Дискретная математика

Сложность и NP-полнота

Константин Чухарев

Осень 2026

Сложность и NP-полнота

> *Всякий, кто способен оценить симфонию, был бы Моцартом.*
>
> ***— Скотт Аронсон***

Коммивояжёр

Задача коммивояжёра: даны n городов и попарные расстояния, найти кратчайший маршрут через каждый город ровно один раз. Полный перебор просматривает все $n!$ маршрутов. Алгоритм корректен и завершается для любого n. Для $n = 100$ число маршрутов превышает число атомов в наблюдаемой Вселенной. Алгоритм существует, но ответа мы не дождёмся.

Пример (Родственники коммивояжёра)

- Расписание: можно ли выполнить все работы к сроку?
- Карта: можно ли покрасить страны тремя цветами так, чтобы соседние различались?
- Сумма подмножества: есть ли подмножество чисел с заданной суммой?

Для каждой из этих задач известен только полный перебор. Полвека поисков не принесли ничего существенно лучшего.

Что значит «быстро»

Интуиция: проблема в экспоненциальном переборе. Быстро — это за полиномиальное время: число шагов растёт как n^k для некоторой константы k.

Полиномиальный рост n^2 — повседневность. Экспоненциальный рост 2^n при умеренных n недостижим. Одна развилка — две дороги, двадцать — больше миллиона, триста — больше, чем атомов во Вселенной.

Класс P

Определение 1 (Класс P)

Языки, разрешимые детерминированной машиной Тьюринга за полиномиальное время:

$$P = \bigcup_{k \geq 1} \mathrm{TIME}\left(n^k\right).$$

Класс P устойчив к изменению модели вычислений: одноленточная и многоленточная машины Тьюринга и RAM-машина меняют время работы не более чем на полиномиальный фактор, поэтому принадлежность классу P при этом сохраняется.

Почему именно полином

Выбор полинома — не произвол.

- Полиномы замкнуты относительно композиции: алгоритм с подпрограммой остаётся в классе.
- Класс $O(n \log n)$ таким свойством не обладает. Композиция двух его алгоритмов даёт $O(n \log^2 n)$.
- Свойство «быть полиномиальным» не зависит от модели вычислений.
- Полином — минимальный класс, одновременно замкнутый относительно композиции и устойчивый к смене модели.

Оговорки:

- Алгоритм за $O(n^{100})$ формально в P, но на практике неприменим.
- Экспоненциальный алгоритм за $O(1.0001^n)$ может работать быстро для умеренных n.

Задачи из P

Пример (Задачи из P)

- Сортировка, поиск в ширину и глубину.
- Кратчайшие пути (Дейкстра), минимальное остовное дерево (Прим, Крускал).
- 2-раскраска (проверка двудольности), эйлеров цикл, проверка планарности.
- Проверка простоты (алгоритм AKS, 2002), линейное программирование (эллипсоиды).

Наше знание о P не статично: линейное программирование попало в P в 1979 году, проверка простоты — в 2002.

Класс NP

NP расшифровывается как Nondeterministic Polynomial time. Вопреки распространённой ошибке, это не Not Polynomial.

Определение 2 (Класс NP)

Языки L, для которых есть полиномиальный верификатор V и полином p:

$$w \in L \;\leftrightarrow\; \exists c : |c| \leq p(|w|) \text{ и } V(w, c) \text{ принимает.}$$

Неформально: «угадай сертификат, проверь за полиномиальное время».

Эквивалентное определение: языки, разрешимые недетерминированной машиной Тьюринга за полиномиальное время.

Задачи из NP

Пример (Сертификаты)

- SAT: сертификат — выполняющее присваивание переменных.
- Гамильтонов цикл: сертификат — сам цикл.
- Вершинное покрытие размера $\leq k$: сертификат — множество вершин.
- Клика размера $\geq k$: сертификат — k попарно смежных вершин.
- Сумма подмножества: сертификат — подмножество чисел.

Из быта: судоку ищут часами, а предъявленную расстановку проверяют за минуты.

P вложен в NP

$$P \subseteq \mathrm{NP}.$$

Если задача решается за полиномиальное время, в качестве сертификата подойдёт пустая строка: верификатор просто выполнит алгоритм решения. Обратное включение — вопрос тысячелетия.

Проблема P против NP

Совпадает ли класс задач, решаемых быстро, с классом задач, проверяемых быстро:

$$P = \mathrm{NP}?$$

- Если $\mathrm{P} = \mathrm{NP}$, поиск решения принципиально не сложнее проверки.
- Если $\mathrm{P} \neq \mathrm{NP}$, есть задачи, где проверка неизмеримо легче нахождения.
- Большинство теоретиков полагает, что $\mathrm{P} \neq \mathrm{NP}$: опрос 2019 года дал около 80% сторонников неравенства.

Вопрос открыт полвека и остаётся главной нерешённой проблемой computer science.

Оракулы и релятивизация

Оракул — бесплатная подпрограмма: машина за один шаг узнаёт у оракула, принадлежит ли строка его языку. С оракулом A получаем классы P^A и NP^A — те же определения, но машины могут звать оракул.

Бейкер, Гилл и Соловей (1975) показали, что диагонализация не решит вопрос P против NP. Существуют оракулы A и B, для которых $\mathrm{P}^A = \mathrm{NP}^A$, но $\mathrm{P}^B \neq \mathrm{NP}^B$. Диагонализация «не чувствует» оракула: её аргумент переносится в любой оракульный мир и потому не может дать разные ответы для A и B. Для вопроса P против NP нужна техника, выходящая за пределы диагонализации.

Полиномиальные сведения

Сравнивать задачи по трудности позволяют сведения.

Интуиция из быта: в игре «числовой скрэббл» двое выбирают числа от 1 до 9, и выигрывает тот, кто первым соберёт три числа с суммой 15. Если расположить числа в магическом квадрате, игра превращается в крестики-нолики: стратегия переносится без изменений.

Определение 3 (Полиномиальное сведение)

Язык A полиномиально сводится к языку B, если существует вычислимая за полиномиальное время функция f. При этом $w \in A \;\leftrightarrow\; f(w) \in B$.

Сведение $A \underset{p}{\leq} B$ означает: если бы мы умели эффективно решать B, то умели бы и A.

Что даёт сведение

Сведение — вычислимая за полиномиальное время функция, и размер её выхода тоже полиномиален. Запись результата занимает время, сравнимое с его длиной: сведение не раздувает экземпляр.

Замкнутость относительно сведений

Теорема 1 (Замкнутость)

Если $A \leq_p B$ и $B \in \mathrm{P}$, то $A \in \mathrm{P}$. Если $A \leq_p B$ и $B \in \mathrm{NP}$, то $A \in \mathrm{NP}$.

Доказательство (Доказательство)

Алгоритм для A: вычислить $f(w)$ за полиномиальное время и подать результат алгоритму для B. Композиция полиномов полиномиальна. Для NP: верификатор $V_A(w, c) = V_B(f(w), c)$.

NP-трудные и NP-полные

Определение 4 (NP-трудная задача)

Задача B NP-трудна, если любая задача из NP сводится к ней. Формально: $\forall A \in \text{NP} : A \underset{p}{\leq} B$.

Определение 5 (NP-полная задача)

Задача B NP-полна, если $B \in \text{NP}$ и B NP-трудна.

NP-трудная задача может лежать вне NP: проблема остановки NP-трудна, но неразрешима. NP-полнота — «самая трудная в NP», а NP-трудность — «не легче любой NP-задачи».

Значимость NP-полноты

Теорема 2 (Единый класс эквивалентности)

Если хотя бы одна NP-полная задача лежит в P, то $\mathrm{P} = \mathrm{NP}$. Если ни одна NP-полная задача не лежит в P, то $\mathrm{P} \neq \mathrm{NP}$.

Доказательство (Доказательство)

Пусть B NP-полна и $B \in \mathrm{P}$. Для любой $A \in \mathrm{NP}$ имеем $A \underset{p}{\leq} B$, и по замкнутости $A \in \mathrm{P}$. Значит, $\mathrm{NP} \subseteq \mathrm{P}$, а обратное включение всегда верно.

Все NP-полные задачи образуют единый класс: либо все они решаются быстро, либо ни одна.

Теорема Кука–Левина

NP-полные задачи существуют: их предъявляет теорема Кука–Левина.

Теорема 3 (Теорема Кука–Левина)

SAT NP-полна.

Доказательство (Эскиз доказательства)

SAT в NP: сертификат — выполняющее присваивание, проверка — подстановка в формулу за линейное время.

NP-трудность: пусть $L \in \mathrm{NP}$ и недетерминированная машина M принимает L за время $p(n)$. Вычисление M на входе w — таблица размера $p(n) \times p(n)$. Локальность переходов позволяет закодировать корректность таблицы конъюнкцией дизъюнктов фиксированного размера. Формула φ выполнима тогда и только тогда, когда M принимает w. Полное построение переменных — в конспекте.

Кодирование таблицы вычисления

Дизъюнкты кодируют начальную конфигурацию, корректность переходов и достижение принимающего состояния. Формула φ выполнима тогда и только тогда, когда M принимает w. Размер формулы: $O(p(|w|)^2)$.

Почему Кук–Левин открыл дверь

Таблица вычисления — матрица $p(n) \times p(n)$, где каждая ячейка зависит только от трёх ячеек предыдущей строки. Локальность переходов машины делает кодирование возможным: корректность вычисления — конъюнкция дизъюнктов фиксированного размера.

> До Кука NP был абстрактным классом недетерминированных машин. После появился конкретный представитель, к которому сводят другие задачи. Если SAT решается за полиномиальное время, то $\mathrm{P} = \mathrm{NP}$.

Кук опубликовал результат в 1971 году, Левин — независимо в 1973. Карп в 1972 году добавил 21 NP-полную задачу и цепочки сведений.

Цепочки сведений

Теорема 4 (Стандартные цепочки сведений)

$$\text{SAT} \underset{p}{\leq} \text{3-SAT} \underset{p}{\leq} \text{Vertex Cover} \underset{p}{\leq} \text{Clique} \underset{p}{\leq} \text{Independent Set}.$$

Другие ветви:

- $\text{3-SAT} \underset{p}{\leq} \text{Subset Sum} \underset{p}{\leq} \text{Partition} \underset{p}{\leq} \text{Knapsack}$;
- $\text{3-SAT} \underset{p}{\leq} \text{3-Coloring}$;
- $\text{Vertex Cover} \underset{p}{\leq} \text{Hamiltonian Cycle} \underset{p}{\leq} \text{TSP}$.

Дойдя от SAT до задачи B, мы автоматически доказали её NP-трудность.

IND и CLIQUE

Пример (Независимое множество и клика)

IND: есть ли k попарно несмежных вершин? CLIQUE: есть ли k попарно смежных вершин?

Это одна задача, вывернутая наизнанку. Возьмём дополнение $\overline{G}$. Ребро есть в $\overline{G}$ тогда и только тогда, когда его нет в G. Независимое множество в G — клика в $\overline{G}$. Сведение: $f(G, k) = \left(\overline{G}, k\right)$. Построение дополнения занимает $O(|V|^2)$ времени.

3-SAT к вершинному покрытию: конструкция

Пример (Конструкция)

Дана 3-КНФ формула φ с m дизъюнктами от n переменных. Строим граф G и число $k = n + 2m$.

Гаджеты:

- переменный: ребро из двух вершин $x_i, \overline{x_i}$;
- дизъюнктный: треугольник из вершин-литералов $(l_1 \vee l_2 \vee l_3)$;
- соединение: вершина треугольника связана с одноимённой вершиной переменного гаджета.

3-SAT к вершинному покрытию: корректность

Пример (Корректность)

Если φ выполнима: берём истинный литерал из каждого переменного ребра (n вершин) и две оставшиеся вершины каждого треугольника ($2m$). Третья вершина покрыта со стороны переменного гаджета.

Обратно: для покрытия каждого переменного ребра нужно не менее n вершин, для каждого треугольника — не менее $2m$. Покрытие размера $n + 2m$ содержит ровно одну вершину каждого ребра и ровно две вершины каждого треугольника. Третья вершина покрыта со стороны гаджета: её литерал истинен, и дизъюнкт выполнен.

Размер графа: $2n + 3m$ вершин.

3-SAT к вершинному покрытию: вживую на числах

Пример (Малый пример)

Формула $\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$: $n = 3$ переменные, $m = 2$ дизъюнкта. По конструкции $k = n + 2m = 3 + 4 = 7$.

Гаджеты:

- переменные: рёбра $\{x_1, \neg x_1\}$, $\{x_2, \neg x_2\}$, $\{x_3, \neg x_3\}$;
- дизъюнкты: треугольники $(x_1, x_2, \neg x_3)$ и $(\neg x_1, x_2, x_3)$;
- соединение: вершина треугольника связана с одноимённой вершиной переменного гаджета.

Покрытие размера k

Пример (Вершинное покрытие из $k = 7$ вершин)

Возьмём выполняющее присваивание с ровно одним истинным литералом в каждом дизъюнкте: $x_1 =$ истина, $x_2 =$ ложь, $x_3 =$ истина.

Вершинное покрытие размера $k = 7$:

- из каждого переменного ребра — истинный литерал: x_1, $\neg x_2$, x_3 ($n = 3$ вершины);
- из каждого треугольника — две вершины с ложными литералами ($2m = 4$ вершины).

Третья вершина каждого треугольника (истинный литерал) покрыта со стороны переменного гаджета. Все рёбра покрыты: переменное ребро — выбранной вершиной, ребро треугольника — двумя выбранными вершинами, соединение — со стороны истинного литерала.

NP-полнота — про худший случай

NP-полнота утверждает трудность в худшем случае. Полиномиального алгоритма для всех случаев не существует. Конкретные экземпляры почти всегда имеют структуру: малую древесную ширину, разреженность ограничений, декомпозицию.

> NP-полные задачи решают каждый день: TSP для тысяч городов, SAT-решатели на миллионах переменных. Сложность — граница в принципе. Инженерия — то, что сходится на конкретных данных.

Когда точное решение недостижимо, инженер меняет цель: вместо оптимального решения — достаточно хорошее. Стратегии:

- точные алгоритмы с отсечениями;
- приближения с гарантированной точностью;
- эвристики без гарантий;
- параметризованные алгоритмы: экспонента только по малому параметру.

Параметризованная сложность

Определение 6 (Класс FPT)

Задача с параметром k параметрически разрешима, если она разрешима за время $f(k) \cdot n^{O(1)}$. Экспонента загнана в параметр. По размеру входа алгоритм полиномиален.

Пример (Вершинное покрытие)

Перебор всех подмножеств размера k занимает $O\left(n^k\right)$: параметр в показателе, это не FPT. Ветвление по концам непокрытого ребра даёт $O\left(2^k \cdot n\right)$. Для графа из миллиона вершин и $k = 10$ ветвление — секунды. Перебор n^k при тех же n и k даёт порядка 10^{60} операций.

За пределами NP

Таксономия сложности не кончается на P и NP.

- Память: PSPACE — полиномиальная память; QBF — каноническая задача; Сэвич: $\mathrm{NPSPACE} = \mathrm{PSPACE}$.
- Случайность: BPP — полиномиальные вероятностные алгоритмы с ограниченной ошибкой; RP и ZPP — односторонняя и нулевая ошибка.
- Подсчёт: #P — задачи подсчёта решений; #SAT сложнее SAT.

$$P \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}.$$

Известно, что $P \subset \mathrm{EXP}$: хотя бы одно включение строгое. Какое именно — открытый вопрос.

Зоопарк классов

Класс	Машина	Ошибка	Ресурс	Пример
P	детерминированная	нет	время	сортировка
NP	недетерминированная	нет	время	SAT
RP	вероятностная	на «да»	время	различие полиномов
coRP	вероятностная	на «нет»	время	проверка простоты
BPP	вероятностная	двусторонняя	время	подсчёт решений
ZPP	вероятностная	нулевая	ожидаемое время	быстрая сортировка
PSPACE	детерминированная	нет	память	QBF

ZPP внутри RP

ZPP $\subseteq$ RP: алгоритм с нулевой ошибкой и полиномиальным *ожидаемым* временем превращается в односторонний.

Запустим алгоритм A из ZPP на $2T$ шагов, где T — его ожидаемое время. Если A завершился — вернём его ответ, он всегда верен. Если не завершился — вернём «нет». Так получается алгоритм B из RP.

Случай	A завершился	A не завершился	Ответ алгоритма B
$x \in L$	«да», верно	«нет», ошибка	«да» с вероятностью $\geq \frac{1}{2}$
$x \notin L$	«нет», верно	«нет», верно	всегда «нет»

Для $x \in L$ ошибка возможна только при тайм-ауте. По неравенству Маркова $P[T \geq 2T] \leq \frac{\mathrm{E}[T]}{2T} = \frac{1}{2}$. С числами: при $\mathrm{E}[T] = 10$ запускаем на 20 шагов, и $P[T \geq 20] \leq \frac{10}{20} = \frac{1}{2}$. Значит, для $x \in L$ ответ «да» приходит с вероятностью не меньше $\frac{1}{2}$ — это ровно определение RP.

Что мы поняли?

- P — задачи, решаемые за полиномиальное время; NP — задачи, проверяемые за полиномиальное время.
- Сведение $A \underset{p}{\leq} B$ показывает, что B не легче A; NP-полная задача — самая трудная в NP.
- Теорема Кука–Левина даёт первую NP-полную задачу — SAT; цепочки сведений переносят NP-полноту на клику и вершинное покрытие.
- NP-полнота — свойство худшего случая; практика живёт структурой экземпляров: точные алгоритмы, приближения, эвристики, FPT.
- Вопрос $\mathrm{P} = \mathrm{NP}$ открыт полвека; за NP лежат PSPACE, BPP и #P.

Вопросы для самопроверки

- Почему в качестве эталона эффективности выбран полином, а не $O(n \log n)$?
- Чем NP отличается от P, и почему $\mathrm{P} \subseteq \mathrm{NP}$ доказывается так коротко?
- Чем NP-полная задача отличается от NP-трудной?
- Как свести независимое множество к клике, и почему сведение корректно?
- Почему из NP-полноты SAT следует NP-полнота вершинного покрытия?
- Что было бы, если бы $\mathrm{P} = \mathrm{NP}$?