Основные понятия теории NP-полноты

Труднорешаемые задачи

Рассмотренные ранее алгоритмы были полиномиальными.

Полиномиальным алгоритмом (или алгоритмом полиномиальной временной сложности) называется алгоритм, временная сложность которого равна O($p(n)$), где $p(n)$ – некоторая полиномиальная функция, а n – входная длина.

Множество полиномиальных алгоритмов образует класс полиномиальных алгоритмов – *класс P*.

Всякая ли задача может быть решена за полиномиальное время? Ответ отрицательный, так как существуют вообще неразрешимые задачи, и есть задачи разрешимые, но за более чем полиномиальное время. Примеры:

1. Решение задачи об укладке рюкзака. При условии, что каждый из n предметов может быть упакован или не упакован в рюкзак, всего будет 2^n вариантов взятых предметов.
2. Решение задачи коммивояжера сводится в худшем случае к перебору всех возможных маршрутов, а их всего существует (n-1)!. Это зависимость не может быть выражена полиномиальной функцией, причем можно заметить, что $n \geq 2^n$

Задачи, для решения которых не существует полиномиального алгоритма, но существует экспоненциальный алгоритм (O(a^n)), называют *труднорешаемыми*.

Отличие полиномиальных и экспоненциальных алгоритмов будет более ощутимо, если обратиться к таблице, в которой отображено время работы алгоритма на компьютере, выполняющем 1000000 оп/с:

n / $O(n)$	10	20	30	40	50	60
n	0.00001 с	0.00002 с	0.00003 с	0.00004 с	0.00005 с	0.00006 с
n^2	0.0001 с	0.0004 с	0.0009 с	0.0016 с	0.0025 с	0.0036 с
n^5	0.1 с	3.2 с	24.3 с	1.7 мин	5.2 мин	13 мин
2^n	0.001 с	1 с	17.9 мин	12.7 дней	35.7 лет	366 столет.
3^n	0.059 с	58 мин	6.5 лет	3855 стол.	$2 \cdot 10^8$ стол.	$1.3 \cdot 10^{13}$ст.
$n!$	3.6 с	771,5 стол.	$8 \cdot 10^{16}$стол	…	…	…

Таким образом, понятие полиномиально разрешимой задачи принято считать уточнением идеи «практически разрешимой» задачи:

1. Полиномиальные алгоритмы действительно работают быстро, а полиномы больших степеней (n^{100}) и полиномы с большим коэффициентом пропорциональности ($10^{99} \cdot n^a$) на практике встречаются крайне редко.
2. Полиномиальные алгоритмы не зависят от выбора конкретной модели вычислений ($P_{\text{по Тьюрингу}} = P_{\text{на МПД}} = P_{\text{для параллельных вычислений}}$).

3. Класс *P* обладает свойством замкнутости (композиция двух полиномиальных алгоритмов дает алгоритм также полиномиальный, так как сумма, произведение и композиция многочленов, тоже являются многочленами).

Абстрактные задачи

Введем следующую абстрактную модель вычислительной задачи.

Будем называть *абстрактной задачей* произвольное бинарное отношение *Q* между элементами двух множеств – множества условий *I* и множество решений *S*.

Например, рассмотрим задачу коммивояжера.

Задача: имеются n городов, расстояния между которыми заданы. Коммивояжеру необходимо выйти из какого-то города, посетить остальные n-1 городов точно по одному разу и вернуться в исходный город. При этом маршрут коммивояжера должен быть минимальным.

Абстрактная задача для данной следующая:

I={*C* – матрица расстояний, *k* – номер начального города}.

S={{*k*, j_2, j_3, j_4, j_{n-1}, j_n} – последовательность вершин, таких, что сумма расстояний $c_{k,j2}+ c_{j2,j3}+ c_{j3,j4}+\ldots+ c_{jn-1,jn}+ c_{jn,k}$ минимальна}.

В теории NP-полноты рассматриваются только *задачи разрешения* – задачи, в которых требуется дать ответ «да» или «нет» на некоторый вопрос.

Формально задачу разрешения можно рассматривать как функцию, отображающую множество условий *I* во множество {0,1} (1 – «да», 0 – «нет»).

Часто встречаются *задачи оптимизации*, в которых требуется минимизировать или максимизировать значение некоторой величины. Прежде чем ставить вопрос о *NP*-полноте таких задач, их следует преобразовать в задачу разрешения путем постановки вопроса, является ли число *K* верхней (или нижней) границей для оптимизируемой величины.

Например, для задачи коммивояжера может быть построена следующая задача разрешения: «Существует ли путь коммивояжера, не превосходящий *K*?»

Как решить эту задачу? Найти минимальное решение и сравнить с *K*. Несмотря на то что зачастую практически невозможно быстро решить труднорешаемую задачу, ее конкретное решение возможно быстро (за полиномиальное время) проверить. Решение этой задачи имеет экспоненциальную оценку. Однако чтобы проверить, является ли некоторая последовательность городов решением задачи (образует путь, не превосходящий *K*), требуется линейное время.

Заметим, что если полученная задача разрешения – экспоненциальная задача (*NP*-полная), то и исходная оптимизационная задача тоже является экспоненциальной (*NP*-полная), так как если бы для исходной задачи существовал полиномиальный алгоритм, то и задача разрешения решалась

бы за полиномиальное время. Таким образом, теорию *NP*-полноты можно использовать и для исследования задач оптимизации.

Класс NP – класс всех задач разрешения, которые могут быть решены недерминированным (*N*) алгоритмом за полиномиальное время (*P*).

В чем *отличие детерминированных и недетерминированных алгоритмов*?

1. Недетерминированный алгоритм – на входе одно «угаданное» решение, корректность которого проверяется за полиномиальное время. Детерминированный алгоритм – на входе условие задачи, на выходе – решение, полученное за полиномиальное время.

2. Отсутствие симметрии между ответами «да» и «нет». Если задача «Дано *I*; верно ли, что для *I* выполняется свойство *X*?» может быть решена полиномиальным (детерминированным) алгоритмом, то такое же утверждение справедливо и для дополнения задачи: «Дано *I*; верно ли, что для *I* не выполняется свойство *X*?» Для недетерминированного алгоритма это утверждение неверно, например, дополнением задачи коммивояжера является задача «Верно ли, что любой путь коммивояжера длиннее *K*?» Решение этой задачи, очевидно, не может быть проверено за полиномиальное время.

Формальные языки

Для задач разрешения удобно использовать терминологию формальных языков.

Алфавитом Σ называется любой конечный набор символов.

Языком L над алфавитом Σ называется произвольное множество строк символов из алфавита Σ (такие строки называют *словами в алфавите* Σ).

Например: алфавит $\Sigma=\{0,1\}$ и язык $L=\{10,11,101,111,1011,1101,10001,\ldots\}$, состоящий из двоичных записей простых чисел.

Введем *обозначения*:

ε – пустое слово;

$\varnothing$ – пустой язык, не содержащий слов;

Σ^* – язык, состоящий из всех строк в алфавите Σ, т. е. $\forall\, L \subset \Sigma^*$.

Операции над языками:

1) объединение;
2) пересечение;
3) дополнение (дополнением языка L называют язык $\overline{L} = \Sigma^* \backslash L$);
4) конкатенация (конкатенацией (соединением) двух языков L_1 и L_2 называется язык $L=\{x_1x_2 : x_1 \in L_1, x_2 \in L_2\}$);
5) замыкание (замыканием (*-операцией Клини) языка L называется язык $L^*=\{\varepsilon\} \cup L \cup L^2 \cup L^3 \cup \ldots$, где L^k – язык, полученный k-кратной конкатенацией с самим собой).

Таким образом, задача разрешения является языком над алфавитом $\Sigma=\{0,1\}$.

Например: задаче коммивояжера соответствует язык $TSP=\{<G,v,k>: G=(V,E)$ – ориентированный граф, $v \in V$ – номер начального города, $k \geq 0$ – целое число, и существует путь коммивояжера, не превосходящий $k\}$.

Алгоритм A допускает слово $x \in \{0,1\}^*$, если на входе x алгоритм выдает результат 1 ($A(x)=1$).

Алгоритм A отвергает слово $x \in \{0,1\}^*$, если на входе x алгоритм выдает результат 0 ($A(x)=0$).

Замечание. Алгоритм A может на слове x зациклиться или выдать ответ, отличный от 0 или 1, тогда говорят, что алгоритм A не отвергает и не принимает слово x.

Алгоритм A допускает язык L, если алгоритм допускает те и только те слова, которые принадлежат языку L.

Если алгоритм допускает все слова из L, а все остальные слова отвергает, то говорят, что *алгоритм A распознает язык L*.

Язык L допускается за полиномиальное время, если имеется алгоритм A, который допускает данный язык, причем всякое слово $x \in L$ допускается алгоритмом за время $O(n^k)$, где n – длина слова x, а k – некоторое не зависящее от x число.

Язык L распознается за полиномиальное время, если имеется алгоритм A, который распознает данный язык, причем время работы алгоритма на каждом слове длины n не больше $O(n^k)$.

Переформулируем определение сложностного класса P следующим образом: $P=\{L \subset \{0,1\}^*$: существует алгоритм A, распознающий язык L за полиномиальное время$\}$.

Назовем *проверяющим алгоритмом* алгоритм A с двумя аргументами; первый аргумент мы будем называть входной строкой, а второй – сертификатом.

Алгоритм A с двумя аргументами допускает вход x, если существует сертификат y, для которого $A(x,y)=1$.

Языком, проверяемым алгоритмом A, называется язык L, такой, что $L=\{x \in \{0,1\}^*$: существует $y \in \{0,1\}^*$, для которого $A(x,y)=1\}$.

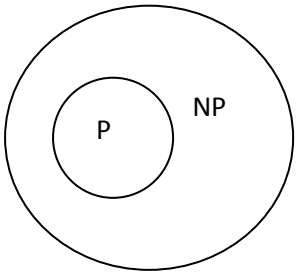
Другими словами, *алгоритм A проверяет язык L*, если для любой строки $x \in L$ найдется сертификат y, с помощью которого A может проверить принадлежность x к языку L, а для $x \notin L$ такого сертификата нет. Например, для задачи коммивояжера сертификатом будет путь коммивояжера с длиной не более k.

Переформулируем определение сложностного класса NP следующим образом: класс NP – это класс языков, для которых существуют проверяющие алгоритмы, работающие полиномиальное время, причем длина сертификата также ограничена некоторым полиномом, т. е. L принадлежит классу NP, если существует такой полиномиальный алгоритм A с двумя

аргументами и такой многочлен $p(x)$ с целыми коэффициентами, что $L=\{x\in\{0,1\}^*$: существует сертификат y, для которого $|y|\le p(|x|)$ и $A(x,y)=1\}$. В этом случае говорят, что алгоритм A проверяет язык L за полиномиальное время.

Нерешенные проблемы теории NP-полноты

Как соотносятся классы P и NP? Так как любая задача класса P проверяема за полиномиальное время, то очевидно следующее включение.

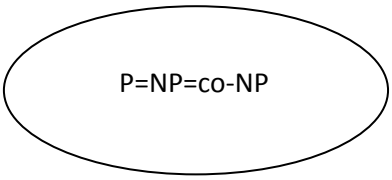


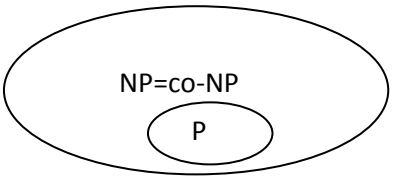
Однако не известно, совпадают ли классы P и NP, но большинство специалистов считает, что нет. Интуитивно класс P можно представить себе как класс задач, которые можно быстро решить, а класс NP – как класс задач, решение которых можно быстро проверить.

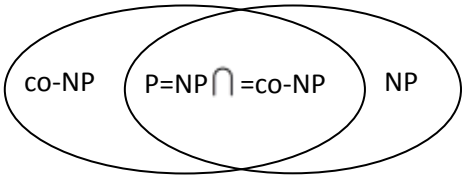
Существуют и другие открытые вопросы относительно класса NP:

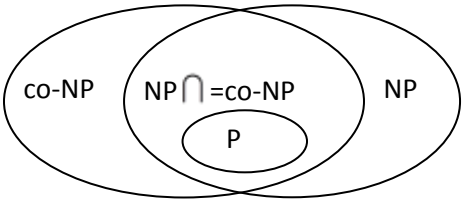
1. Вопрос эквивалентности классов P и NP.
2. Вопрос замкнутости класса NP относительно дополнения, т. е. верно ли, что если $L\in NP$, то и $\bar{L}\in NP$. Множество всех задач L, для которых это условие выполняется, назовем классом *co-NP*.

Так как класс P замкнут относительно дополнения, то возможны следующие соотношения классов.









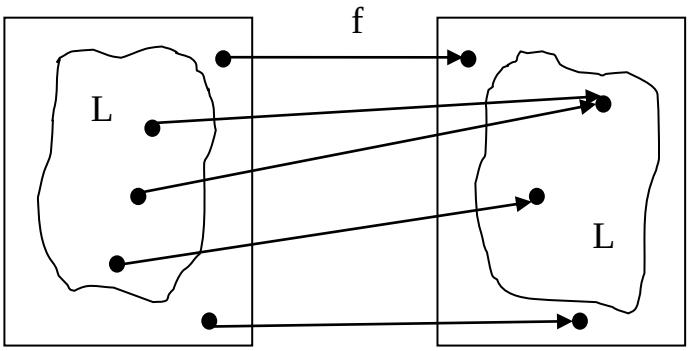
Полиномиальная сводимость и NP-полные задачи

Наиболее убедительным аргументом в пользу того, что классы *P* и *NP* различны, является существование так называемых *NP*-полных задач. *NP*-полные языки являются самыми «трудными» в классе *NP*. При этом трудность языков можно сравнивать, сводя один язык к другому.

Полиномиальная сводимость

Определение. Язык L_1 за полиномиальное время к языку L_2 (обозначение: $L_1 \leq_p L_2$ или $L_1 \propto L_2$), если существует такая функция $f : \{0, 1\}^*$, вычислимая за полиномиальное время, что для любого слова $x \in \{0, 1\}^* : x \in L_1 \Leftrightarrow f(x) \in L_2$ (x является индивидуальной задачей L_1 тогда и только тогда, когда $f(x)$ является индивидуальной задачей L_2).

Функцию f называют *сводящей функцией*, а полиномиальный алгоритм A_f, вычисляющий функцию f, – *сводящим алгоритмом*.



$$\forall\, x \in L_1 \rightarrow f(x) \in L_2$$
$$\forall\, x \notin L_1 \rightarrow f(x) \notin L_2$$

Лемма 1. Если язык $L_1 \subseteq \{0, 1\}^*$ сводится за полиномиальное время к языку $L_2 \subseteq \{0, 1\}^*$ и L_2 принадлежит классу P, то L_1 также принадлежит классу P (*если* $L_1 \propto L_2$ *и* $L_2 \in P$, *то* $L_1 \in P$).

Доказательство

1) $L_1 \propto L_2$, следовательно, существует полиномиальный сводящий алгоритм A;

2) $L_2 \in P$, следовательно, существует полиномиальный алгоритм A_{L_2} распознавания языка L_2.

Возьмем произвольное слово $x \in L_1$ и с помощью алгоритма A сведем его к слову $A(x) \in L_2$, а затем определим, принимается или отвергается слово $A(x)$ полиномиальным алгоритмом A_{L_2}. Из определения полиномиальной сводимости следует, что слово $A(x)$ допускается алгоритмом A_{L_2} тогда и только тогда, когда слово x допускается алгоритмом, распознающим язык L_1. Таким образом, последовательное выполнение алгоритмов A и A_{L_2} дает алгоритм распознавания языка L_1, и этот алгоритм полиномиален (сумма полиномов есть полином), т. е. $L_1 \in P$.

Лемма 2. Отношение полиномиальной сводимости транзитивно (*если* $L_1 \propto L_2$ *и* $L_2 \propto L_3$, *то* $L_1 \propto L_3$).

Определение. Языки L_1 и L_2 полиномиально эквивалентны, если сводятся друг к другу.

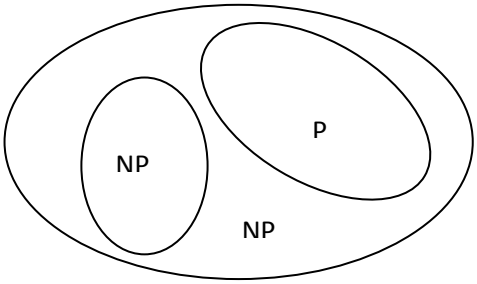
Понятие полиномиальной сводимости позволяет произвести классификацию языков по степени их «трудности». Запись $L_1 \propto L_2$ можно интерпретировать так: сложность языка L_2 не более чем полиномиально превосходит сложность языка L_1. Наиболее трудные в этом смысле – *NP*-полные языки.

Определение. Язык $L \subseteq \{0, 1\}^*$ называется *NP*-полным:

1) если $L \in NP$;

2) $L' \propto L$ для любого языка $L' \in NP$.

Обозначение класса *NP*-полных языков – *NPC*.



Лемма 3. Если языки L_1 и L_2 принадлежат классу *NP*, L_1 – это *NP*-полный язык и L_1 полиномиально сводится к L_2, то L_2 также *NP*-полный язык (*если* $L_1 \in NP$, $L_2 \in NP$, $L_1 \in NPC$ *и* $L_1 \propto L_2$, *то* $L_2 \in NPC$).

Доказательство. Так как $L_2 \in NP$, то, согласно определению, достаточно доказать, что любой язык $L' \in NP$ полиномиально сводится к языку L_2 ($L' \propto L_2$).

По условию леммы $L_1 \in NPC$, следовательно, любой язык $L' \in NP$ сводится к языку L_1 ($\forall L' \in NP$: $L' \propto L_1$), а язык L_1 сводится к языку L_2 (по условию леммы $L_1 \propto L_2$). По свойству транзитивности полиномиальной сводимости (лемма 2) $\forall L' \in NP$: $L' \propto L_1$, $L_1 \propto L_2$, следовательно, $\forall L' \in NP$: $L' \propto L_2$, т.е язык $L_2 \in NPC$.

Основное свойство NP-полных языков: если некоторый *NP*-полный язык разрешим за полиномиальное время, то классы *P* и *NP* эквивалентны (*P*=*NP*), т. е. если в классе *NPC* существует язык разрешимый за полиномиальное время, то все языки класса *NPC* разрешимы за полиномиальное время.

Доказательство. Пусть L – *NP*-полный язык, который одновременно оказался разрешимым за полиномиальное время ($L \in NPC$ и $L \in P$). По определению *NP*-полного языка справедливо утверждение: любой язык L', проверяемый за полиномиальное время, полиномиально сводится к языку L ($\forall L' \in NP$: $L' \propto L$). Таким образом, $\forall L' \in NP$: $L' \propto L$ и $L \in P$, значит, по лемме 1 следует, что $\forall L' \in NP$ разрешима за полиномиальное время ($\forall L' \in NP$: $L' \in P$), т. е. классы *P* и *NP* совпадают.