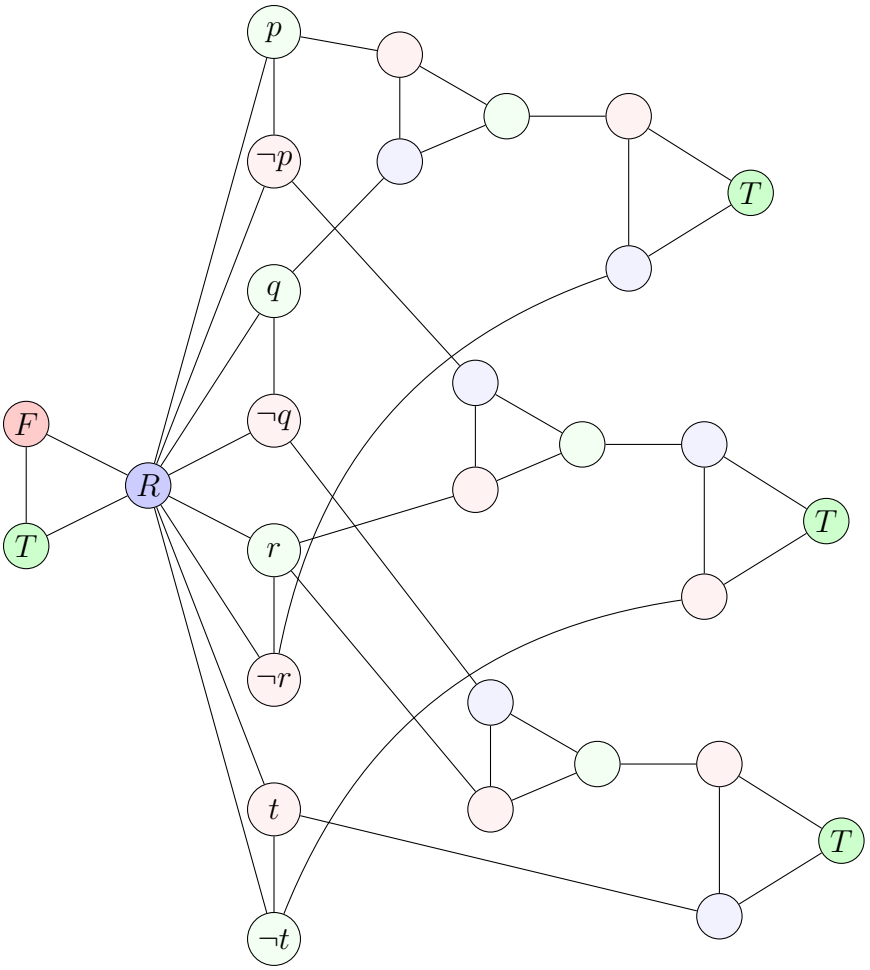
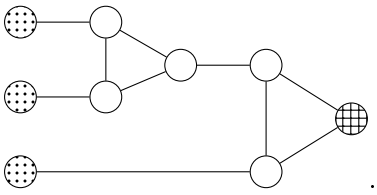


Рис. 2: Сводимость 3SAT к 3COL на примере формулы $(p \vee q \vee \neg r) \wedge (\neg p \vee r \vee \neg t) \wedge (\neg q \vee r \vee t)$. Более яркими цветами показаны жёстко зафиксированные цвета, более слабыми — раскраска, соответствующая выполняющему набору $p = q = r = 1, t = 0$. Вершина T нарисована несколько раз для удобства восприятия.



логичный гаджет для дизъюнкции трёх переменных: . Теперь всё готово для построения всего графа. Во-первых, мы включим в него палитру. Во-вторых, для каждой переменной сделаем две вершины, соединённые между собой и с вершиной R. Это будет гарантировать, что они покрашены в цвета T и F, причём разные. Один вариант будет соответствовать истинности переменной, другой — ложности. Поэтому и назовём эти вершины p и $\neg p$. Наконец, для каждого дизъюнкта гаджет тройной дизъюнкции, подключим его точечные вершины к соответствующим литералам, а клетчатую — к вершине T. На рис. 2 показан граф для формулы $(p \vee q \vee \neg r) \wedge (\neg p \vee r \vee \neg t) \wedge (\neg q \vee r \vee t)$.

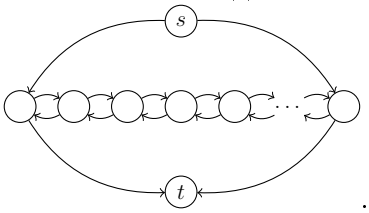
Докажем, что выполнимость исходной формулы эквивалентна 3-раскрашиваемости

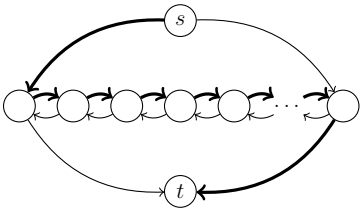
полученного графа. Пусть формула выполнима. Покрасим в зелёный истинные вершины-литералы и в красный ложные. В силу выполнимости к каждому гаджету будет полключена хотя бы одна зелёная вершина, и он успешно раскрасится. Обратно, пусть граф раскрашен. Вершины-литералы соединены с синей вершиной палитры, поэтому из каждой пары одна красная, а другая зелёная. Поскольку каждый гаджет правильно раскрашен, он не может быть подключён к трём красным вершинам. Значит, если сделать истинными литералы, соответствующие которым вершины покрашены в зелёный, то в каждом дизъюнкте будет хотя юы один истинный, т.е. формула выполнима. □

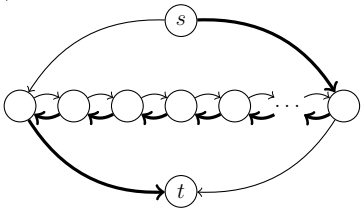
4.3 Пути и циклы в графе

Напомним, что эйлеровым путём называется путь, проходящий по одному разу по каждому ребру, а гамильтоновым — проходящий по одному разу через каждую вершину. Для эйлеровости существует простой критерий: граф должен быть связен, а степени всех вершин, кроме, быть может, двух, чётны. Обе характеристики можно проверить за полиномиальное время. А вот для гамильтоновости простого критерия нет. Более того, задача о гамильтоновом пути является **NP**-полной. Сначала рассмотрим ориентированный случай, затем перейдём к неориентированному.

Теорема 15. *Язык* DHAMPATH $= \{(G, s, t) \mid$ *в ориентированном графе* G *есть ориентированный путь из* s *в* t*, проходящий ровно один раз через каждую вершину*$\}$ *является* **NP***-полным.*

Доказательство. Сведём 3SAT к DHAMPATH. Ключевым инструментом будет такой гаджет: 

. Ясно, что его можно обойти двумя способами: зигзагом 

и загзигом 

. Мы возьмём по одному такому гаджету для каждой переменной из формулы и соединим их в одну большую цепочку. Число промежуточных вершин в каждом гаджете возьмём равным $3m+1$, где m — число дизъюнктов в 3-КНФ. Из этих вершина 1-ая, 4-ая, 7-ая, ..., $(3m+1)$-ая точно больше ни с чем не соединены, а остальные могут быть соединены с вершинами для дизъюнктов, которые мы добавим по одной для каждого. Если переменная входит в дизъюнкт C_i без отрицания, то добавим рёбра из $(3i-1)$-ой вершины в вершину для конъюнкта, а оттуда в $3i$-ую. Если же переменная входит с отрицанием, то наоборот: из $3i$-ой вершины в вершину для конъюнкта, а оттуда в $(3i-1)$-ую.

Покажем, что формула выполнима тогда и только тогда, когда в построенном графе есть гамильтонов путь из начальной вершины первого гаджета цепочки в конечную вершину последнего. По выполняющему набору путь строится очень просто: гаджеты,

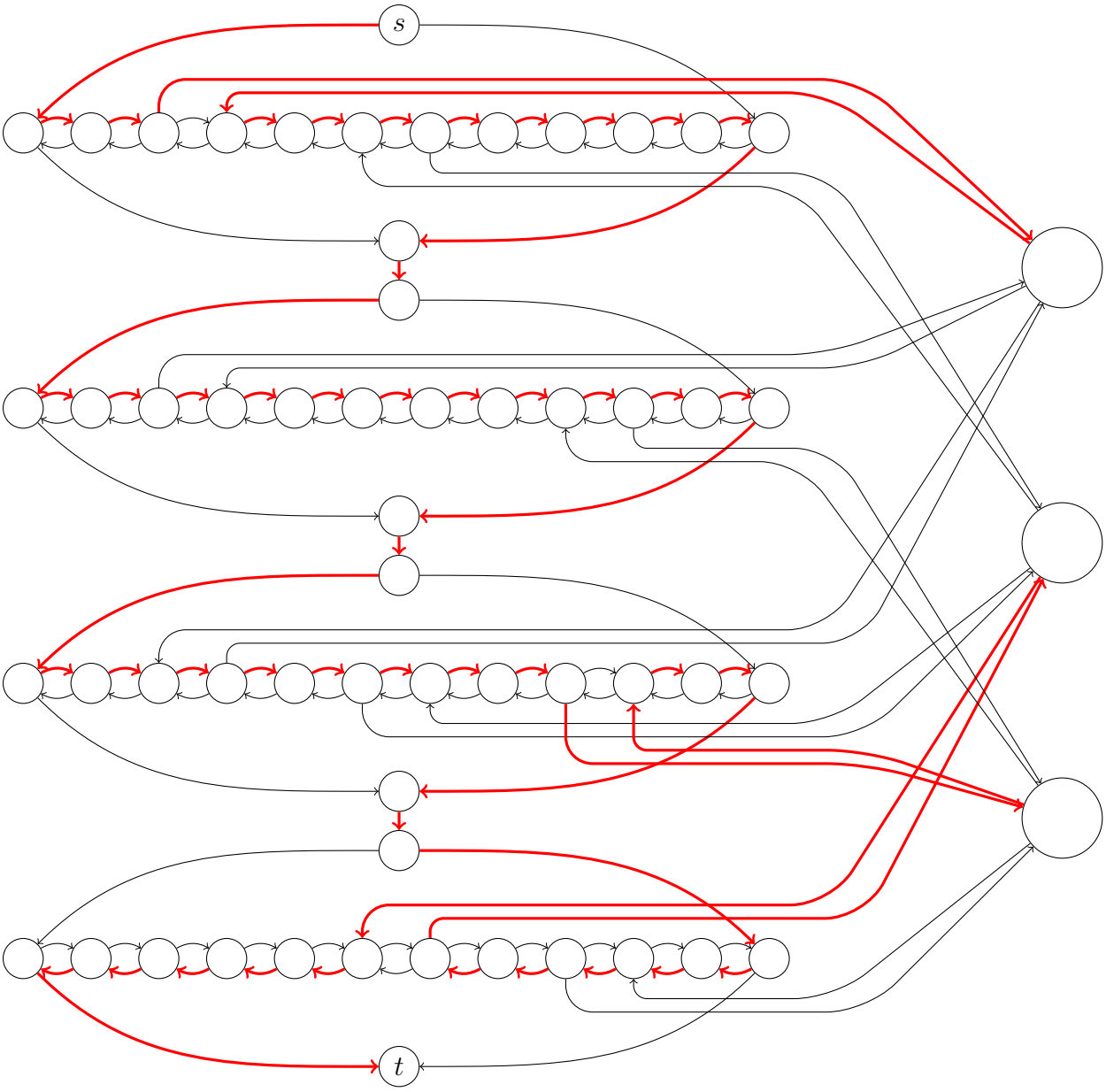
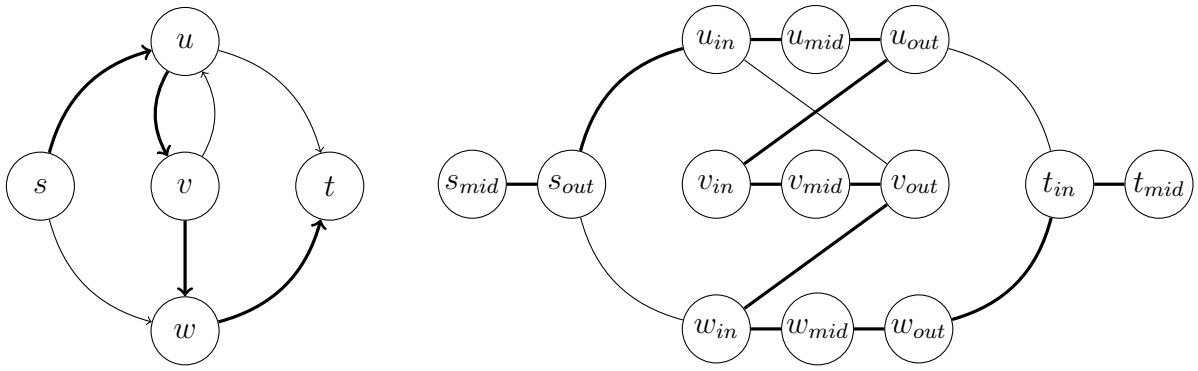


Рис. 3: Сводимость 3SAT к DHAMPATH на примере формулы $(p \vee q \vee \neg r) \wedge (\neg p \vee r \vee \neg t) \wedge (\neg q \vee r \vee t)$ (конечные и начальные вершины гаджетов можно склеить, но мы их рисуем отдельно для лучшей читаемости). Гамильтонов путь выделен жирными дугами и красным цветом.

соответствующие истинным переменным, путь будет проходить зигзагом, а гаджеты, соответствующие ложным, — загзигом. Когда путь проходит гаджет переменной, которая делает истинным некоторый дизъюнкт, он отклоняется по дороге в вершину, соответствующую этому литералу, но тут же возвращается обратно. Поскольку все дизъюнкты истинны, все вершины будут посещены. Построение графа и гамильтонова пути в нём проиллюстрировано на рис. 3.

Осталось доказать, что любой гамильтонов путь будет иметь именно такой вид.



(а) Исходный граф и путь в нём (б) Преобразованный граф и путь в нём

Рис. 4: Пример построения неориентированного графа по ориентированному с сохранением гамильтоновости

Посетив вершину, соответствующую некоторому дизъюнкту, путь мог бы вернуться в какой-то другой гаджет. Однако в таком случае в оставленном гаджете осталась бы вершина, которую заведомо нельзя посетить: именно для этого мы оставляли каждую третью вершину ни с чем более не связанной. Значит, путь уже не может быть гамильтоновым, что и требовалось. Таким образом, по любому гамильтонову пути можно восстановить выполняющий набор. □

Теперь перейдём к случаю неориентированного графа.

Теорема 16. *Язык* UHAMPATH $= \{(G, s, t) \mid$ *в неориентированном графе* G *есть путь из* s *в* t*, проходящий ровно один раз через каждую вершину*$\}$ *является* **NP***-полным.*

Доказательство. Сведём DHAMPATH к UHAMPATH. Проведём такое преобразование: каждую вершину v, кроме s и t, превратим в 3 вершины v_{in}, v_{mid} и v_{out}, соединённые в цепочку: v_{in}—v_{mid}—v_{out}. Аналогично s и t превратим в пары соединённых вершин s_{mid}—s_{out} и t_{in}—t_{mid}. Для каждого ребра (u, v) исходного графа проведём ребро (u_{out}, v_{in}) в новом. Стартовой и конечной вершинами будем считать s_{mid} и t_{mid}.

По гамильтонову пути в исходном графе легко получить гамильтонов путь в преобразованном. Достаточно вместо каждого ребра (u, v) из исходного пути взять цепочку вершин $(u_{mid}, u_{out}, v_{in}, v_{mid})$. Или, иначе говоря, первую вершину пути заменить на пару вершин (s_{mid}, s_{out}), каждую промежуточную вершину w заменить на три последовательные вершины $(w_{in}, w_{mid}, w_{out})$, а последнюю опять на пару (t_{in}, t_{mid}). Пример показан на рис. 4. Теперь покажем, что и по гамильтонову пути в преобразованном графе можно восстановить гамильтонов путь в исходном. Для этого достаточно заметить, что вершины обязательно идут тройками вида $(w_{in}, w_{mid}, w_{out})$. Это доказывается по индукции. База: из s_{mid} пусть идёт в s_{out}, поскольку других вариантов нет. Переход: пусть до сих пор все вершины шли тройками. Из очередной out-вершины можно попасть только в in-вершины, причём из пока не задействованных троек. Если дальше путь пойдёт не в соответствующую mid-вершину, то эта вершина никогда не сможет быть посещена без повторного посещения текущей in-вершины, а значит путь не будет гамильтоновым. Значит, в гамильтоновом пути после in-вершины будет mid-вершина, а затем и

out-вершина, поскольку других вариантов нет. Значит, снова получилась целая тройка, что и требовалось. □

Небольшой модификацией можно получить **NP**-полноту и задач о гамильтоновом *цикле* в ориентированном и неориентированном варианте. Ещё одна известная задача о путях в графе — *задача коммивояжёра* (traveling salesman problem). В этом случае дан не просто граф, а граф с весами, т.е. неотрицательными числами на каждом ребре. *Путём коммивояжёра* называется путь минимального суммарного веса, проходящий хотя бы один раз через каждую вершину. Вершины интерпретируются как города, рёбра — как дороги между городами, веса — как длины дорог. Коммивояжёр должен посетить все города, проехав как можно меньше.

Теорема 17. *Язык* $\mathsf{TSP} = \{(G, \mathbf{w}, s, t, l) \mid$ *во взвешенном графе* $(G, \mathbf{w})$ *есть путь коммивояжёра из* s *в* t *длины не больше* $l\}$ *является* **NP***-полным.*

Доказательство. Сведём UHAMPATH к TSP. Достаточно сопоставить всем рёбрам единичные веса и взять $l = n - 1$, где n — число вершин. (В случае $s = t$ нужно взять $l = n$). Действительно, путь из $n-1$ ребра, проходящий через все вершины, обязан быть гамильтоновым. □

Иногда при формулировке задачи коммивояжёра требуют, чтобы граф обязательно был полным. Ясно, что это не влияет на суть задачи: все отсутствующие рёбра можно провести, но сопоставить им очень большие веса, так чтобы оптимальный путь через них заведомо не проходил. В рассуждении об **NP**-полноте можно взять все рёбра, отсутствующие в исходном графе, с весом больше 1: на вывод это не повлияет. При этом веса рёбер образуют метрику, т.е. для них выполняется неравенство треугольника, соответственно, *метрическая* задача коммивояжёра также будет **NP**-полной. Можно показать, что даже евклидова задача коммивояжёра, в которой все вершины лежат в евклидовом пространстве, а расстояния получаются при помощи обычной евклидовой метрики, является **NP**-полной. Конструкция работает уже в двумерном пространстве, т.е. на декартовой плоскости.

4.4 Покрытие множествами и задача о рюкзаке

Рассмотрим такую задачу о покрытии множествами: $\mathsf{EXACTSETCOVER} = \{(n, S_1, \ldots, S_m) \mid$ можно выбрать непересекающиеся множества $S_{i_1}, \ldots, S_{i_k}$, в объединении дающие все числа от 1 до $n\}$. Иными словами, все числа от 1 до n должны быть покрыты ровно по одному разу.

Теорема 18. *Задача* EXACTSETCOVER *является* **NP***-полной.*

Доказательство. Сведём 3SAT к EXACTSETCOVER. Ясно, что элементы множеств можно называть как угодно. Мы заведём для каждой переменной три возможных элемента p, p_0 и p_1, а для каждого дизъюнкта — четыре элемента C, C_1, C_2, C_3. Множества будут трёх видов. Во-первых, для каждой переменной возьмём множества $\{p, p_0\}$ и $\{p, p_1\}$. Во-вторых, для каждого дизъюнкта возьмём множества $\{C\}$, $\{C, C_1\}$, $\{C, C_2\}$, $\{C, C_3\}$, $\{C, C_1, C_2\}$, $\{C, C_2, C_3\}$ и $\{C, C_1, C_3\}$. Наконец, в-третьих, для каждой переменной заведём множества $\{p_0, C_j, \ldots\}$ и $\{p_1, D_j, \ldots\}$, где в первое множество включены элементы

для всех дизъюнктов C, содержащих литерал p на j-ом месте, а во второе — для дизъюнктов D, содержащих литерал $\neg p$ на j-ом месте.

Пусть есть выполняющий набор. Если переменная p в этом наборе истинна, возьмём множество $\{p, p_1\}$, если ложна, то $\{p, p_0\}$. Далее, в первом случае возьмём $\{p_0, C_j, \dots\}$, во-втором — $\{p_1, D_j, \dots\}$. Поскольку в каждм дизъюнкте есть хотя бы один истинный литерал, мы уже покрыли хотя бы один из элементов C_1, C_2, C_3. Добавляем в покрытие множество из C и непокрытых элементов C_i и получаем покрытие всех элементов.

Обратно, пусть имеется покрытие. Для каждой переменной p в него входит ровно одно из множеств $\{p, p_0\}$ и $\{p, p_1\}$, а для каждого дизъюнкта C — ровно одно из множеств $\{C\}$, $\{C, C_1\}$, $\{C, C_2\}$, $\{C, C_3\}$, $\{C, C_1, C_2\}$, $\{C, C_2, C_3\}$ и $\{C, C_1, C_3\}$. Выбор первого множества задаст истинность p в выполняющем наборе. Оставшийся элемент из p_0 и p_1 должен быть покрыт множеством вида $\{p_i, C_j, \dots\}$. Поскольку покрыты все элементы, для каждого дизъюнкта хотя бы один элемент C_j покрывается множеством вида $\{p_i, C_j, \dots\}$. По построению это означает, что соответствующий литерал в дизъюнкте истинен. Значит, все дизъюнкты истинны, т.е. набор выполняющий, что и требовалось. □

Перейдём к задаче о рюкзаке и её частному случаю $\mathsf{SUBSET\text{-}SUM} = \{(k_1, \dots, k_m, N) \mid \exists \alpha \in \{0,1\}^m \sum \alpha_i k_i = N\}$.

Теорема 19. *Задача* SUBSET-SUM *является* **NP***-полной.*

Доказательство. Сведём EXACTSETCOVER к SUBSET-SUM. В качестве N возьмём число из n единиц в $(m+1)$-ичной записи (это займёт $O(n \log m)$ битов). Число k_i также будет записано в $(m+1)$-ичной записи: в разрядах, номера которых лежат в S_i, будут стоять единицы, в остальных нули.

Если существует покрытие, то сумма соответствующих k_i даёт в точности N: в каждом разряде ровно один раз встретится единица.

Пусть теперь $\sum \alpha_i k_i = N$. В силу того, что основание системы счисления больше числа слагаемых, а в разрядах стоят только нули и единицы, переносов при сложении не будет. Значит, для каждого разряда ровно в одном слагаемом стоит единица. Значит, множества S_i, для которых $\alpha_i = 1$, образуют покрытие. □

4.5 Вариации с выполнимостью

Многие частные случаи задачи о выполнимости булевых формул, а также многие смежные задачи являются **NP**-полными. Приведём несколько примеров.

Теорема 20. *Задача* 3SAT *остаётся* **NP***-полной, даже если ограничиться формулами, в которых каждая переменная входит не более 3 раз, а каждый литерал — не более двух раз.*

Доказательство. Идея проста: каждую формулу преобразуем к указанному виду с сохранением выполнимости. Для этого заменим каждое вхождение каждой переменной на новую переменнную. Если новые переменные, соответствующие p, суть $p_1, \dots, p_k$, то добавии в конъюнкцию формулу $(p_1 \vee \neg p_2) \wedge (p_2 \vee \neg p_3) \wedge \dots \wedge (p_k \vee \neg p_1)$. Условие на число

вхождений, очевидно, будет соблюдено. Покажем, что выполнимость сохранилась. Действительно, выполнимость добавленной для переменной p формулы эквивалентна тому, что все переменные p_i принимают одно и то же значение. Значит, выполняющий набор новой формулы определяет набор значений старых переменных, при этом он также будет выполняющим. Обратное тоже верно, значит сводимость установлена. □

Теорема 21. *Задача* NAESAT $= \{\varphi \mid$ *для формулы* φ *в форме 3-КНФ есть набор значений, при котором в каждом дизъюнкте есть как истинный, так и ложный литералы*$\}$ *является* **NP***-полной.*

Доказательство. Сведём 3SAT к NAESAT. Для каждого дизъюнкта $(\alpha_i \vee \beta_i \vee \gamma_i)$ введём два новых дизъюнкта: $(\alpha_i \vee \beta_i \vee z_i)$ и $(\neg z_i \vee \gamma_i \vee t_i)$, где переменная z_i своя для каждого дизъюнкта, а переменная u едина для всех.

Пусть исходная формула была выполнима. Рассмотрим набор, в котором все исходные переменные те же самые, а $u = 0$. Если $\alpha_i = 1$ или $\beta_i = 1$, положим $z_i = 0$. В таком случае в обоих дизъюнктах будет по истинному и ложному литералу. Если же $\alpha_i = \beta_i = 0$, положим $z_i = 1$. Поскольку в этом случае обязательно $\gamma_i = 1$, вновь в обоих дизъюнктах будет по истинному и ложному литералу.

Теперь обратно, пусть в новой формуле есть набор, при котором в каждой скобке есть истинный и ложный литералы. Если в этом наборе $u = 0$, то хотя бы один из литералов α_i, β_i и γ_i должен быть истинным, т.е. соответствующий дизъюнкт выполним. Если же в этом наборе $u = 1$, то рассмотрим другой набор, в котором все переменные принимают противоположные значения. Если в исходном наборе в каждом дизъюнкте были истинный и ложный литералы, то и в новом тоже так будет. Значит, этот случай свёлся к предыдущему, и сводимость доказана. □

Можно показать, что задача о выполнимости 2-КНФ решается за полиномиальное время. Но если расширить задачу в другом направлении, а именно максимизировать число выполненных дизъюнктов в 2-КНФ, она снова станет **NP**-полной.

Теорема 22. *Задача* MAX2SAT $= \{(\varphi, k) \mid \varphi$ *— формула в виде 2-КНФ, для которой существует набор значений, при котором выполнены* k *дизъюнктов*$\}$ *является* **NP***-полной.*

Доказательство. Сведём 3SAT к MAX2SAT. Рассмотрим следующие 10 дизъюнктов: p, q, r, s, $\neg p \vee \neg q$, $\neg p \vee \neg r$, $\neg q \vee \neg r$, $p \vee \neg s$, $q \vee \neg s$, $r \vee \neg s$. Заметим, что если p, q и r одновременно истинны, то $\neg p \vee \neg q$, $\neg p \vee \neg r$ и $\neg q \vee \neg r$ ложны, при $s = 1$ истинны ровно 7 из 10 дизъюнктов. Если $p = q = 1$, $r = 0$, то ложны r, $\neg p \vee \neg q$ и один из дизъюнктов s, $r \vee \neg s$, т.е. снова истинны 7 из 10. Если $p = 1$, $q = r = 0$, то ложны q, r и либо s, либо $q \vee \neg s$ и $r \vee \neg s$, снова истинны не больше 7 из 10. И только если $p = q = r = 0$, будут истинны не больше 6 из 10 дизъюнктов. Сводимость построим так: каждый дизъюнкт $(p \vee q \vee r)$ заменим на 10 вышеуказанных дизъюнктов, взяв новую переменную s. В качестве k возьмём число $7m$, где m — общее число дизъюнктов. Из вышесказанного следует, что если все дизъюнкты выполнимы, то в каждой группе можно выполнить по 7 дизъюнктов, всего $7m$. А если все одновременно не выполнимы, то хотя бы в одной группе будут выполнены 6 дизъюнктов, а в остальных не больше 7, в итоге $7m$ не наберётся. □

4.6 Линейное и квадратичное программирование

Многие практические задачи можно сформулировать как задачи линейного программирования (ЛП), т.е. оптимизации некоторой линейной функции при условии выполнения некоторых линейных уравнений и неравенств. На практике используют экспоненциальный в худшем случае, но обычно быстрый симплекс-метод. При этом существуют полиномиальные в любом случае алгоритмы, например, метод эллипсоидов Хачияна. Как обычно, можно превратить задачу оптимизации в задачу распознавания, поставив вопрос, может ли целевая функция быть меньше некоторого порога.

Ситуация сильно осложняется, если требуется найти не произвольное решение, а целочисленное (в таком случае задачу называют задачей целочисленного линейного программирования — ЦЛП). Даже если рассматривать лишь системы, в которых все коэффициенты есть нули или единицы, задача уже будет **NP**-трудной. Практически любую задачу можно переформулировать как задачу ЦЛП. Например, задача о вершинном покрытии формулируется так: существуют ли такие целые $x_1, \ldots, x_n$, что $\sum x_i \leqslant k$, но $x_i + x_j \geqslant 1$ для всех рёбер (i, j). Однако в даннм случае неочевиден вопрос о том, лежит ли язык в классе **NP**. Понятно, что если решение есть, то его можно проверить, но при этом решение должно иметь короткую запись. Известно, что любую систему линейных неравенств при помощи добавления новых переменных можно привести к виду $Ax = b, x \geqslant 0$. Именно для такой системы мы докажем **NP**-полноту.

Теорема 23. *Задача* $\mathsf{INTPROG} = \{(A, \mathbf{b}) \in \mathbb{Z}^{nm} \times \mathbb{Z}^n \mid \exists \mathbf{x} \in \mathbb{N}^m \ A\mathbf{x} = \mathbf{b}\}$ *является* **NP**-*полной.*

Доказательство. **NP**-трудность мы фактически уже доказали. Для полноты изложения приведём сводимость к задаче ровно такого вида. В этот раз будем сводить задачу EXACTSETCOVER. Пусть дана система множеств $(S_1, \ldots, S_m)$. В столбцах матрицы A запишем характеристические функции этих множеств, а вектор $\mathbf{b}$ составим из единиц. Если покрытие есть, то возьмём вектор $\mathbf{x}$, такой что $x_i = 1$, если S_i входит в покрытие, и $x_i = 0$, если S_i не входит. Поскольку каждый элемент входит ровно в одно множество покрытия, в сумме получится как раз вектор из всех единиц. Обратно, пусть есть решение задачи ЦЛП. Если в нём есть хотя бы одно число больше 1, то и в итоговой сумме будет число больше 1, т.е. вектор из всех единиц получиться не может. Значит, все числа будут либо 0, либо 1, что и задаст покрытие.

Осталось доказать принадлежность к **NP**. Для этого мы докажем, что если задача ЦЛП имеет какое-то решение, то она имеет и решение, все элементы которого не превосходят $2^{\mathrm{poly}(n,m,\log a)}$, где a — максимальный элемент A и $\mathbf{b}$. В качестве точного значения границы возьмём $N = n(ma)^{2m+1}$. Введём также обозначение $M = (am)^m$. Предположим, что во всех натуральных решениях системы $A\mathbf{x} = \mathbf{b}$ хотя бы одна координата будет больше N. Из всех решений выберем одно, у которого сумма координат, превышающих M (а не N!), будет наименьшей. Без ограничения общности для этого $\mathbf{x}$ выполнено $x_1 > M, \ldots, x_k > M, x_{k+1} \leqslant M, \ldots, x_m \leqslant M$. Обозначим через $\mathbf{v}_1, \ldots, \mathbf{v}_k$ соответствующие столбцы матрицы A. Рассмотрим два случая:

а) Существует нетривиальная целочисленная линейная комбинация с коэффициентами от 0 до M, равная нулю: $\sum \lambda_i \mathbf{v}_i = 0$. В этом случае вектор $(x_1 - \lambda_1, \ldots, x_k -$

$\lambda_k, x_{k+1}, \ldots, x_m$) также неотрицателен и удовлетворяет уравнению, что противоречит минимальности исходного $\mathbf{x}$.

б) Такой линейной комбинации не существует. Для начала заметим, что в таком случае вообще никакой нетривиальной неотрицательной линейной комбинации не существует. Действительно, коэффициенты любой такой линейной комбинации находятся как решение некоторой системы линейных уравнений размерности не больше $k-1$ с коэффициентами не больше a. По правилу Крамера числа из этого решения будут дробями с числителем и знаменателем — определителями размера не больше $(k-1)\times(k-1)$. Каждый из этих определителей будет не превышать $(k-1)!\cdot a^{k-1} < m!\cdot a^m < M$. При этом все знаменатели одинаковы, поэтому была бы и целочисленная комбинация с коэффициентами меньше M. А раз такой нет, то нет никакой.

Далее, рассмотрим множество S всех неотрицательных линейных комбинаций векторов $\mathbf{v}_1, \ldots, \mathbf{v}_k$. Оно, очевидно, выпукло (и является конусом). При этом точка $\mathbf{0}$ находится на его границе, иначе была бы нетривиальная нулевая комбинация. По теореме об опорной гиперплоскости существует $\mathbf{h} \in \mathbb{R}^n$, такое что $\langle \mathbf{h}, \mathbf{v}_i \rangle \geqslant 0$ при всех i. Более того, все неравенства можно сделать строгими, иначе в множестве S были бы два противоположных вектора, и нетривиальная нулевая комбинация существовала бы.[3] И более того, можно выбрать $\mathbf{h}$ с коэффициентами по модулю не больше M, так чтобы $\langle \mathbf{h}, \mathbf{v}_i \rangle \geqslant 1$.[4]

Теперь заметим, что $\sum_{j=1}^k x_j \leqslant \sum_{j=1}^k \langle \mathbf{h}, \mathbf{v}_j \rangle x_j = \langle \mathbf{h}, \sum_{j=1}^k \mathbf{v}_j x_j \rangle = \langle \mathbf{h}, Ax - \sum_{j=k+1}^m \mathbf{v}_j x_j \rangle = \langle \mathbf{h}, b - \sum_{j=k+1}^m \mathbf{v}_j x_j \rangle$. Последнее выражение есть скалярное произведение двух n-мерных целочисленных векторов, координаты первого из которых не больше nM, а второго — не больше $a + a(m-1)M < (am)^{m+1}$. Значит, всё произведение меньше $n^2(am)^{2m+1} = N$. Однако в левой части стоит сумма чисел каждое из которых больше N. Получаем противоречие, значит этот случай также невозможен.

Таким образом, у какого-то решения все компоненты не больше N, что и требовалось. □

От линейного программирования перейдём к квадратичному. Под задачей квадратичного программирования будем понимать систему уравнений второго порядка от n переменных. Вначале рассмотрим задачу с булевыми коэффициентами.

---

[3] Подробнее: множество H таких векторов $\mathbf{h}$, таких что $\langle \mathbf{h}, \mathbf{v}_i \rangle \geqslant 0$, также является конусом, причём двойственным к S. Если все неравенства нельзя сделать строгими, то это конус неполной размерности, т.е. вложен в некоторую гиперплоскость. Если размерность конуса равна $n-1$, то два вектора, нормальные к этой гиперплоскости и противоположные по направлению, должны найтись среди векторов $\mathbf{v}_i$. В общем случае, если размерность равна $n-p$, среди $\mathbf{v}_i$ должны найтись p пар противоположных по направлению векторов, образующие ортогональное подпространство.

[4] А это делается так: вначале нужно найти целочисленные образующие конуса H. Это делается решением линейных систем из n уравнений с n неизвестными. Как мы уже знаем, все компоненты этих векторов будут меньше $(an)^n$. При этом можно считать, что $n \leqslant k < m$, иначе задача будет тривиальной. Из векторов можно выбрать k, в каждом из которых выполнено своё строгое неравенство $\langle \mathbf{h}, \mathbf{v}_i \rangle > 0$. Теперь нужно рассмотреть сумму этих векторов. У неё все компоненты будут меньше $k(an)^n < M$.

Теорема 24. *Задача* QUADEQ $= \{(A, \mathbf{b}) \in \{0,1\}^{n^2} \times \{0,1\}^n \mid \exists \mathbf{x} \in \{0,1\}^n \mathbf{x}^\mathsf{T} A\mathbf{x} = \mathbf{b}\}$ *является* **NP**-*полной.*

Доказательство. Мы сведём SAT к QUADEQ. Доказательство в целом повторяет доказательство утверждения 9. Отличие в том, что утверждения вида $p = q \vee s$ записываются не в 3-КНФ, а в виде многочленов. Поскольку с каждой стороны не больше двух переменных, получатся квадратичные многочлены. □

Наконец, рассмотрим задачу с произвольными коэффициентами:

Теорема 25. *Задача* REALQUADEQ $= \{(A, \mathbf{b}) \in \mathbb{Z}^{n^2} \times \mathbb{Z}^n \mid \exists \mathbf{x} \in \mathbb{R}^n \mathbf{x}^\mathsf{T} A\mathbf{x} = \mathbf{b}\}$ *является* **NP**-*полной.*

Доказательство. Слегка изменим предыдущую конструкцию, чтобы получилась сводимость к REALQUADEQ. Для этого потребуется две вещи: во-первых, для каждой переменной добавим уравнение $x^2 = x$, что ограничит её значение нулями и единицами. Во-вторых, многочлены для записи булевых функций запишем в действительнозначной арифметике, а не в булевой: $\neg x = 1 - x$, $x \wedge y = xy$, $x \vee y = x + y - xy$. □

5 Задачи поиска

С каждой задачей распознавания из **NP** связана задача поиска: по входу x найти сертификат s, такой что $V(x, s) = 1$, либо указать, что таких сертификатов нет. Ясно, что задача поиска сводится к задаче распознавания: достаточно посмотреть, найден сертификат или нет. Удивительно, что верна и обратная сводимость. Разумеется, это не может быть сводимость по Карпу, которая определена только для задач распознавания. Это будет сводимость по Куку: если использовать решатель задачи распознавания как оракул, то можно решить и задачу поиска. Формализацию вычислений с оракулом мы изучим на следующей лекции. Пока что будем представлять, что алгоритм за 1 шаг может получить ответ на требуемый вопрос.

Теорема 26. *Пусть $L \in$* **NP**. *Пусть машина может за один шаг получать ответ, лежит ли слово x в языке L. Тогда существует полиномиальный алгоритм, решающий задачу поиска для L.*

Доказательство. Заметим, что сводимость, описанная в доказательстве теоремы Кука–Левина, на самом деле была сводимостью не просто по Карпу, а *по Левину*, т.е. по выполняющему набору для полученной формулы можно было восстановить сертификат для исходного входа. Действительно, биты сертификата попросту были равны некоторым битам набора. Поэтому текущую теорему достаточно доказать только для задачи о выполнимости.

Пусть дана формула φ, зависящая от переменных $p_1, \ldots, p_n$. Вначале можно проверить выполнимость φ (при помощи оракула), и если формула невыполнима, такой ответ и вернуть. Если же формула выполнима, рассмотрим формулы φ_1' и φ_1'', полученные из φ фиксацией значений $p_1 = 0$ и $p_1 = 1$ соответственно. Хотя бы одна из них будет выполнима, какая именно, можно найти при помощи двух запросов к оракулу. Обозначим выполнимую формулу через φ_1 и применим к ней ту же процедуру рекурсивно.

Так будут определяться значения всех переменных, пока не останется только одна. Это будет база рекурсии: значение последней переменной можно найти непосредственной подстановкой обоих вариантов. □

Для некоторых задач из **NP** можно провести подобное рассуждение о «самосводимости» непосредственно.

6 Как решать NP-полные задачи на практике

Многие вычислительные задачи, возникающие на практике, являются **NP**-полными. Поскольку рассчитывать на быстрые алгоритмы для решения таких задач не приходится, исследователи ищут другие методы:

- **Решение для частного случая.** Часто бывает, что в частном случае полиномиальный алгоритм существует. Например, общего решения для задачи о выполнимости нет, но есть полиномиальный алгоритм для задачи о выполнимости 2-КНФ. Поиск частного случая, под который подпадут интересующие задачи, и для которого есть полиномиальный алгоритм, может быть весьма нетривиальной задачей. При этом многие важные частные случаи остаются **NP**-полными, такие как SUBSET-SUM для задачи о рюкзаке или задача коммивояжёра на евклидовой плоскости.

- **Приближённое решение**. В задачах оптимизации обычно достаточно найти не оптимальное решение, а просто достаточно хорошее, например в пределах нескольких процентах от оптимального. Задача оптимизации ставится так: имеется полиномиально вычислимая функция $F\colon \{0,1\}^* \times \{0,1\}^* \to \mathbb{N}$. Требуется по входу x найти параметр s, при котором значение $F(x,s)$ минимально (или максимально). Если оптимальное значение равно OPT, то решение с точностью ε означает поиска такого s, что $F(x,s) < (1+\varepsilon)OPT$ (Для задачи максимизации $F(x,s) > (1-\varepsilon)OPT$). Можно также ставить требование иначе: найти число в полуинтервале $[OPT, (1+\varepsilon)OPT)$ (для задачи максимизации — в $((1-\varepsilon)OPT, OPT]$). Как правило, задачи оптимизации даже не лежат в **NP**: непонятно, как проверить оптимальность найденного решения. Но многие из них связаны с **NP**-полными задачами и потому **NP**-трудны: например, поиск в графе кликового числа, числа независимости, хроматического числа, поиск во взвешенном графе пути коммивояжёра, поиск для 3-КНФ набора, делающего истинным максимальное число дизъюнктов, выполнение наибольшего числа уравнений в задаче целочисленного линейного программирования и т.д.

  В различных конкретных задачах получены совершенно различные результаты для приближённых задач оптимизации. Для одних задач построены полиномиальные приближённые алгоритмы, так называемые **PTAS** (polynomial-time approximation schemes — аппроксисмирующие схемы, работающие за полиномиальное время). Примером такой задачи является задача коммивояжёра на евклидовой плоскости. Если алгоритм работает полиномиальное время не только от входа x,

но и от параметра приближения $\frac{1}{\varepsilon}$, то аппроксимирующая схема называется полностью полиномиальной (fully polynomial-time, **FPTAS**). Примером такой задачи является задача о рюкзаке. Для других задач есть алгоритмы для одних приближений, но поиск лучшего приближения уже является **NP**-трудной задачей.[5] Примером такой задачи является задача о максимизации числа выполненных дизъюнктов в 3-КНФ: легко построить алгоритм для $\varepsilon = \frac{1}{8}$, а вот более точное решение уже **NP**-трудно (в этом состоит утвреждение знаменитой **PCP**-теоремы). Другой пример такой задачи — задача коммивояжёра в метрическом пространстве. Ну а для некоторых задач не существует приближённого алгоритма ни с какой степенью точности, например для задачи о клике или для задачи коммивояжёра с произвольными весами.

Построение приближённых алгоритмов, а также доказательство их отсутствия в тех или иных предположениях — активно разивающаяся область, важная и для теории, и для практики.

- **Эвристики.** Для некоторых задач существуют алгоритмы, которые долго работают в худшем случае, но быстро дают ответ в среднем, или в «типичном» случае. Особенно много их в задачах оптимизации: метод ветвей и границ, генетические алгоритмы, метод пчелиного роя, метод муравьиной колонии, метод имитации отжига и т.д. Зачастую про такие алгоритмы нет строгих теорем, но на практике оказывается, что они работают неплохо. Разумеется, часто они также ищут не точное, а приближённое решение. Но для некоторых задач они хорошо ищут и точное решение. Например, задача коммивояжёра успешно решается для входов с тысячами городов.

[5] **NP**-трудность тут понимается в таком смысле: описывается процедура, которая по слову из **NP**-полного языка строит задачу с оптимальным значением OPT, в по слову не из языка — с оптимальным значением не меньше $(1 + \varepsilon)OPT$. Приближённый алгоритм с точностью выше ε позволит отделить один случай от другого.