28.04.2025

NP-полнота и вычислительная неразрешимость.

Филиппов Михаил Витальевич

m.filippov@g.nsu.ru

89232283872

Императивное программирование, 2024-2025

Новосибирский государственный университет

*НАСТОЯЩАЯ НАУКА

Давайте познакомимся

Филиппов Михаил Витальевич

- Окончил магистратуру ФФ НГУ
- Окончил аспирантуру ИТ СО РАН
- Являюсь м.н.с. ИТ СО РАН
- 7+ лет опыт в программировании C/C++

Адженда

NP-алгоритмы

90 минут

Введение

Рабочим критерием эффективности будет полиномиальное время выполнения. У такого конкретного определения есть одно важное преимущество: оно позволяет привести математическое доказательство того, что некоторые задачи не могут быть решены при помощи алгоритмов с полиномиальным временем (то есть «эффективных»).

Когда ученые занялись серьезными исследованиями вычислительной сложности, им удалось сделать первые шаги к доказательству того, что эффективных алгоритмов для решения некоторых особенно сложных задач не существует. Но для многих фундаментальных дискретных вычислительных задач (из области оптимизации, искусственного интеллекта, комбинаторики, логики и т. д.) этот вопрос оказался слишком сложным и с тех пор так и остается открытым: алгоритмы с полиномиальным временем для таких задач неизвестны, но и доказательств того, что таких алгоритмов не существует, пока нет.

Если для одной из таких задач будет найден алгоритм с полиномиальным временем выполнения, это будет означать существование алгоритма для всех таких задач. Такие задачи ***называются NP-полными.***

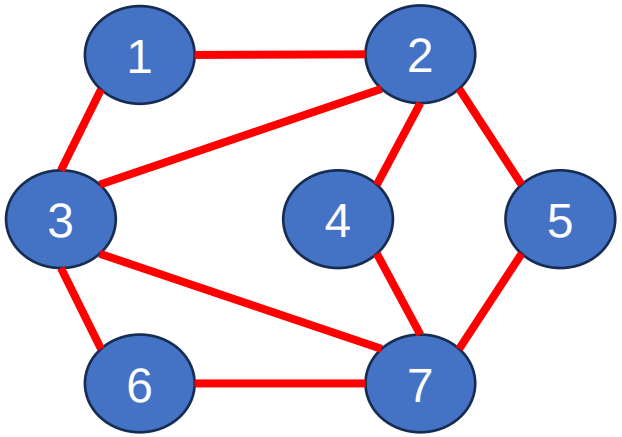
Полиномиальное сведение

Какая математической характеристика такого большого класса?
«задача X обладает как минимум не меньшей сложностью, чем задача Y».
Докажем, что «черный ящик», способный решить задачу X, также сможет решить задачу Y.
Предположим, что задача X в нашей вычислительной модели может быть напрямую решена с полиномиальным временем.
Утверждение 1.1. Допустим, $Y \leq_P X$. Если задача X решается за полиномиальное время, то и задача Y может быть решена за полиномиальное время.
Утверждение 1.2. Допустим, $Y \leq_P X$. Если задача Y не решается за полиномиальное время, то и задача X не может быть решена за полиномиальное время.

Первое сведение: независимое множество и вершинное покрытие

Алгоритм с полиномиальным временем для нее неизвестен, но мы также не знаем, как доказать, что такого алгоритма не существует.

Вернемся к формулировке задачи о независимом множестве, так как в нее будет добавлен один нюанс. Вспомните, что в графе G = (V, E) множество узлов S ⊆ V называется независимым, если никакие два узла в S не соединены ребром. Найти малое независимое множество в графе несложно (например, одиночный узел образует независимое множество); трудности возникают при поиске больших независимых множество так как требуется построить большой набор узлов, в котором нет соседних узлов. Например, множество узлов {3, 4, 5} является независимым множеством размера 3 в графе на рис. 1.1, тогда как множество узлов {1, 4, 5, 6} является независимым множеством большего размера.



Граф, для которого размер наибольшего независимого множества равен 4, а наименьшее вершинное покрытие имеет размер 3

задача о независимом множестве будет сформулирована следующим образом:

Для заданного графа G и числа k содержит ли G независимое множество с размером не менее k?

Первое сведение: независимое множество и вершинное покрытие

С точки зрения разрешимости с полиномиальным временем оптимизационная версия задачи (найти максимальный размер независимого множества) не так уж сильно отличается от версии с проверкой условия (решить, существует ли в G независимое множество с размером не менее заданного k, — да или нет). Зная метод решения первой версии, мы автоматически решим вторую (для произвольного k).

Но также существует чуть менее очевидный обратный факт: если мы можем решить версию задачи о независимом множестве с проверкой условия для всех k, то мы также можем найти максимальное независимое множество. Для заданного графа G с n узлами версия с проверкой условия просто проверяется для каждого k; наибольшее значение k, для которого ответ будет положительным, является размером наибольшего независимого множества в G. Теперь для демонстрации нашей стратегии по установлению связи между сложными задачами будет рассмотрена другая фундаментальная задача из теории графов, для которой неизвестен эффективный алгоритм: задача о вершинном покрытии. Для заданного графа $G = (V, E)$ множество узлов $S \subseteq V$ образует вершинное покрытие, если у каждого ребра $e \in E$ по крайней мере один конец принадлежит S.

Мы не знаем, как решить задачу о независимом множестве или задачу о вершинном покрытии за полиномиальное время; но что можно сказать об их относительной сложности?

Первое сведение: независимое множество и вершинное покрытие

Утверждение 1.3. Имеется граф G = (V, E). S является независимым множеством в том и только в том случае, если его дополнение V−S является вершинным покрытием.

Доказательство. Предположим, что S является независимым множеством. Рассмотрим произвольное ребро e = (u, v). Из независимости S следует, что u и v не могут одновременно принадлежать S; следовательно, один из этих концов должен принадлежать V−S. Отсюда следует, что у каждого ребра как минимум один конец принадлежит V−S, а значит, V−S является вершинным покрытием.

И наоборот, предположим, что V−S является вершинным покрытием. Рассмотрим два любых узла u и v в S. Если бы они были соединены ребром e, то ни один из концов e не принадлежал бы V−S, что противоречило бы предположению о том, что V−S является вершинным покрытием. Отсюда следует, что никакие два узла в S не соединены ребром, а значит, S является независимым множеством. ■

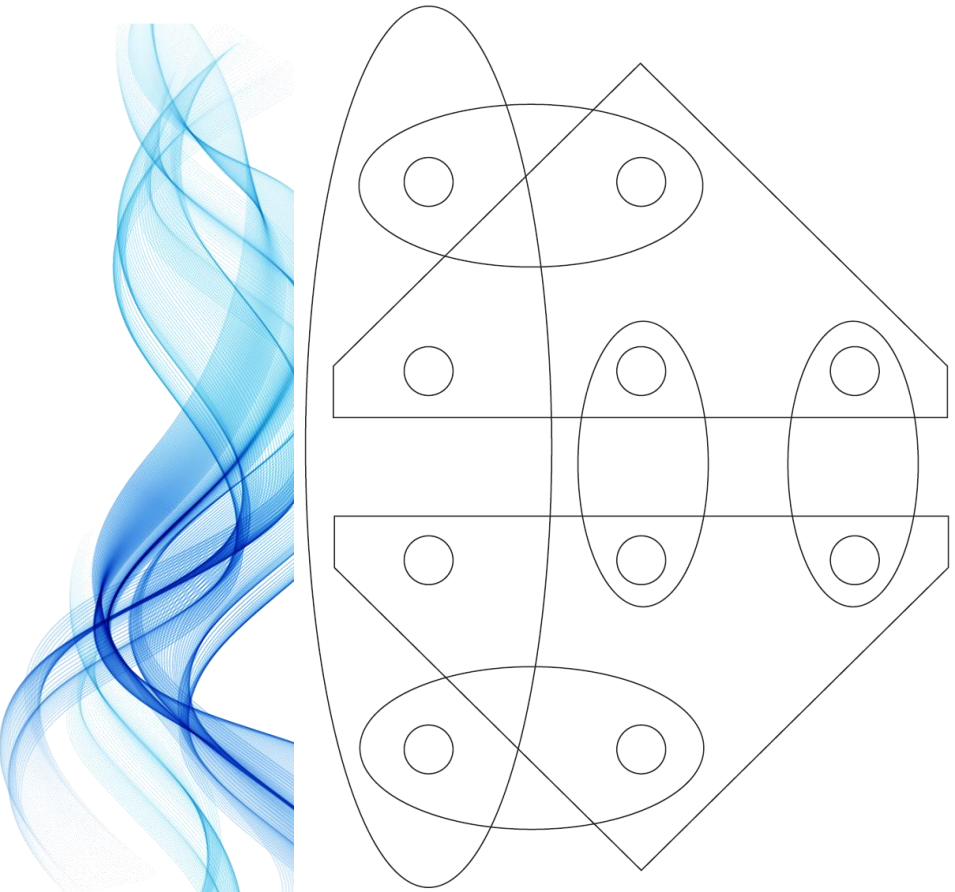
Утверждение 1.4. Независимое множество $\leq_P$ Вершинное покрытие.

Доказательство. Имея «черный ящик» для решения задачи о вершинном покрытии, мы могли бы решить, содержит ли G независимое множество с размером не менее k, обратившись к «черному ящику» с вопросом о том, содержит ли G вершинное покрытие с размером не более n − k. ■

Утверждение 1.5. Вершинное покрытие $\leq_P$ Независимое множество.

Доказательство. Имея «черный ящик» для решения задачи о независимом множестве, мы могли бы решить, содержит ли G вершинное покрытие размером не более k, обратившись к «черному ящику» с вопросом о том, содержит ли G независимое множество с размером не менее n − k. ■

Сведение к более общему случаю: вершинное покрытие к покрытию множества



Задача о независимом множестве и задача о вершинном покрытии представляют собой две разные категории задач. Задача о независимом множестве может рассматриваться как «задача упаковки»: требуется «упаковать» как можно больше вершин с учетом конфликтов (ребер), препятствующих нам в этом. С другой стороны, задача о вершинном покрытии может рассматриваться как «задача покрытия»: требуется экономно «накрыть» все ребра в графе с использованием минимального количества вершин.

Задано множеств U из n элементов, набор S_1, ..., S_m подмножеств U и число k. Существует ли набор из не более чем k таких множеств, объединение которых равно всему множеству U?

Представьте, например, что имеются m программных компонентов и множество U из n функций, которыми должна обладать система. i-й компонент реализует множество $S_i \subseteq U$ функций. В задаче покрытия множества требуется включить небольшое количество компонентов, чтобы система поддерживала все n функций.

Рисунок. Пример задачи покрытия множества

Сведение к более общему случаю

Утверждение 1.6. Вершинное покрытие $\leq_P$ Покрытие множества.

Доказательство. Допустим, имеется «черный ящик», который умеет решать задачу о покрытии множества, и произвольный экземпляр задачи о вершинном покрытии, заданный графом G = (V, E) и числом k. Как использовать «черный ящик» для решения этой задачи?

Наша цель — найти покрытие для ребер E, поэтому мы формулируем экземпляр задачи покрытия множества, в которой универсальное множество U равно E. Каждый раз, когда мы выбираем вершину в задаче вершинного покрытия, мы покрываем все ребра, инцидентные этой вершине; следовательно, для каждой вершины $i \in V$ в экземпляр задачи покрытия множества добавляется множество $S_i \subseteq U$, состоящее из всех ребер G, инцидентных i.

Утверждается, что U может быть покрыто с использованием не более k из множеств S_1, ..., S_n в том, и только в том случае, если G имеет вершинное покрытие с размером не более k. Это утверждение доказывается очень просто. Если $l \leq k$ множеств $S_{i_1}, \ldots, S_{i_l}$ покрывают U, то каждое ребро в G инцидентно одной из вершин $i_1 \ldots i_l$, а значит, множество $\{i_1 \ldots i_l\}$ является вершинным покрытием G с размером $l \leq k$. И наоборот, если $\{i_1 \ldots i_l\}$ является вершинным покрытием G с размером $l \leq k$, то множества покрывают U.

Итак, для заданного экземпляра задачи о вершинном покрытии формулируется экземпляр задачи покрытия множества, описанный выше, который передается «черному ящику». Положительный ответ на исходный вопрос дается в том, и только в том случае, ес «черный ящик» отвечает положительно.■

Утверждение 1.7. Независимое множество $\leq_P$ Упаковка множества.

Сведение с применением «регуляторов»: задача выполнимости

Сейчас мы займемся более абстрактными задачами, которые формулируются в булевой записи. Эти задачи применяются для моделирования широкого спектра задач, в которых требуется присваивать значения условных переменных для выполнения заданного набора ограничений: например, подобные формальные конструкции часто встречаются в области искусственного интеллекта.

Задачи SAT и 3-SAT

Допустим, имеется множество X из n булевых переменных x_1, ..., x_n; каждая переменная может принимать значение 0 или 1 (эквиваленты false и true). Литералом по X называется одна из переменных x_i или ее отрицание $\overline{x_i}$. Наконец, условием называется обычная дизъюнкция литералов

$$t_1 \vee t_2 \vee \cdots \vee t_l.$$

(Еще раз: все $t_i \in \{x_1, x_2, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$). Мы говорим, что условие имеет длину l, если оно содержит l литералов.

Логическим присваиванием для X называется присваивание значения 0 или 1 каждому x_i; другими словами, это функция v : X → {0,1}. Присваивание v неявно задает $\overline{x_i}$ значение истинности, противоположное x_i. Присваивание выполняет условие C, если после него C дает результат 1 по условиям булевой логики; это эквивалентно требованию о том, чтобы по крайней мере один из литералов в C имел значение 1. Присваивание выполняет совокупность условий C_1, ..., C_k, если в результате его все C_i дают результат 1; иначе говоря, если в результате его конъюнкция

$$C_1 \wedge C_2 \wedge \cdots \wedge C_k$$

дает результат 1. В этом случае v называется выполняющим присваиванием в отношении C_1, ..., C_k, а набор условий C_1, ..., C_k называется выполнимым.

Задачи SAT и 3-SAT

Пример. Допустим, имеются три условия:

$$(x_1 \vee \overline{x_2}), (\overline{x_1} \vee \overline{x_3}), (x_2 \vee \overline{x_3}).$$

Что является выполняющим присваиванием?

Формулировка задачи выполнимости, также обозначаемой SAT:

Для заданного множества условий C_1, ..., C_k по множеству переменных $X = \{x_1, ..., x_n\}$ существует ли выполняющее логическое присваивание?

Существует частный случай SAT, обладающий эквивалентной сложностью, но при этом более понятный; в нем все условия содержат ровно три литерала (соответствующих разным переменным). Назовем эту задачу задачей 3-выполнимости, или 3-SAT:

Для заданного множества условий C_1, ..., C_k, каждое из которых имеет длину 3, по множеству переменных $X = \{x_1, ..., x_n\}$, существует ли выполняющее логическое присваивание?

Задачи выполнимости и 3-выполнимости являются фундаментальными задачами комбинаторного поиска; они содержат основные составляющие сложной вычислительной задаче и в предельно упрощенном виде. Требуется принять n независимых решений (присваивания всем x_i) так, чтобы выполнить набор ограничений. Существуют разные способы выполнения каждого ограничения по отдельности, но решения придется скомбинировать так, чтобы все ограничения выполнялись одновременно.

Сведение задачи 3-SAT к задаче о независимом множестве

А теперь свяжем вычислительную сложность, воплощенную в задачах SAT и 3-SAT, с другой (на первый взгляд) сложностью, представленной поиском независимых множеств и вершинных покрытий в графах, а именно: мы покажем, что 3-SAT $\leq_P$ **Независимое множество**.

Утверждение 1.8. 3-SAT $\leq_P$ Независимое множество.

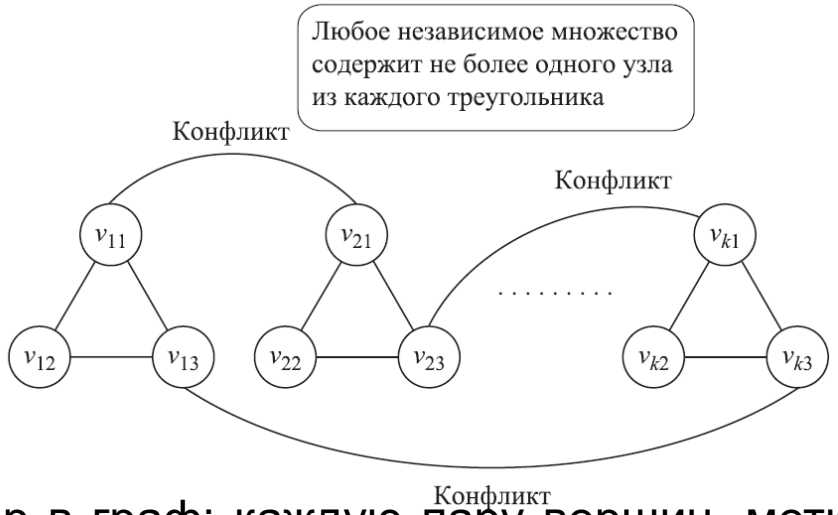
Доказательство. Имеется «черный ящик» для задачи о независимом множестве; мы хотим решить экземпляр задачи 3-SAT, состоящий из переменных $X = \{x_1, ..., x_n\}$ и условий $C_1, ..., C_k$. Чтобы правильно взглянуть на проблему сведения, следует понять, что существуют две концептуально различающиеся точки зрения на экземпляр 3-SAT.

- Первый способ представления экземпляра 3-SAT был предложен ранее: для каждой из n переменных принимается независимое решение 0/1, а успех достигается при достижении одного из трех способов выполнения каждого условия.
- Тот же экземпляр 3-SAT можно представить иначе: нужно выбрать один литерал из каждого условия, а затем найти логическое присваивание, в результате которого все эти литералы дают результат 1 с выполнением всех условий. Итак, успех достигается в том случае, если вы сможете выбрать литерал из каждого условия так, чтобы никакие два выбранных литерала не «конфликтовали»; мы говорим, что конфликт двух литералов возникает тогда, когда один равен переменной x_i, а другой ее отрицанию $\overline{x_i}$. Если удастся избежать конфликтов, мы сможем найти логическое присваивание, в результате которого выбранные литералы из каждого условия дают результат 1.

Сведение задачи 3-SAT к задаче о независимом множестве

Следующее сведение базируется на втором представлении экземпляра 3-SAT; мы закодируем его с использованием независимых множеств в графе. Сначала построим граф G = (V, E), состоящий из 3k узлов, сгруппированных в k треугольников. Таким образом, для i = 1, 2, ..., k строятся три вершины v_{i1}, v_{i2}, v_{i3}, соединенные друг с другом ребрами. Каждой вершине присваивается метка; v_{ij} помечается j-м литералом из условия C_i экземпляра 3-SAT.

Прежде чем продолжать, рассмотрим, как выглядят независимые множества с размером k на этом графе: так как две вершины не могут быть выбраны из одного треугольника, они состоят из всех способов выбора одной вершины из каждого треугольника. Так реализуется наша цель по выбору литерала в каждом условии, который дает результат 1; но пока мы не сделали ничего, чтобы предотвратить конфликт между двумя литералами.



Конфликты будут кодироваться добавлением новых ребер в граф: каждую пару вершин, метки которых соответствуют конфликтующим литералам, мы соединяем ребром. Приведет ли это к уничтожению всех независимых множеств размера k или такое множество существует? Это зависит от того, можно ли выбрать один узел из каждого треугольника, чтобы при этом не были выбраны конфликтующие пары узлов. Но ведь именно то, что требуется в экземпляре 3-SAT!

Сведение задачи 3-SAT к задаче о независимом множестве

Утверждается, что исходный экземпляр задачи 3-SAT выполним в том и только в том случае, если построенный граф G имеет независимое множество с размером не менее k.

Во-первых, если экземпляр 3-SAT выполним, то каждый треугольник в графе содержит как минимум один узел, метка которого дает результат 1. Пусть S — множество, состоящее из одного такого узла в каждом треугольнике. Множество S является независимым; если бы между двумя узлами u, v ∈ S существовало ребро, то метки u и v должны были бы конфликтовать; но это невозможно, так как обе они дают результат 1.

И наоборот, предположим, что граф G содержит независимое множество S с размером не менее k. Тогда прежде всего размер S равен точно k, и оно должно содержать один узел из каждого треугольника. Далее утверждается, что существует логическое присваивание v для переменных в экземпляре 3-SAT, обладающее тем свойством, что метки всех узлов S дают результат 1. Как же построить такое присваивание v? Для каждой переменной x_i, если ни x_i, ни $\overline{x_i}$ не используется как метка узла в S, мы присваиваем $v(x_i) = 1$. В противном случае либо x_i, либо $\overline{x_i}$ является меткой узла в S; потому что если бы один узел в S был помечен x_i, а другой $\overline{x_i}$, то эти два узла были бы соединены ребром, что противоречило бы нашему предположению о том, что S является независимым множеством. Если x_i является меткой узла в S, мы присваиваем $v(x_i) = 1$; в противном случае выполняется присваивание $v(x_i) = 0$. При таком построении v все метки узлов в S дают результат 1.Так как G содержит независимое множество с размером не менее k в том, и только в том случае, если исходный экземпляр 3-SAT был выполнимым, сведение завершено. ■

Транзитивность сведения

Мы рассмотрели несколько разных сложных задач, относящихся к разным классам, и выяснили, что они тесно связаны друг с другом. Мы можем вычислить несколько дополнительных отношений, используя факт транзитивности $\leq_P$.

Утверждение 1.9. Если $Z \leq_P Y$ и $Y \leq_P X$, то $Z \leq_P X$.

Доказательство. Располагая «черным ящиком» для решения X, мы покажем, как решить экземпляр задачи Z; по сути, мы просто выполняем композицию двух алгоритмов, подразумеваемых $Z \leq_P Y$ и $Y \leq_P X$. Алгоритм для Z выполняется с использованием «черного ящика» для Y; но каждое обращение к «черному ящику» для Y моделируется полиномиальным количеством шагов, использующих алгоритм, решающий экземпляры Y с использованием «черного ящика» для X. ■

Свойство транзитивности весьма полезно. Например, так как мы доказали, что 3-SAT $\leq_P$ Независимое множество $\leq_P$ Вершинное покрытие $\leq_P$ Покрытие множества, можно сделать вывод, что 3-SAT $\leq_P$ Покрытие множества.

Эффективная сертификация и определение NP

Сведение задач было первым главным компонентом в нашем изучении вычислительной неразрешимости. Вторым компонентом является получение характеристик класса задач, с которыми мы имеем дело. Объединение этих двух компонентов с мощной теоремой Кука и Левина приводит к некоторым неожиданным последствиям.

Здесь важен контраст между поиском решения и проверкой предложенного решения. Для задачи о независимом множестве или 3-SAT алгоритм поиска решений с полиномиальным временем нам неизвестен; однако проверка предлагаемого решения таких задач легко выполняется за полиномиальное время. Чтобы понять, что эта проблема не так уж тривиальна, представьте, какая проблема возникла бы, если бы потребовалось доказать невыполнимость экземпляра задачи 3-SAT. Какие «доказательства» можно было бы предъявить, чтобы убедить вас за полиномиальное время в невыполнимости задачи?

Задачи и алгоритмы

Теперь мы перейдем формализации.

Входные данные вычислительной задачи могут быть закодированы в виде конечной двоичной строки s. Длина строки s обозначается $|s|$. Задача принятия решения X будет описываться множеством строк, для которых возвращается ответ «да». Алгоритм A для задачи принятия решения получает входную строку s и возвращает «да» или «нет» — это возвращаемое значение будет обозначаться $A(s)$. Алгоритм A решает задачу X, если для всех строк s условие $A(s) = \text{да}$ выполняется в том, и только в том случае, если $s \in X$.

Как обычно, мы говорим, что A имеет полиномиальное время выполнения, если существует такая полиномиальная функция $p(\cdot)$, что для каждой входной строки s алгоритм A завершается для s не более чем за $O(p(|s|))$ шагов. До настоящего момента мы занимались задачами, которые решались за полиномиальное время. В приведенной выше записи этот факт можно выразить как множество P всех задач X, для которых существует алгоритм A с полиномиальным временем выполнения, решающий X.

Эффективная сертификация

Как же формализовать идею о том, что решение задачи может быть проверено эффективно независимо от того, насколько эффективно может решаться сама задача? Структура «алгоритма проверки» для задачи X отличается от структуры алгоритма, ищущего решение; для «проверки» решения необходима входная строка s и отдельная строка t («сертификат»), доказывающая, что s является «положительным» экземпляром X.

Итак, алгоритм B называется эффективным сертифицирующим алгоритмом для задачи X, если он обладает следующими свойствами:

- B — алгоритм с полиномиальным временем, получающий два входных аргумента s и t;
- существует такая полиномиальная функция p, что для каждой строки s условие $s \in X$ выполняется в том, и только в том случае, если существует строка t, для которой $|t| \leq P(|s|)$ и B(s, t) = да.

Эффективный сертифицирующий алгоритм следует рассматривать как подход к задаче X с «управленческой» точки зрения.

Эффективный сертифицирующий алгоритм B может использоваться как центральный компонент алгоритма «грубой силы» для задачи X: для входной строки s проверить все строки t длины $\leq P(|s|)$ и проверить, выполняется ли условие B(s, t) = да для каких-либо из этих строк. Однако существование B не дает никакого очевидного способа построения эффективного алгоритма, который решает X.

NP: класс задач

Мы определим NP как множество задач, для которых существует эффективный сертифицирующий алгоритм. Операция поиска строки t, с которой эффективный сертифицирующий алгоритм примет ввод s, часто рассматривается как недетерминированный поиск по пространству возможных доказательств t; по этой причине термин NP был выбран как сокращение для «Nondeterministic Polynomial time» («недетерминированное полиномиальное время»).

Утверждение 1.10. $P \subseteq NP$.

Доказательство. Рассмотрим задачу $X \in P$; это означает, что существует алгоритм с полиномиальным временем, который решает X. Чтобы показать, что $X \in NP$, необходимо показать, что существует эффективный сертифицирующий алгоритм B для X.

Сделать это очень просто; B строится по следующей схеме. При получении входной пары (s, t) сертифицирующий алгоритм B просто возвращает значение A(s). (Считайте B своего рода «прагматиком», который игнорирует предложенное доказательство t и просто решает задачу своими силами.) Почему B является эффективным сертификатором для X? Очевидно, он выполняется с полиномиальным временем, как и A. Если строка $s \in X$, то для всех t длины не более p(|s|) имеем B(s, t) = да. С другой стороны, если $s \notin X$, то для всех t длины не более p(|s|) имеем B(s, t) = нет. ■

Нетрудно убедиться в том, что задачи, представленные в первых двух разделах, относятся к категории NP.

NP: класс задач

Как сертифицировать задачи. Например:

- для задачи 3-SAT сертификат t является результатом присваивания булевых значений переменным; сертифицирующий алгоритм B проверяет заданное множество условий в соответствии с этим присваиванием;
- для задачи о независимом множестве сертификат t является описанием множества, содержащим не менее k вершин; сертифицирующий алгоритм B проверяет, что среди этих вершин нет двух, соединенных ребром;
- для задачи покрытия множества сертификат t представляет собой список k множеств из заданного набора; сертифицирующий алгоритм проверяет, что объединение этих множеств равно базовому множеству U.

Утверждение 1.11. Присутствует ли в NP задача, не принадлежащая P? Возможно, P = NP?

Вопрос о том, истинно равенство P = NP или нет, является фундаментальной проблемой в теории алгоритмов и одной из самых известных задач в современной теории вычислений. Общественное мнение склоняется к тому, что P ≠ NP, и это считается рабочей гипотезой в этой области, однако никаких убедительных технических свидетельств для этого нет. Скорее считается, что результат P = NP выглядит слишком невероятно, чтобы быть правдой. Неужели может существовать общее преобразование задачи проверки решения в куда более сложную задачу фактического поиска решения? Неужели могут существовать общие средства проектирования эффективных алгоритмов, достаточно мощных для решения всех этих сложных задач, которые почему-то не были обнаружены?

NP-полные задачи

Не достигнув прогресса в разрешении вопроса P = NP, ученые обратились к сопутствующему, но более доступному вопросу: какие задачи в NP обладают наибольшей сложностью? Сведение к полиномиальному времени позволяет нам рассмотреть этот вопрос и получить представление о структуре NP.

Возможно, самый естественный способ определения «наиболее сложной» задачи X основан на следующих двух свойствах: **(i)** $X \in NP$ и **(ii)** для всех $Y \in NP$ $Y \leq_P X$. Другими словами, мы требуем, чтобы каждая задача в NP сводилась к X. Такая задача X будет называться NP-полной задачей.

Следующий факт подкрепляет нашу трактовку термина «наиболее сложная».

Утверждение 1.12. Допустим, X является NP-полной задачей. Тогда X решается за полиномиальное время в том, и только в том случае, если P = NP.

Доказательство. Очевидно, если P = NP, то задача X решается за полиномиальное время, так как она принадлежит NP. И наоборот, предположим, что X решается за полиномиальное время. Если Y — любая другая задача в NP, то $Y \leq_P X$, а следовательно, из (1.1) задача Y может быть решена за полиномиальное время. Отсюда $NP \subseteq P$; в сочетании с (1.10) приходим к искомому заключению. ■

Выполнимость булевой схемы: первая NP-полная задача

Наше определение NP-полноты имеет очень полезные свойства.

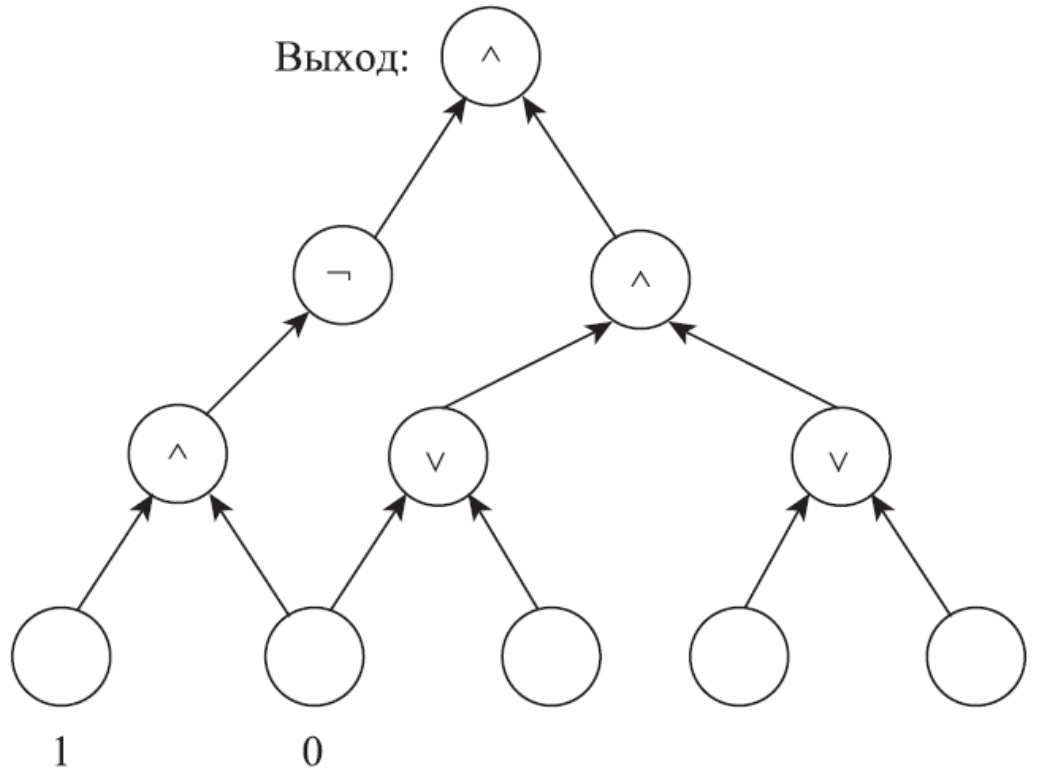
Сам факт существования NP-полных задач не так уж очевиден. Почему не могут существовать две несовместимые задачи X' и X'', для которых не существует $X \in NP$ с тем свойством, что $X' \leq_P X$ и $X'' \leq_P X$? Почему в NP не может существовать бесконечная последовательность задач X_1, X_2, X_3, ..., каждая из которых строго сложнее предыдущей? Чтобы доказать NP-полноту задачи, необходимо показать, как она может использоваться для кодирования любой задачи из NP.

В 1971 году Кук и Левин независимо показали, как это делается для очень естественных задач в NP. Возможно, самым естественным выбором для первой NP-полной задачи была задача выполнимости булевой схемы (circuit satisfiability), описанная ниже.

Чтобы дать определение задачи, необходимо четко сформулировать, что имеется в виду под «булевой схемой». Рассмотрим стандартные булевы операторы, которые используются при определении задачи выполнимости: $\wedge$ (И), $\vee$ (ИЛИ) и $\neg$ (НЕ). Наше определение строится по образцу физической схемы, собранной из логических элементов, реализующих эти операторы. Затем булева схема K определяется как помеченный, направленный ациклический граф

Выполнимость булевой схемы: первая NP-полная задача

- Источники в K (узлы, не имеющие входящих ребер) помечаются одной из констант — 0 или 1 либо именем конкретной переменной. Узлы последнего типа в дальнейшем называются входами схемы.
- Все остальные узлы помечаются одним из булевых операторов ∧, ∨ или ¬; узлы с меткой ∧ или ∨ имеют два входящих ребра, а узлы с меткой ¬ — одно входящее ребро.
- Также существует один узел, который не имеет выходящих ребер и представляет выход — результат, вычисляемый схемой. Схема вычисляет функцию своих входов следующим естественным способом.



Пример. Двум левым источникам присвоены значения 1 и 0, а следующие три источника соответствуют входам. Если подать на входы значения 1, 0, 1 (слева направо), то в логических элементах второй строки будут получены значения 0, 1, 1, в логических элементах третьей строки — значения 1, 1, и на выходе — значение 1.

Выполнимость булевой схемы: первая NP-полная задача

Задача выполнимости булевой схемы формулируется следующим образом. Для заданной схемы нужно решить, существует ли распределение значений по входам, для которого на выходе будет получено значение 1.

Теорему Кука и Левина можно рассматривать как эквивалентную следующему утверждению.

Утверждение 1.13. Задача выполнимости булевой схемы является NP-полной.

Как упоминалось выше, чтобы доказать (1.13), следует рассмотреть произвольную задачу X в NP и показать, что $X \leq_P$ ***Выполнимость булевой схемы***.

Основная идея доказательства. Мы используем тот факт, что любой алгоритм, который получает фиксированное количество n битов на входе и выдает ответ «да/нет», может быть представлен схемой только что определенного типа: такая схема эквивалентна алгоритму в том смысле, что ее результат равен 1 точно для тех входных значений, для которых алгоритм выдает ответ «да». Более того, если алгоритм выполняется за серию шагов, полиномиальных по n, то схема имеет полиномиальный размер.

Пример

Рассмотрим простой и конкретный пример. Допустим, имеется следующая задача.

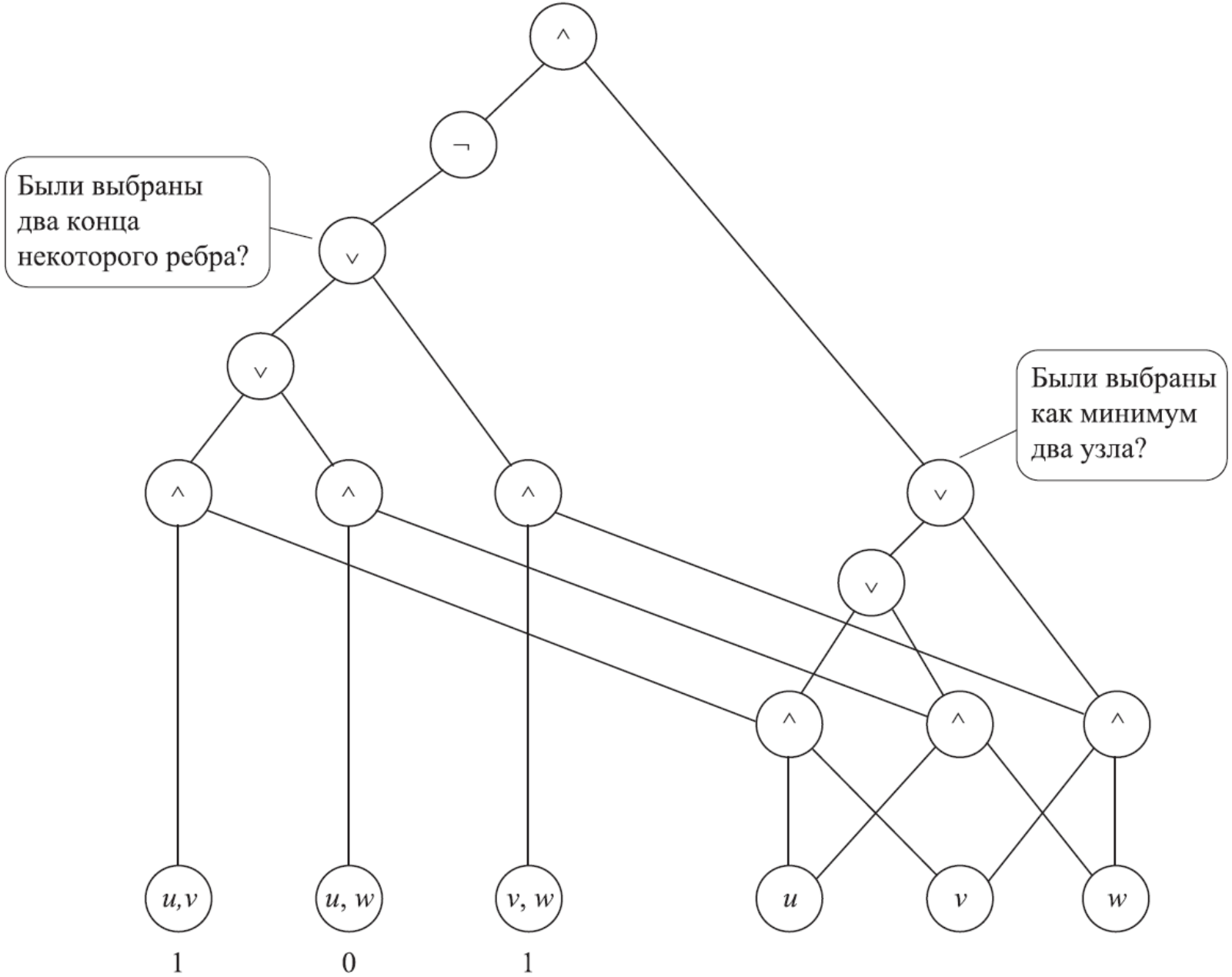
Содержит ли заданный граф G независимое множество из двух узлов?

Следуя структуре приведенного выше доказательства, мы сначала рассмотрим эффективный сертифицирующий алгоритм для этой задачи. Входные данные s представляют граф из n узлов, который будет задаваться $\binom{n}{2}$ битами: для каждой пары узлов будет присутствовать бит, указывающий, соединены ли эти два узла ребром. Сертификат t может задаваться n битами: для каждого узла включается бит, указывающий, принадлежит ли этот узел предлагаемому независимому множеству. Эффективный сертифицирующий алгоритм должен проверить два факта: что по крайней мере два бита в t равны 1 и что никакие два бита в t не равны 1, если они представляют два конца ребра (что определяется соответствующим битом в s).

Теперь для конкретной длины n, соответствующей интересующим нас входным данным s, строится эквивалентная схема K. Предположим, к примеру, что мы хотим получить ответ на задачу для графа G с тремя узлами u, v, w, в котором узел v соединен с u и w. Это означает, что мы имеем дело с вводом длины n = 3.

Пример

Схема, эквивалентная эффективному сертифицирующему алгоритму для нашей задачи с произвольным графом из трех узлов. (Фактически правая часть схемы проверяет, что были выбраны как минимум два узла, а левая — что не были выбраны оба конца любого ребра). Ребра G кодируются константами в первых трех источниках, а остальные три источника (представляющие узлы, включаемые в независимое множество) остаются переменными. Заметим, что выполнимость этого экземпляра задачи выполнимости схемы проверяется передачей 1, 0, 1 на вход. Это соответствует выбору узлов u и w, которые действительно образуют независимое множество из двух узлов в трехузловом графе G.



Доказательство NP-полноты других задач

Утверждение (1.13) открывает путь к более полному пониманию сложных задач в NP: получив первую NP-полную задачу.

Утверждение 1.14. Если Y — NP-полная задача, а X — задача в NP, обладающая свойством $Y \leq_P X$, то задача X является NP-полной.

Доказательство. Так как $X \in NP$, необходимо проверить только свойство (ii) определения. Пусть Z — любая задача в NP. Имеем $Z \leq_P Y$ вследствие NP-полноты Y и $Y \leq_P X$ по определению. Из (1.9) следует, что $Z \leq_P X$. ■

Утверждение 1.15. Задача 3-SAT является NP-полной.

Доказательство. Очевидно, задача 3-SAT принадлежит NP, так как мы можем проверить за полиномиальное время, что предложенный вариант присваивания удовлетворяет заданному набору условий. Для доказательства ее NP-полноты будет использоваться сведение $SAT \leq_P 3\text{-}SAT$.

Для заданного экземпляра задачи выполнимости булевой схемы мы сначала построим эквивалентный экземпляр SAT, в котором каждое условие содержит не более трех переменных. Затем этот экземпляр SAT будет преобразован в эквивалентный, в котором каждое условие содержит ровно три переменные. Таким образом, последний набор условий будет экземпляром 3-SAT, а следовательно, завершает сведение.

Доказательство NP-полноты других задач

Итак, рассмотрим произвольную булеву схему K. Переменная x_v связывается с каждым узлом v схемы для кодирования логического значения, содержащегося в этом узле схемы. Далее мы определим условия задачи SAT. Сначала нужно закодировать требование о том, что схема правильно вычисляет значение в каждом логическом элементе по входным значениям.

- Если узел v помечен операцией ¬ из узла u, то должно выполняться и его единственное входящее ребро выходят $x_v = \overline{x_u}$. Чтобы гарантировать это, мы добавляем два условия: $(x_v \vee x_u)$ и $(\overline{x_v} \vee \overline{x_u})$.
- Если узел v помечен операцией u и w, то должно выполняться $x_v = x_u \vee x_w$ и два его входящих ребра выходят из узлов. Чтобы гарантировать это, мы добавляем следующие условия: $(x_v \vee \overline{x_u}), (x_v \vee \overline{x_w})$ и $(\overline{x_v} \vee x_u \vee x_w)$.
- Если узел v помечен операцией ∧ и два его входящих ребра выходят из узлов u и w, то должно выполняться $x_v = x_u \wedge x_w$. Чтобы гарантировать это, мы добавляем следующие условия: $(\overline{x_v} \vee x_u), (\overline{x_v} \vee x_w)$ и $(x_v \vee \overline{x_u} \vee \overline{x_w})$.

Наконец, необходимо гарантировать, что константы в источниках имеют указанные значения, а на выходе выдается результат 1. Соответственно для источника v, помеченного константным значением, добавляется условие с одной переменной x_v или $\overline{x_v}$, под воздействием которого x_v принимает заданное значение. Для выходного узла o добавляется условие с одной переменной x_o, которое требует, чтобы значение o было равно 1. На этом построение

Доказательство NP-полноты других задач

Не так сложно показать, что только что построенный экземпляр SAT эквивалентен заданному экземпляру задачи выполнимости булевой схемы. Чтобы продемонстрировать эквивалентность, необходимо привести два обоснования. Во-первых, предположим, что заданная схема K выполнима. Выполняющее присваивание входам схемы расширяется для создания значений во всех узлах K. Это множество значений очевидным образом обеспечивает выполнимость построенного экземпляра SAT.

Далее предположим, что построенный экземпляр SAT является выполнимым. Рассмотрим выполняющее присваивание в этом экземпляре и значения переменных, соответствующие входам схемы K. Утверждается, что эти значения образуют выполняющее присваивание для схемы K. Чтобы убедиться в этом, достаточно заметить, что условия SAT гарантируют, что значения, присвоенные всем узлам K, совпадают с теми, которые вычисляются схемой для этих узлов. В частности, выходу будет присвоено значение 1, поэтому присваивание значений входам обеспечивает выполнимость K. Итак, мы показали, как создать экземпляр SAT, эквивалентный задаче выполнимости булевой схемы. Но работа еще не завершена, так как нашей целью является создание экземпляра 3-SAT, который требует, чтобы длина всех условий была равна в точности 3, — а в созданном нами экземпляре некоторые условия имеют длину 1 или 2. Итак, для завершения доказательства необходимо преобразовать этот экземпляр SAT в эквивалентный экземпляр, в котором каждое условие состоит ровно из трех переменных.

Доказательство NP-полноты других задач

Для этого мы введем четыре новые переменные: z_1, z_2, z_3, z_4. Идея заключается в том, чтобы гарантировать, что для каждого выполняющего присваивания $z_1 = z_2 = 0$; для этого добавляются условия $(\bar{z}_1 \vee z_3 \vee z_4), (\bar{z}_1 \vee \bar{z}_3 \vee z_4), (\bar{z}_1 \vee z_3 \vee \bar{z}_4)$ и $(\bar{z}_1 \vee \bar{z}_3 \vee \bar{z}_4)$ для i = 1 и i = 2. Обратите внимание: выполнение всех этих условий и возможно только в том случае, если $z_1 = z_2 = 0$.

Теперь рассмотрим условие в построенном экземпляре SAT, которое содержит один литерал t (которым может быть переменная или отрицание переменной). Каждый такой литерал заменяется условием $(t \vee z_1 \vee z_2)$. Аналогичным образом каждое условие, содержащее два литерала (допустим, t ∨ t'), заменяется условием $(t \vee t' \vee z_1)$. Полученная формула 3-SAT очевидно эквивалентна формуле SAT, содержащей не более трех переменных в каждом условии, что завершает доказательство. ■

Используя этот результат и последовательность сведений

3-SAT $\leq_P$ Независимое множество $\leq_P$ Вершинное покрытие $\leq_P$ Покрытие множества, мы через (1.14) приходим к следующему выводу:

Утверждение 1.16. Все следующие задачи являются NP-полными: задача о независимом множестве, задача упаковки множества, задача о вершинном покрытии и задача покрытия множества.

Доказательство. Каждая из этих задач обладает тем свойством, что она принадлежит NP, и задача 3-SAT (а следовательно, и задача SAT) может быть сведена к ней. ■

Общая стратегия доказательства NP-полноты новых задач

Основная стратегия доказательства NP-полноты новой задачи X выглядит примерно так:
1. Доказать, что $X \in NP$.
2. Выбрать задачу Y, которая заведомо является NP-полной.
3. Доказать, что $Y \leq_P X$.

Ранее было замечено, что многие сведения $Y \leq_P X$ состоят из преобразования заданного экземпляра Y в экземпляр X с тем же ответом. Для решения X используется одно обращение к «черному ящику». При использовании подобных сведений приведенная выше стратегия превращается в следующий план доказательства NP-полноты.

1. Доказать, что $X \in NP$.
2. Выбрать задачу Y, которая заведомо является NP-полной.
3. Рассмотреть произвольный экземпляр s_Y задачи Y и показать, как построить за полиномиальное время экземпляр s_X задачи X, обладающий следующими свойствами.

a. Если s_Y является экземпляром Y, на который дается ответ «да», то и s_X является экземпляром X, на который дается ответ «да».
b. Если s_X является экземпляром X, на который дается ответ «да», то и s_Y является экземпляром Y, на который дается ответ «да».

Другими словами, тем самым устанавливается, что s_Y и s_X имеют одинаковый ответ.

Задачи упорядочения

До настоящего момента рассматривались задачи, которые (как, например, задачи о независимом множестве и вершинном покрытии) подразумевали перебор подмножеств набора объектов; также были представлены задачи (как, например, 3-SAT), основанные на поиске комбинаций 0/1 в наборе переменных. Еще один тип вычислительно сложных задач связан с поиском по множеству всех перестановок коллекции объектов.

Задача коммивояжера

Представьте коммивояжера, который должен посетить n городов v_1, v_2, ..., v_n. Коммивояжер начинает с города v_1, в котором он живет, и хочет найти маршрут — порядок, в котором он посетит все остальные города и вернется домой. Требуется найти маршрут с минимальным суммарным расстоянием всех поездок.

Чтобы формализовать эту задачу, мы воспользуемся предельно обобщенной концепцией расстояния: для каждой упорядоченной пары городов (v_i,v_j) задается неотрицательное число $d(v_i,v_j)$, которое считается расстоянием от v_i до v_j. От расстояний не требуется ни симметричность (может оказаться, что $d(v_i,v_j) \neq d(v_j,v_i)$), ни выполнение неравенства треугольника (сумма $d(v_i,v_j)$ и $d(v_j,v_k)$ может быть меньше «прямого» расстояния $d(v_i,v_k)$). Это делается для того, чтобы формулировка задачи была как можно более общей

Формулировка задачи коммивояжера применяется: планирование оптимальных перемещений манипулятора; для обслуживания запросов ввода/вывода к диску; или для упорядочения выполнения n программных модулей для минимизации времени переключения контекста. Итак, для заданного множества расстояний требуется упорядочить города в маршрут $v_{i_1}, v_{i_2}, \ldots, v_{i_n}$, с $i_1 = 1$, чтобы свести к минимуму общее расстояние $\sum_j d(v_{i_j}, v_{i_{j+1}}) + d(v_{i_n}, v_{i_1})$. Требование $i_1 = 1$ просто «ориентирует» маршрут так, чтобы он начинался в исходном городе, а слагаемые в сумме просто задают расстояние от текущего города в маршруте до следующего.

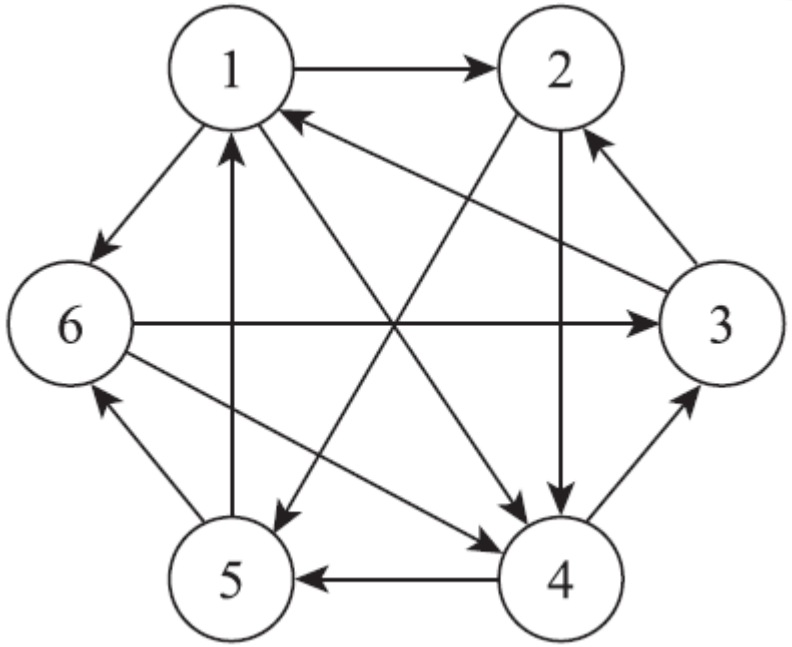
А вот как выглядит задача коммивояжера в версии с принятием решения:

Для заданного набора расстояний между n городами и границы D существует ли маршрут, длина которого не превышает D?

Задача о гамильтоновом цикле

У задачи коммивояжера имеется естественная аналогия, которая образует одну из фундаментальных задач в теории графов. Для заданного направленного графа G = (V, E) цикл C в графе G называется гамильтоновым циклом, если он посещает каждую вершину ровно один раз, — иначе говоря, он «обходит» все вершины без повторений. Например, направленный граф содержит несколько гамильтоновых циклов; в одном из них узлы посещаются в порядке 1, 6, 4, 3, 2, 5, 1, а в другом — в порядке 1, 2, 4, 5, 6, 3, 1.

Задача о гамильтоновом цикле формулируется просто: Содержит ли заданный направленный граф G гамильтонов цикл?



Доказательство NP-полноты задачи о гамильтоновом цикле

Утверждение 1.17. Задача о гамильтоновом цикле является NP-полной.

Доказательство. Сначала мы покажем, что задача о гамильтоновом цикле принадлежит NP. Для направленного графа G = (V, E) сертификатом, подтверждающим наличие решения, является упорядоченный список вершин гамильтонового цикла. По этому списку можно проверить за полиномиальное время, что каждая вершина входит в список ровно один раз, а каждая последовательная пара соединена ребром; эти проверки установят, что упорядочение определяет гамильтонов цикл.

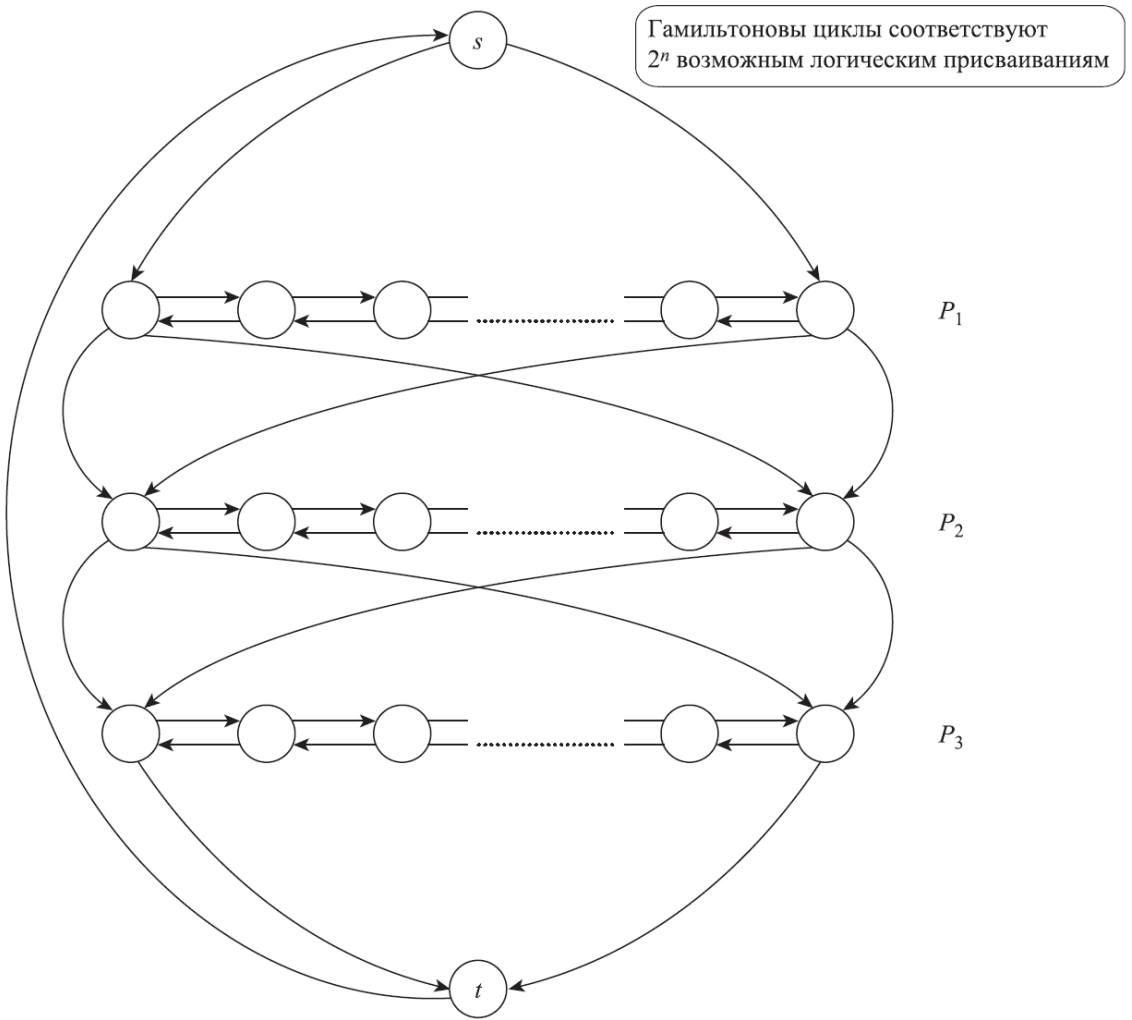
Покажем, что 3-SAT $\leq_P$ Гамильтонов цикл. Почему мы выполняем сведение к 3-SAT? Столкнувшись с задачей о гамильтоновом цикле, мы на самом деле понятия не имеем, что выбрать для сведения; задача достаточно сильно отличается от всех задач, которые мы видели до сих пор, так что реальной основы для выбора нет. В такой ситуации одна из возможных стратегий заключается в возврате к 3-SAT, потому что комбинаторная структура этой задачи очень проста. Конечно, эта стратегия гарантирует по крайней мере некоторый уровень сложности при сведении, потому что переменные и условия необходимо будет выразить на языке графов. Итак, рассмотрим произвольный экземпляр 3-SAT с переменными x_1, ..., x_n и условиями C_1, ..., C_k. Необходимо показать, как решить эту задачу, если имеется возможность обнаружения гамильтоновых циклов в направленных графах. Как обычно, стоит сосредоточиться на важнейших ингредиентах 3-SAT: значения переменных можно задать по своему усмотрению, и для выполнения каждого условия есть три возможности.

Доказательство NP-полноты задачи о гамильтоновом цикле

Начнем с описания графа, содержащего 2^n разных гамильтоновых циклов, естественно соответствующих возможным 2^n логическим присваиваниям значений переменных. После этого мы добавим узлы для моделирования ограничений, накладываемых условиями.

Мы строим n путей P_1, ..., P_n, где P_i состоит из узлов v_{i1}, v_{i2}, ..., v_{ib} для величины b, которая, как предполагается, немного больше количества условий k; допустим, $b = 3k + 3$. Существуют ребра из v_{ij} к $v_{i,j+1}$ и в обратном направлении, от $v_{i,j+1}$ к v_{ij}. Таким образом, обход P_i может осуществляться «слева направо» от v_{i1} к v_{ib}, или «справа налево», от v_{ib} к v_{i1}.

Эти пути связываются следующим образом: для всех $i = 1, 2, ..., n - 1$ мы определяем ребра из v_{i1} в $v_{i+1,1}$ и в $v_{i+1,b}$. Также определяются ребра из v_{ib} в $v_{i+1,1}$ и $v_{i+1,b}$. В граф добавляются два дополнительных узла s и t; мы определяем ребра из s в v_{11} и v_{1b}; из v_{n1} и v_{nb} в t; и из t в s. Построение до настоящего момента изображено на рис.



Доказательство NP-полноты задачи о гамильтоновом цикле

Здесь важно задержаться и подумать, как выглядят гамильтоновы циклы на нашем графе. Так как из t выходит только одно ребро, мы знаем, что любой гамильтонов цикл C должен использовать ребро (t, s). После входа в s цикл C может идти по пути P_1 либо слева направо, либо справа налево; какое бы направление ни было выбрано, затем он идет по пути P_2 либо слева направо, либо справа налево; и т. д., вплоть до завершения P_n и входа в t. Другими словами, существуют ровно 2^n разных гамильтоновых циклов, и они соответствуют n независимым выборам направления обхода каждого P_i.

Эта структура, естественно, моделирует n независимых вариантов присваивания значений переменных x_1, ..., x_n в экземпляре 3-SAT. Следовательно, каждый гамильтонов цикл будет однозначно идентифицироваться следующим логическим присваиванием: если C проходит P_i слева направо, то x_i присваивается 1; в противном случае x_i присваивается 0.

Теперь добавим узлы для моделирования условий; экземпляр 3-SAT является выполнимым в том, и только в том случае, если останется хотя бы один гамильтонов цикл. Рассмотрим конкретный пример — условие

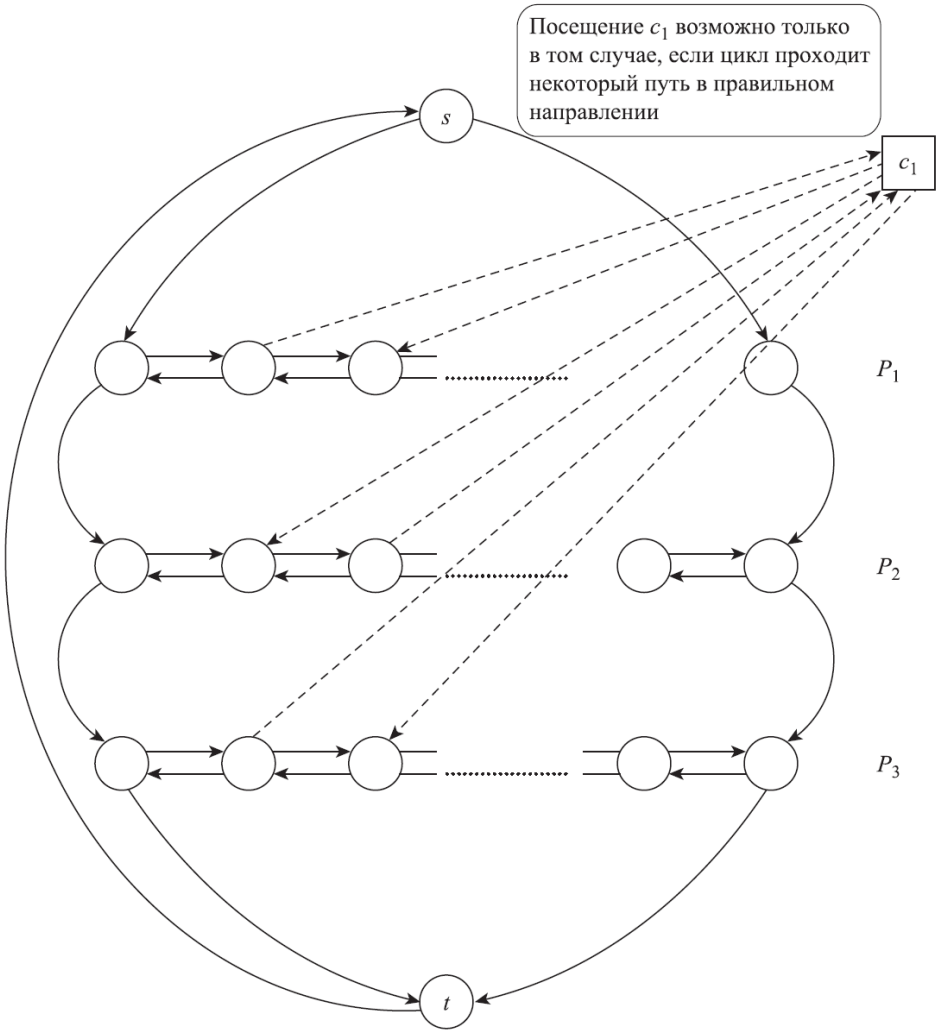
$$C_1 = x_1 \vee x_2 \vee x_3.$$

На языке гамильтоновых циклов это условие означает: «Цикл должен проходить P_1 слева направо; или он должен проходить P_2 справа налево; или он должен проходить P_3 слева направо».

Доказательство NP-полноты задачи о гамильтоновом цикле

Итак, мы добавляем узел c_1 (рис.), который делает именно это. Для некоторого значения l узел c_1 имеет ребра из v_{1l}, $v_{2,l+1}$, и v_{3l}, а также ребра в $v_{1,l+1}$, $v_{2,l}$, и $v_{3,l+1}$. Следовательно, он легко вставляется в любой гамильтонов цикл, который проходит P_1 слева направо, для чего узел c_1 посещается между v_{1l} и $v_{1,l+1}$; аналогичным образом c_1 может быть вставлен в любой гамильтонов цикл, проходящий P_2 справа налево или P_3 слева направо. Он не может быть вставлен в гамильтонов цикл, который не делает ничего из перечисленного.

На более общем уровне узел c_j определяется для каждого условия C_j. В каждом пути P_i позиции узлов 3j и 3j + 1 резервируются для переменных, участвующих в условии C_j. Предположим, условие C_j содержит литерал t. Если $t = x_i$, то добавляются ребра $(v_{i,3j}, c_j)$ и $(c_j, v_{i,3j+1})$; если $t = \bar{x}_i$, то добавляются ребра $(v_{i,3j+1}, c_j)$ и $(c_j, v_{i,3j})$. На этом построение графа G завершается. Далее, в соответствии с приведенной выше общей схемой доказательства NP-полноты, мы утверждаем, что экземпляр 3-SAT выполним в том, и только в том случае, если G содержит гамильтонов цикл.



Доказательство NP-полноты задачи о гамильтоновом цикле

Для начала предположим, что для экземпляра 3-SAT существует выполняющее присваивание; тогда мы определяем гамильтонов цикл в соответствии с приведенным выше неформальным планом. Если x_i в выполняющем присваивании задается значение 1, то путь P_i обходится слева направо; в противном случае обход производится справа налево. Для каждого условия C_j, так как оно выполняется вследствие присваивания, будет по крайней мере один путь P_i, в котором обход будет осуществляться в «правильном» направлении относительно узла c_j, и его можно вставить в маршрут по ребрам, инцидентным $v_{i,3j}$ и $v_{i,3j+1}$.

И наоборот, предположим, что в G существует гамильтонов цикл C. Здесь важно заметить следующее: если C входит в узел c_j по ребру из $v_{i,3j}$, то он должен выходить по ребру в $v_{i,3j+1}$. В противном случае у $v_{i,3j+1}$ останется только один непосещенный сосед, а именно $v_{i,3j+2}$, поэтому маршрут не сможет посетить этот узел и сохранить гамильтоново свойство. И симметрично, если маршрут входит из $v_{i,3j+1}$, он должен немедленно выйти в $v_{i,3j}$. Следовательно, для каждого узла c_j узлы, находящиеся непосредственно до и после c_j в цикле C, соединяются ребром e в G; следовательно, если удалить c_j из цикла и вставить это ребро e для каждого j, мы получим гамильтонов цикл C' для подграфа $G - \{c_1, ..., c_k\}$. Это наш исходный подграф, до добавления узлов условий; как упоминалось ранее, любой гамильтонов цикл в этом подграфе должен полностью пройти каждый путь P_i в том или ином направлении. Соответственно мы используем C' для определения следующего логического присваивания для экземпляра 3-SAT. Если C' проходит P_i слева направо, то мы задаем $x_i = 1$; в противном случае задается $x_i = 0$. Так как больший цикл C смог посетить узел каждого условия c_j, по крайней мере один из путей проходился в «правильном» направлении относительно узла c_j, поэтому с определенным нами присваиванием выполняются все условия.

Установлено, что экземпляр 3-SAT выполним в том, и только в том случае, если G содержит гамильтонов цикл; на этом наше доказательство завершается.

Доказательство NP-полноты задачи коммивояжера

Вооружившись базовым результатом сложности задачи о гамильтоновом цикле, мы можем перейти к демонстрации сложности задачи коммивояжера.

Утверждение 1.18. Задача коммивояжера является NP-полной.

Доказательство. Легко увидеть, что задача коммивояжера принадлежит NP: сертификат представляет собой перестановку городов, а сертифицирующий алгоритм проверяет, что длина соответствующего маршрута не превышает заданной границы.

Теперь покажем, что Гамильтонов цикл $\leq_P$ Коммивояжер. Для заданного направленного графа $G = (V,E)$ определяется следующий экземпляр задачи коммивояжера. Каждому узлу v_i графа G соответствует город v'_i. Мы определяем $d(v'_i,v'_j)$ равным 1, если в G существует ребро (v_i,v_j), и 2 — в противном случае.

Утверждается, что G содержит гамильтонов цикл в том, и только в том случае, если в экземпляре задачи коммивояжера имеется маршрут длины не более n. Если G содержит гамильтонов цикл, то упорядочение соответствующих городов определяет маршрут длины n. И наоборот, предположим, что существует маршрут длины не более n. Выражение для длины маршрута представляет собой сумму n слагаемых, каждое из которых не менее 1; следовательно, все слагаемые должны быть равны 1. Следовательно, каждая пара узлов G, соответствующих соседним городам маршрута, должна быть соединена ребром; из этого следует, что упорядочение соответствующих узлов должно образовать гамильтонов цикл. ■

Расширения: задача о гамильтоновом пути

Для заданного направленного графа G = (V, E) путь P в G называется гамильтоновым путем, если каждая вершине входит в него ровно один раз. Такой путь состоит из разных узлов $v_{i_1}, v_{i_2}, \dots, v_{i_n}$ в определенном порядке, образующих все множество вершин V; в отличие от гамильтонова цикла, наличие ребра из v_{i_n} обратно в v_{i_1} не обязательно. Задача о гамильтоновом пути формулируется так:

Содержит ли заданный направленный граф G гамильтонов путь?

Утверждение 1.19. Задача о гамильтоновом пути является NP-полной.

Доказательство. Прежде всего задача о гамильтоновом пути принадлежит NP: сертификатом может быть путь в G, а сертифицирующий алгоритм проверяет, что он действительно является путем и каждый узел входит в него ровно один раз.

Один из способов демонстрации NP-полноты задачи о гамильтоновом пути заключается в сведении от 3-SAT, почти идентичного тому, которое мы использовали для задачи о гамильтоновом цикле: мы строим граф, с тем отличием, что в него не включается ребро из t в s. Если в измененном графе нет ни одного гамильтонова пути, он должен начинаться в s (так как s не имеет входящих ребер) и завершаться в t (так как t не имеет выходящих ребер).

Альтернативный способ демонстрации NP-полноты гамильтонова пути основан на доказательстве Гамильтонов цикл $\leq_P$ Гамильтонов путь.

Задачи о разбиении

Фундаментальные задачи о разбиении - в которых ищутся способы разбиения коллекции объектов на подмножества. Сейчас мы продемонстрируем NP-полноту задачи, которая будет называться задачей о трехмерном сочетании.

Задача о трехмерном сочетании

Начнем с обсуждения задачи о трехмерном сочетании, которая может считаться усложненной версией задачи о двудольном паросочетании. Задача о двудольном паросочетании может рассматриваться следующим образом: даны два множества X и Y, каждое из которых имеет размер n, и множество P пар из X × Y. Вопрос: существует ли в P такое множество из n пар, в котором каждый элемент в X ∪ Y содержится ровно в одной паре? Связь с задачей двудольного паросочетания очевидна: множество P таких пар просто соответствует ребрам двудольного графа.

Мы уже знаем, как решать задачу двудольного паросочетания за полиномиальное время. Однако ситуация заметно усложняется при переходе от упорядоченных пар к упорядоченным триплетам. Рассмотрим следующую задачу о трехмерном сочетании:

Для заданных непересекающихся множеств X, Y и Z, каждое из которых имеет размер n, и заданного множества T ⊆ X × Y × Z упорядоченных триплетов существует ли в T такое множество из n триплетов, что каждый элемент X ∪ Y ∪ Z содержится ровно в одном из этих триплетов?

Такой набор триплетов называется идеальным трехмерным сочетанием.

Конкретнее, задача о трехмерном сочетании является частным случаем задачи покрытия множества, поскольку мы ищем покрытие универсального множества U = X ∪ Y ∪ Z с использованием не более n множеств из заданного набора (триплетов).

Доказательство NP-полноты трехмерного сочетания

Приведенные выше рассуждения легко преобразуются в доказательства того, что ***Трехмерное сочетание*** $\leq_P$ ***Покрытие множества*** и **Трехмерное сочетание** $\leq_P$ **Упаковка множества**. Но это не поможет установить NP-полноту задачи о трехмерном сочетании, потому что эти сведения просто показывают, что задача о трехмерном сочетании может быть сведена к некоторым очень сложным задачам. Нам нужно провести доказательство в обратном направлении: что известная NP-полная задача может быть сведена к задаче о трехмерном сочетании.

Утверждение 1.20. Задача о трехмерном сочетании является NP-полной.

Доказательство. Как и следовало ожидать, принадлежность задачи о трехмерном сочетании NP доказывается легко. Для заданного набора триплетов $T \subset X \times Y \times Z$ сертификатом, подтверждающим решение, может быть набор триплетов $T' \subseteq T$. За полиномиальное время можно убедиться в том, что каждый элемент в $X \cup Y \cup Z$ принадлежит ровно одному из триплетов в T'.

Для сведения мы снова возвращаемся к задаче 3-SAT. Итак, рассмотрим произвольный экземпляр 3-SAT с n переменными $x_1, ..., x_n$ и k условиями $C_1, ..., C_k$. Мы покажем, как решить его, если существует возможность обнаружения идеальных трехмерных сочетаний.

Общая стратегия такого сведения очень похожа (на очень высоком уровне) на метод, который использовался в сведении 3-SAT к гамильтоновому циклу.

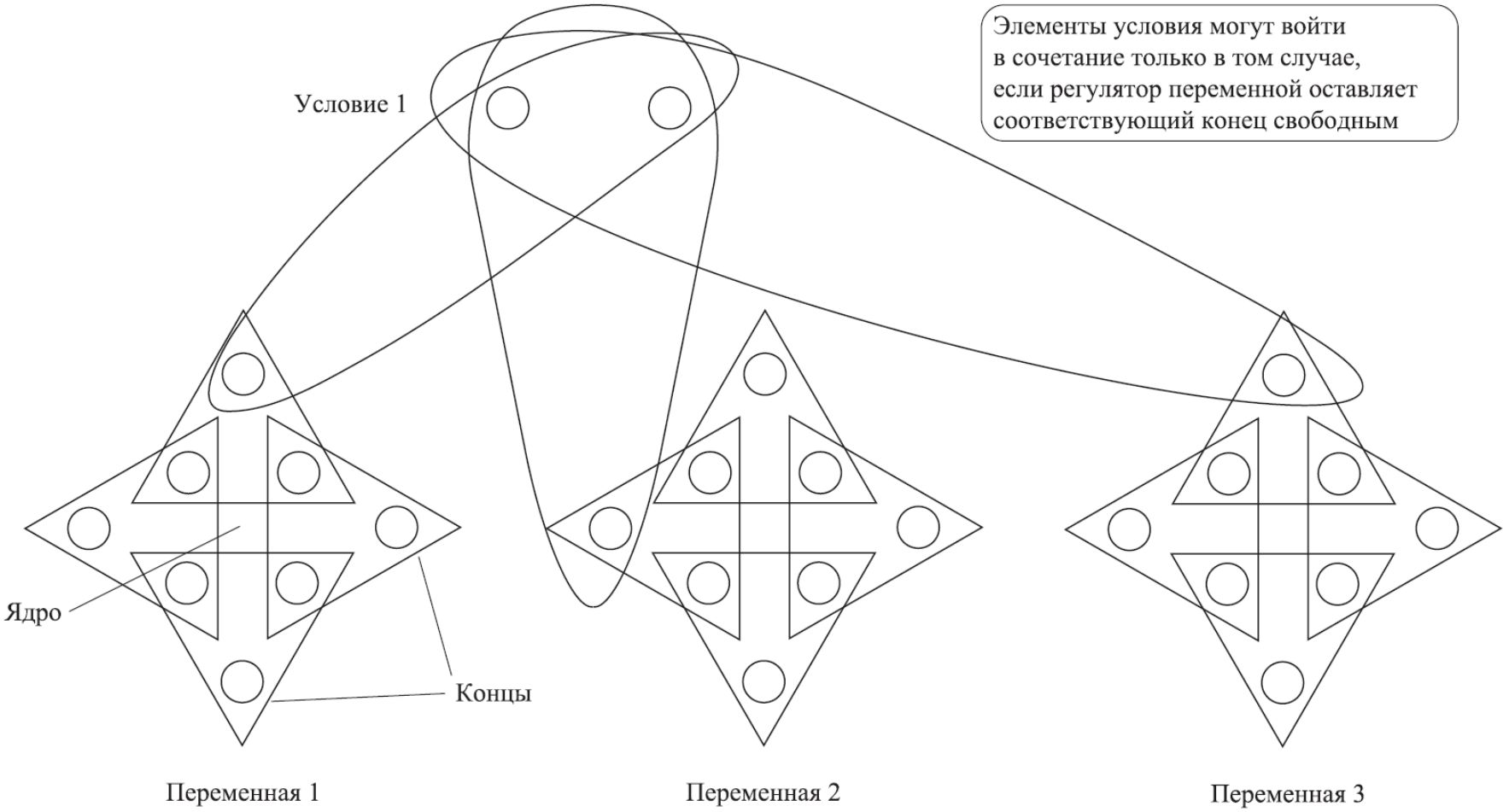
Доказательство NP-полноты трехмерного сочетания

Сначала мы спроектируем регуляторы для кодирования независимых вариантов выбора, задействованных в логическом присваивании каждой переменной; затем будут добавлены регуляторы для кодирования ограничений, наложенных условиями. В этом построении все элементы экземпляра трехмерного сочетания будут изначально называться просто «элементами» без указания того, берутся ли они из X, Y или Z. В конце они естественным образом раскладываются на эти три множества. Рассмотрим базовый регулятор, связанный с переменной x_i. Определим элементы $A_i = \{a_{i1}, a_{i2}, ..., a_{i,2k}\}$, образующие ядро регулятора; определим элементы $B_i = \{b_{i1}, ..., b_{i,2k}\}$ на концах регулятора. Для всех $j = 1, 2, ..., 2k$ определяется триплет $t_{ij} = (a_{ij}, a_{i\ j+1}, b_{ij})$ с интерпретацией сложения по модулю 2k.

Доказательство NP-полноты трехмерного сочетания

Три таких регулятора изображены на рис. В регуляторе i триплет t_{ij} называется четным, если j четно, или нечетным, если j нечетно. Аналогичным образом конец b_{ij} будет называться четным или нечетным.

Это будут единственные триплеты, содержащие элементы A_i, поэтому мы уже можем что-то сказать о том, как они должны покрываться в любом идеальном сочетании: необходимо использовать либо все четные триплеты в регуляторе i, либо все нечетные триплеты в гаджете i.



Доказательство NP-полноты трехмерного сочетания

На этой идее основан наш метод кодирования идеи о том, что x_i задается значение 0 или 1; выбор всех четных триплетов представляет назначение $x_i = 0$, а выбор всех нечетных триплетов представляет назначение $x_i = 1$. Решение о четности/нечетности также может рассматриваться с другой точки зрения — в контексте концов регуляторов. Решив использовать четные триплеты, мы покрываем четные концы регуляторов и оставляем нечетные концы свободными; для нечетных триплетов покрываются четные концы регуляторов, а нечетные концы остаются свободными. Таким образом, решение о присваивании x_i можно рассматривать следующим образом: свободные нечетные концы соответствуют 0, тогда как свободные четные концы соответствуют 1. Пожалуй, такое представление упростит понимание оставшейся части построения.

Пока что выбор четности/нечетности может приниматься независимо для каждого из n регуляторов переменных. Теперь добавим элементы для моделирования условий и ограничения выбираемых вариантов присваивания. Как и в доказательстве (1.17), рассмотрим пример условия

$$C_1 = x_1 \vee \bar{x}_2 \vee x_3.$$

На языке трехмерных сочетаний это означает: «Сочетание по центрам регуляторов должно оставить свободными четные концы первого регулятора; или оно должно оставить свободными нечетные концы второго регулятора; или оно должно оставить свободными нечетные концы третьего регулятора».

Доказательство NP-полноты трехмерного сочетания

Итак, мы добавляем регулятор условия, который делает именно это. Он состоит из множества двух элементов ядра $P_1 = \{p_1, p'_1\}$ и трех триплетов, их содержащих. Один триплет имеет форму (p_1, p'_1, b_{1j}) для четного конца b_{1j}; другой включает p_1, p'_1 и нечетный конец $b_{2,j}$'; и третий включает p_1, p'_1 и четный конец $b_{3,j}$''. Только эти три триплета покрывают P_1, поэтому мы знаем, что один из них должен использоваться; тем самым точно обеспечивается соблюдение условия.

В общем случае для условия C_j создается регулятор с двумя элементами ядра $P_j = \{p_j, p'_j\}$, а также определяются три триплета, содержащие P_j. Предположим, условие C_j содержит литерал t. Если $t = x_i$, мы определяем триплет $(p_j, p'_j, b_{i,2j})$; если $t = \bar{x}_i$, то определяется триплет $(p_j, p'_j, b_{i,2j-1})$. Только регулятор условия j использует концы b_{im} с $m = 2j$ или $m = 2j - 1$; таким образом, регуляторы условий никогда не «конкурируют» друг с другом за свободные концы.

Построение почти закончено, но осталась еще одна проблема. Допустим, множество условий имеет выполняющее присваивание. В этом случае для каждого регулятора переменной принимается соответствующий выбор четности/нечетности; для каждого регулятора условия остается как минимум один свободный конец, так что все элементы ядер регуляторов условий также получают покрытие. Проблема в том, что покрытие получили еще не все концы. Мы начали с $n \cdot 2k = 2nk$ концов; триплеты $\{t_{ij}\}$ обеспечили покрытие nk из них, а регуляторы условий — еще k. Остается обеспечить покрытие еще $(n - 1)k$ концов.

Доказательство NP-полноты трехмерного сочетания

Проблема решается очень простым приемом: в конструкцию добавляются (n − 1)k «регуляторов завершения». Регулятор завершения i состоит из двух элементов ядра $Q_i = \{q_i, q'_i\}$ и триплета $(q_i, q_{i,b})$ для каждого конца b в каждом регуляторе переменной. Это последняя часть построения.

Итак, если множество условий имеет выполняющее присваивание, то в каждом регуляторе переменной делается соответствующий выбор четности/нечетности; как и прежде, при этом остается как минимум один свободный конец для каждого регулятора условия. Использование регуляторов завершения для покрытия всех оставшихся концов гарантирует, что покрытие обеспечено для всех элементов ядер в регуляторах переменных, условий и завершения, а также для всех концов. И наоборот, предположим, что в построенном экземпляре существует идеальное трехмерное сочетание. Тогда, как упоминалось выше, в каждом регуляторе переменной сочетание выбирает либо все четные $\{t_{ij}\}$, либо все нечетные $\{t_{ij}\}$. В первом случае в экземпляре 3-SAT задаются $x_i = 0$, а во втором — $x_i = 1$. Теперь рассмотрим условие C_j; было ли оно выполнено? Так как два элемента ядра в P_j получили покрытие, по крайней мере один из трех регуляторов переменных, соответствующих литералам из C_j, сделал «правильный» выбор решения четности/нечетности, что привело к присваиванию, удовлетворяющему C_j.

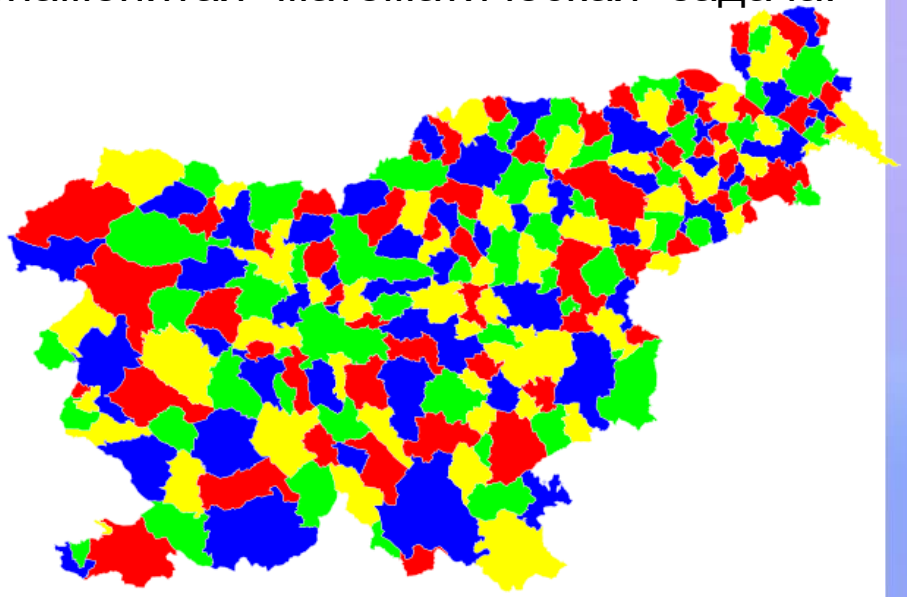
Доказательство NP-полноты трехмерного сочетания

Доказательство практически завершено, осталось ответить на последний вопрос: действительно ли мы сконструировали экземпляр трехмерного сочетания? Имеется набор элементов и триплеты, содержащие некоторые из них, но действительно ли эти элементы можно разбить на соответствующие множества X, Y и Z равного размера?

К счастью, ответ на этот вопрос положителен. Мы можем определить X как множество всех a_{ij} с четными j, множество всех p_j и множество всех q_i. Y определяется как множество всех a_{ij} с нечетными j, множество всех p'_j и множество всех q'_i. Наконец, Z определяется как множество всех концов b_{ij}. Легко убедиться в том, что каждый триплет содержит по одному элементу из множеств X, Y и Z. ■

Задачи о разбиении

При раскрашивании карты (например, стран на карте мира) желательно раскрасить соседние области в разные цвета, чтобы их границы были четко видны, — но при этом, чтобы избежать лишней пестроты, использовать минимальное количество цветов. В середине XIX века Фрэнсис Гатри заметил, что карту графств Англии можно раскрасить таким образом всего четырьмя цветами, и его заинтересовало, обладает ли этим свойством любая карта. Он спросил своего брата, который передал вопрос одному из своих профессоров; так родилась знаменитая математическая задача: гипотеза четырех цветов.



Задача о раскраске графа

Под раскраской графа понимается аналогичный процесс с ненаправленным графом G, в котором узлы играют роль раскрашиваемых областей, а ребра представляют соседние пары. Требуется назначить цвет каждому узлу G так, чтобы при наличии ребра (u, v) узлам u и v были назначены разные цвета; целью является выполнение этих условий с минимальным набором цветов. В более формальном варианте k-раскраской G называется такая функция $f:V \rightarrow \{1, 2, ..., k\}$, что для каждого ребра (u, v) выполняется f (u) = f (v). (Таким образом, цвета обозначаются 1, 2, ..., k, а функция f представляет выбор цвета для каждого узла.) Если для G существует k-раскраска, он называется k-раскрашиваемым.

В отличие от карт на плоскости, вполне очевидно, что не существует фиксированной константы k, с которой любой граф имеет k-раскраску: например, если взять множество из n узлов и соединить каждую пару ребром, то для раскраски полученного графа потребуется n цветов. Тем не менее алгоритмическая версия задачи очень интересна:

Для заданного графа G и границы k имеет ли граф G k-раскраску?

Будем называть эту задачу задачей раскраски графа — или k-раскраски, когда мы хотим подчеркнуть конкретный выбор k.

Задача раскраски графа находит очень широкий диапазон применения. Хотя реальная потребность в ней в картографии пока неочевидна, эта задача естественным образом возникает в ситуациях с распределением ресурсов при наличии конфликтов.

Задача о раскраске графа

- Допустим, имеется набор из n процессов в системе, которая позволяет выполнять несколько заданий в параллельном режиме. При этом некоторые пары заданий не могут выполняться одновременно, потому что им нужен доступ к некоторому ресурсу. Для следующих k временных квантов требуется спланировать выполнение процессов, чтобы каждый процесс выполнялся минимум в одном из них. Возможно ли это? Если построить граф G на базе множества процессов, соединяя два процесса ребром при наличии конфликта, то k-раскраска G будет представлять план выполнения, свободный от конфликтов: все узлы, окрашенные в цвет j, будут выполняться на шаге j, а вся конкуренция за ресурсы будет исключена.
- Другое известное применение задачи встречается при разработке компиляторов. Предположим, при компиляции программы мы пытаемся связать каждую переменную с одним из k регистров. Если две переменные используются одновременно, они не могут быть связаны с одним регистром (в противном случае присваивание одной переменной приведет к потере значения другой). Для множества переменных строится граф G; две переменные соединяются ребром в том случае, если они используются одновременно. K-раскраска G соответствует безопасному способу распределения переменных между регистрами: все узлы с раскраской j могут быть связаны с регистром j, так как никакие два из них не используются одновременно.

Третий пример встречается при распределении частот для беспроводной связи: требуется назначить одну из k частот каждому из n устройств; если два устройства находятся достаточно близко друг к другу, им должны быть присвоены разные длины волн для предотвращения помех.

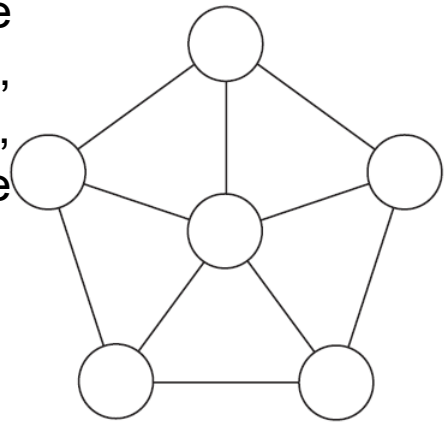
Вычислительная сложность задачи о раскраске графа

Какую сложность имеет задача k-раскраски?

Утверждение 1.21. Граф G является 2-раскрашиваемым в том, и только в том случае, если он является двудольным.

Это означает, что алгоритм может использоваться для принятия решения о том, является ли входной граф 2-раскрашиваемым, за время O(m + n), где n — количество узлов G, а m — количество ребер.

Стоит перейти к k = 3 цветам, как ситуация значительно усложняется. Никакого простого эффективного алгоритма для задачи 3-раскраски не видно, и рассуждать об этой задаче вообще сложно. Например, изначально может возникнуть впечатление, что в любом графе, не являющиеся 3-раскрашиваемым, присутствует «доказательство» в форме четырех узлов, являющихся взаимно смежными (для которых потребуются четыре разных цвета) — но это не так.



Доказательство NP-полноты задачи о 3-раскраске

Утверждение 1.22. Задача о 3-раскраске является NP-полной.

Доказательство. Легко увидеть, почему задача принадлежит NP. Для заданных G и k одним из сертификатов, подтверждающих истинность ответа, является k-раскраска: за полиномиальное время можно убедиться в том, что в раскраске используется не более k цветов и никакая пара узлов, соединенных ребром, не окрашена в один цвет.

Как и другие задачи в этом разделе, задачу о 3-раскраске графа трудно связать с другой NP-полной задачей, виденной ранее, поэтому мы снова вернемся к 3-SAT. Заданный экземпляр 3-SAT с переменными x_1, ..., x_n и условиями C_1, ..., C_k будет решен с использованием «черного ящика» для решения задачи 3-раскраски.

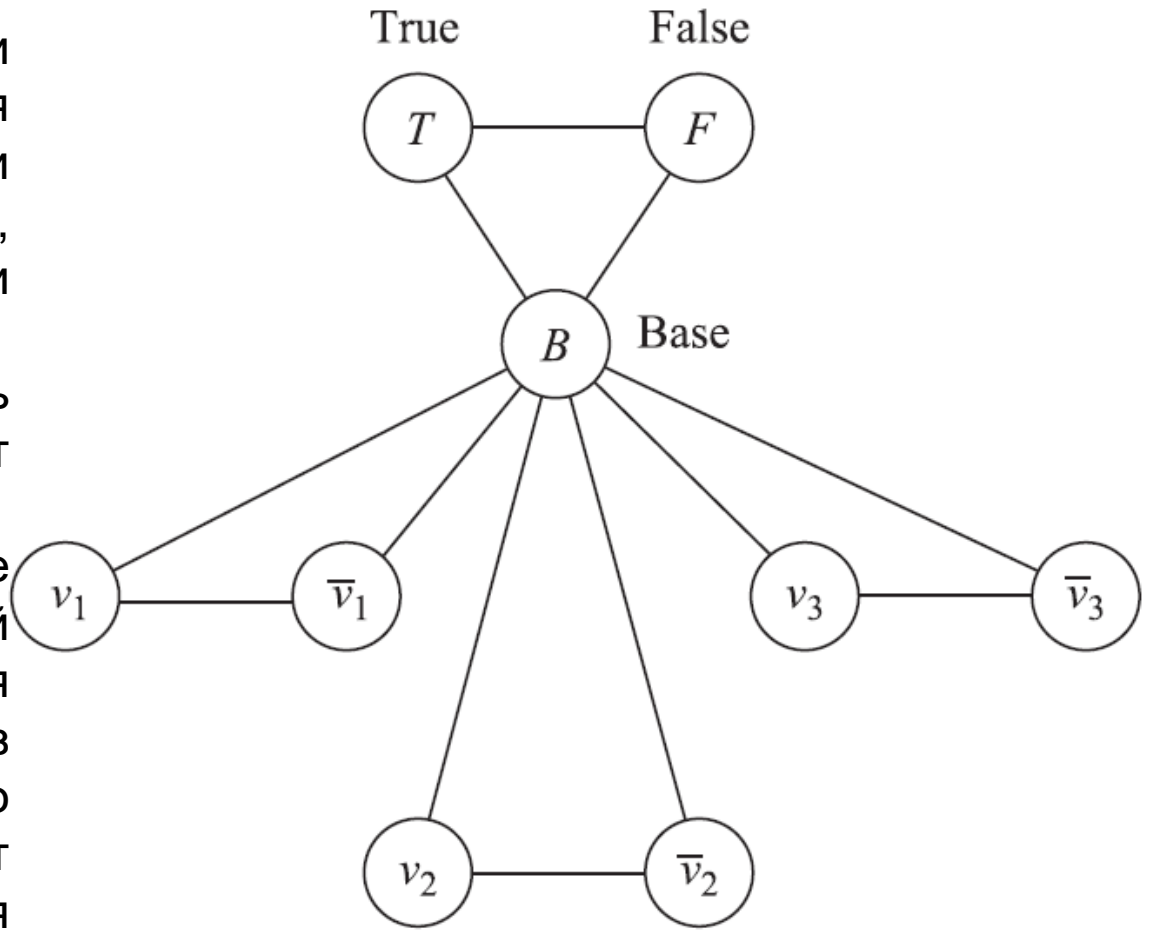
Начало сведения выглядит вполне ожидаемо. Возможно, основное достоинство 3-раскраски при кодировании булевых выражений заключается в том факте, что мы можем связать узлы графа с конкретными литералами, а соединяя узлы ребрами, можно гарантировать, что им будут назначены разные цвета; это обстоятельство позволяет связать с одним узлом истинное, а с другим — ложное значение. С учетом сказанного мы определяем узлы и $\bar{v}_i$, соответствующие каждой переменной и ее отрицанию $\bar{x}_i$. Также определяются три «специальных узла» T, F и B (сокращения от True, False и Base).

Доказательство NP-полноты задачи о 3-раскраске

Для начала мы соединим каждую пару узлов v_i, $\bar{v}_i$ ребром и соединим оба этих узла с Base (в результате чего образуется треугольник из v_i, $\bar{v}_i$ и Base для каждого i). Также True, False и Base соединяются в треугольник. Простой граф G, определенный к настоящему моменту, изображен на рис., и он уже обладает рядом полезных свойств.

В любой 3-раскраске графа G узлам v_i и $\bar{v}_i$ должны быть назначены разные цвета, и оба они должны отличаться от цвета Base.

В любой 3-раскраске графа G узлам True, False и Base должны быть назначены все три цвета в некоторой перестановке. В дальнейшем эти три цвета будут называться цветами True, False и Base в зависимости от того, какому из узлов соответствует тот или иной цвет. В частности, это означает, что для всех i одно из значений v_i или $\bar{v}_i$ получает цвет True, а другому достается цвет False. В оставшейся части этого построения будем считать, что переменной x_i значение 1 в заданном экземпляре 3-SAT присваивается в том, и только в том случае, если узлу v_i назначается цвет

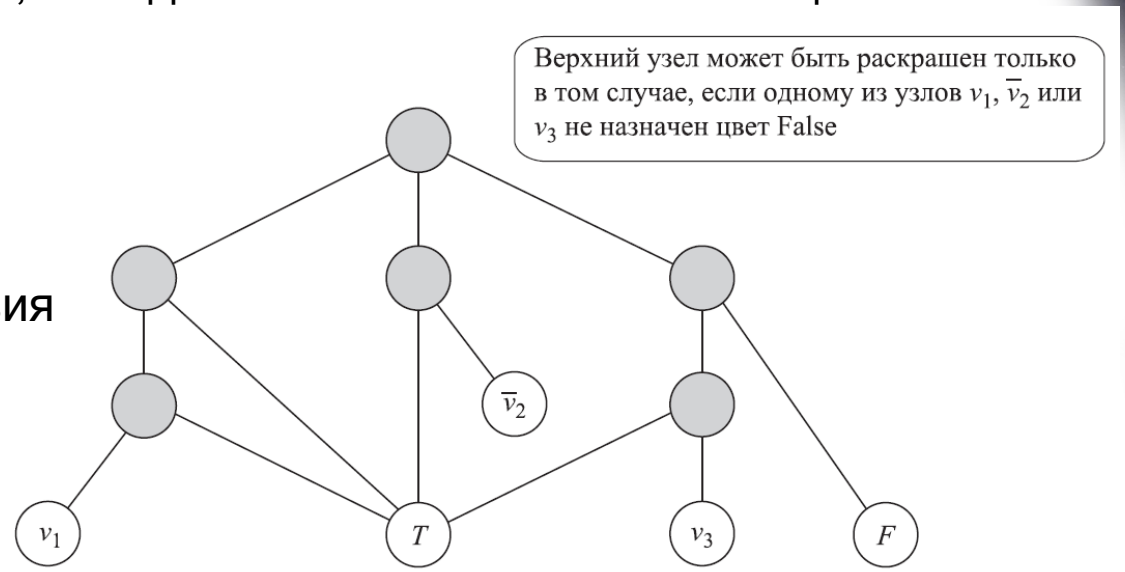


Доказательство NP-полноты задачи о 3-раскраске

Короче говоря, мы получили граф G, в котором любая 3-раскраска неявно определяет логическое присваивание для переменных в экземпляре 3-SAT. Теперь необходимо нарастить G так, чтобы только выполняющие присваивания могли быть расширены до 3-раскрасок полного графа. Как это сделать?

Как и в других сведениях 3-SAT, рассмотрим условие вида $x_1 \vee \bar{x}_2 \vee x_3$. На языке 3-раскраски графа G это означает: «По крайней мере одному из узлов v1, $\bar{v}_2$или v3 должен быть назначен цвет True». Итак, нам нужен маленький подграф, который можно присоединить к G, чтобы любая 3-раскраска, расширяющаяся на этот подграф, обладала свойством назначения цвета True по крайней мере одному из узлов v1, $\bar{v}_2$ или v3.

Присоединение подграфа для представления условия $x_1 \vee \bar{x}_2 \vee x_3$



Доказательство NP-полноты задачи о 3-раскраске

Наконец, ручная проверка показывает, что, если одному из узлов v_1, $\bar{v}_2$ или v_3 назначен цвет True, весь подграф может иметь 3-раскраску.

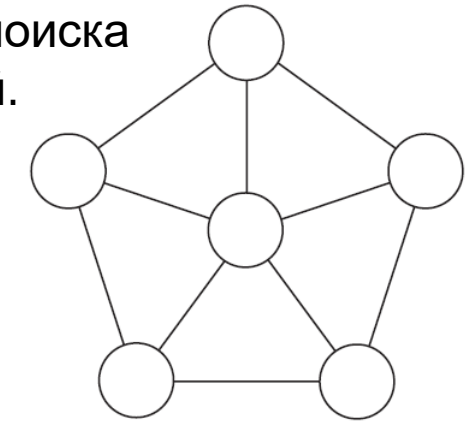
Далее остается лишь завершить построение: мы начинаем с графа G, определенного выше, и для каждого условия в экземпляре 3-SAT присоединяем подграф из шести узлов, изображенный на рис. Назовем полученный граф G'.

Утверждается, что заданный экземпляр 3-SAT выполним в том, и только в том случае, если граф G' имеет 3-раскраску. Сначала предположим, что для экземпляра 3-SAT существует выполняющее присваиванием. Определим раскраску G', окрасив Base, True и False в три разных цвета, а затем для каждого i назначим v_i цвет True, если $x_i = 1$, или цвет False, если $x_i = 0$. После этого $\bar{v}_i$ назначается единственный свободный цвет. Наконец, как объяснялось выше, теперь эта 3-раскраска может быть расширена на каждый подграф условия, состоящий из шести узлов, что приведет к 3-раскраске всех G'.

И наоборот, предположим, что у G' существует 3-раскраска. В этой раскраске каждому узлу v_i назначается либо цвет True, либо цвет False; переменная x_i задается соответствующим образом. Теперь утверждается, что в каждом условии экземпляра 3-SAT по крайней мере один из литералов в условии имеет значение истинности 1. В противном случае всем трем соответствующим узлам в 3-раскраске G' был бы назначен цвет False, но, как было показано выше, в такой ситуации не существует 3-раскраски соответствующего подграфа условия, — возникает противоречие. ■

Заключение: о проверке гипотезы четырех цветов

В завершение этого раздела стоит рассказать и о том, как закончилась история гипотезы четырех цветов для карт на плоскости. Более чем через 100 лет гипотеза была доказана Аппелем и Хакеном в 1976 году. Структура доказательства была простой индукцией по количеству областей, но шаг индукции включал около двух тысяч относительно сложных случаев, проверку которых пришлось поручить компьютеру. Многие математики были недовольны таким результатом: они надеялись получить доказательство, которое бы давало представление о том, почему гипотеза была истинной, — а вместо этого получили перебор запредельной сложности, проверку которого нельзя было осуществить вручную. Задача поиска относительно короткого, доступного доказательства все еще остается открытой.



Численные задачи

А теперь рассмотрим некоторые вычислительно сложные задачи, основанные на арифметических операциях с числами. Как вы увидите, вычислительная неразрешимость в данном случае обусловлена способом кодирования некоторых из представленных ранее задач в представлении очень больших целых чисел.

Задача о суммировании подмножеств

Базовой задачей в этой категории будет задача о суммировании подмножеств — частный случай задачи о рюкзаке. Версия этой задачи с принятием решения может быть сформулирована следующим образом:

Задано множество натуральных чисел w_1, ..., w_n и целевое число W. Существует ли подмножество $\{w_1, ..., w_n\}$, сумма которого равна ровно W?

В более общем виде эту проблему можно сформулировать, так как целые числа обычно будут задаваться в двоичном представлении, величина W в действительности экспоненциально зависит от размера входных данных; наш алгоритм не был алгоритмом с полиномиальным временем.

Эта проблема встречается во многих ситуациях; например, мы встречали ее в контексте алгоритмов сетевого потока, в которых пропускные способности были целочисленными. Возможно, другие примеры тоже покажутся вам знакомыми — например, безопасность такой криптографической системы, как RSA, обусловлена сложностью факторизации числа из 1000 битов. Но если бы время выполнения из 2^{1000} шагов считалось приемлемым, факторизация такого числа не создавала бы никаких трудностей.

Итак, основной вопрос: возможно ли решение задачи о суммировании подмножеств алгоритмом с (действительно) полиномиальным временем? Другими словами, существует ли алгоритм с временем выполнения, полиномиальным по n и log W? Или полиномиальным только по n?

Доказательство NP-полноты задачи о суммировании подмножеств

Следующий результат наводит на мысль, что существование такого алгоритма маловероятно.

Утверждение 1.23. Задача о суммировании подмножеств является NP-полной. Доказательство. Сначала покажем, что задача о суммировании подмножеств принадлежит NP. Для заданных натуральных чисел w1, ..., wn и целевого значения W сертификатом, подтверждающим существование решения, будет подмножество $w_{i_1}, \dots w_{i_n}$, которое в сумме дает W. За полиномиальное время мы можем вычислить сумму этих чисел и убедиться в том, что она равна W.

Теперь попробуем свести заведомо NP-полную задачу к задаче о суммировании подмножеств. Так как мы ищем множество, которое в сумме дает заданную величину (а не ограничивается ею сверху или снизу), речь идет о комбинаторной задаче, основанной на получении точного значения. Задача о трехмерном сочетании является естественным кандидатом; мы покажем, что ***Трехмерное сочетание*** $\leq_P$ ***Сумма подмножеств***. Основная хитрость будет связана со способом кодирования операций с множествами посредством суммирования целых чисел.

Доказательство NP-полноты задачи о суммировании подмножеств

Рассмотрим экземпляр задачи о трехмерном сочетании, определяемый множествами X, Y, Z, каждое из которых имеет размер n, и множеством m триплетов $T \subseteq X \times Y \times Z$. Стандартный способ представления множеств основан на использовании битовых векторов: каждый элемент векторов соответствует разным элементам множества, и равен 1 в том, и только в том случае, если множество содержит этот элемент. Мы воспользуемся этим методом для представления триплетов $t = (x_i, y_j, z_k) \in T$: мы строим число w_t с 3n цифрами, которое содержит 1 в позициях i, n + j и 2n + k и 0 во всех остальных позициях. Другими словами, для некоторого основания $d > 1$ $w_t = d^{i-1} + d^{n+j-1} + d^{2n+k-1}$.

Обратите внимание на то, что вычисление объединения триплетов почти соответствует целочисленному сложению: единицы заполняют те позиции, в которых любое из множеств содержит элементы. Мы говорим «почти», потому что при сложении используются переносы: лишние единицы в одном столбце «переносятся» и образуют ненулевой элемент в следующем столбце. В контексте операции объединения у переноса аналогов нет.

В текущей ситуации эта проблема решается простым приемом. Всего используется m чисел, каждое состоит из цифр 0 или 1; если предположить, что наши числа записаны по основанию d = m + 1, то переносов не будет вообще.

Доказательство NP-полноты задачи о суммировании подмножеств

Построим следующий экземпляр задачи о суммировании подмножеств. Для каждого триплета $t = (x_i, y_j, z_k) \in T$ строится число w_t в записи по основанию m + 1, как определяется выше. Затем W определяется как число в записи по основанию m + 1 с 3n цифрами, каждая из которых равна 1, то есть $\sum_{i=0}^{3n-1}(m+1)^i$.

Утверждается, что множество T триплетов содержит идеальное трехмерное сочетание в том, и только в том случае, если существует подмножество чисел $\{w_t\}$, которое дает в сумме W. Предположим, существует идеальное трехмерное сочетание триплетов t_1, ..., t_n. В этом случае в сумме $w_{t_1}, \dots w_{t_n}$ в каждой из 3n позиций находится одна единица, поэтому результат равен W. И наоборот, предположим, существует множество чисел $w_{t_1}, \dots w_{t_n}$, которое в сумме дает W. Тогда, поскольку содержатся три единицы и переносы отсутствуют, в представлении каждого w_{t_i} мы знаем, что k = n. Следовательно, для каждой из 3n позиций цифр ровно одно из w_{t_i} содержит 1 в этой позиции. Следовательно, t_1, ..., t_k образуют идеальное трехмерное сочетание. ■

Расширения: сложность некоторых задач планирования

Сложность задачи о суммировании подмножеств может использоваться для установления сложности целого класса задач планирования, включая некоторые задачи, не связанные с суммированием чисел. Ниже приведен хороший пример: естественное (но намного более сложное) обобщение задачи планирования.

Допустим, дано множество из n задач, которые должны выполняться на одном компьютере. Для каждого задания i установлено время доступности r_i, когда оно впервые становится доступным для обработки; предельное время d_i, к которому оно должно быть завершено; и продолжительность обработки t_i. Будем считать, что все эти параметры являются целыми числами. Чтобы задание i было завершено, ему должен быть выделен смежный набор из t_i единиц времени где-то в интервале $[r_i, d_i]$. В любой момент на машине может выполняться только одно задание. Вопрос заключается в следующем: можно ли спланировать все задания для выполнения так, чтобы каждое задание было завершено к предельному времени? Назовем этот экземпляр задачей планирования с временем доступности и предельным временем.

Утверждение 1.24. Задача планирования с временем доступности и предельным временем является NP-полной.

Расширения: сложность некоторых задач планирования

Доказательство. Для заданного экземпляра задачи сертификатом, доказывающим наличие решения, будет список начального времени выполнения для каждого задания. При наличии такого списка можно убедиться в том, что каждое задание выполняется в четко определенный интервал времени между временем доступности и предельным временем. Следовательно, задача принадлежит NP.

Теперь покажем, что задача о суммировании подмножеств сводится к этой задаче планирования. Рассмотрим экземпляр задачи о суммировании подмножеств с числами w_1, ..., w_n и целевым значением W. При построении эквивалентного экземпляра задачи планирования сначала бросается в глаза обилие параметров, которыми необходимо управлять: время доступности, предельное время и продолжительность. Здесь важно пожертвовать большей частью этой гибкости и получить «облегченный» экземпляр задачи, который, тем не менее, кодирует задачу о суммировании подмножеств.

Пусть $S = \sum_{i=1}^{n} w_i$. Мы определяем задания 1, 2, ..., n; задание i имеет время доступности 0, предельное время S+1 и продолжительность w_i. Задания из этого множества можно расположить в любом порядке, и все они завершатся вовремя. Теперь мы установим дополнительные ограничения для экземпляра, чтобы решить его можно было только группировкой подмножества заданий, продолжительности которых в сумме дают ровно W.

Расширения: сложность некоторых задач планирования

Для этого мы определяем (n + 1)-е задание с временем доступности W, предельным временем W + 1 и продолжительностью 1. Рассмотрим любое действительное решение этого экземпляра задачи планирования. (N + 1)-е задание должно выполняться в интервале [W, W + 1]. Между общим временем доступности и общим предельным временем свободны S единиц времени; и суммарная продолжительность заданий также равна S. Таким образом, на машине не должно быть простоев, когда не выполняется ни одно задание. В частности, если до времени W выполняются задания i_1, ..., i_k, то соответствующие числа $w_{i_1}, \dots w_{i_k}$ в экземпляре задачи о суммировании подмножеств в сумме дают ровно W.

И наоборот, если существуют числа $w_{i_1}, \dots w_{i_n}$, которые в сумме дают ровно W, их можно распланировать до задания n + 1, а остальные — после задания n + 1; и это расписание будет действительным решением для экземпляра задачи планирования. ■

Внимание: суммирование подмножеств с полиномиально ограничиваемыми числами

С задачей о суммировании подмножеств связан очень распространенный источник ошибок. И хотя он тесно связан с проблемами, о которых уже говорилось ранее, мы считаем, что эта ловушка заслуживает явного упоминания.

Рассмотрим особый случай задачи о суммировании подмножеств с n входными числами, в котором W ограничивается полиномиальной функцией n. В предположении, что P ≠ NP, этот частный случай не является NP-полным.

Он не является NP-полным по той простой причине, что не может быть решен за время O(nW) нашим алгоритмом динамического программирования; когда W ограничивается полиномиальной функцией n, это алгоритм с полиномиальным временем.

Все это вполне понятно; так зачем останавливаться на этом? Дело в том, что существует целая категория задач, которой часто приписывают NP-полноту (даже в опубликованных статьях) посредством сведения от частного случая суммы подмножеств. Ниже приведен типичный пример такой задачи, который мы назовем задачей группировки компонентов.

Для заданного графа G, не являющегося связным, и числа k существует ли подмножество связных компонентов, размер объединения которых равен в точности k? Неправильное утверждение. Задача о группировке компонентов является NP-полной.

Внимание: суммирование подмножеств с полиномиально ограничиваемыми числами

Неправильное доказательство. Задача о группировке компонентов принадлежит NP, доказательство не приводится. Теперь мы попытаемся показать, что Суммирование подмножеств $\leq_P$ Группировка компонентов. Для заданного экземпляра задачи о суммировании подмножеств с числами w_1, ..., w_n и целевого значения W экземпляр задачи группировки компонентов строится следующим образом: для каждого i строится путь P_i длины w_i. Граф G представляет собой объединение путей P_1, ..., P_n, каждый из которых является отдельным связным компонентом. Мы присваиваем k = W. Очевидно, что G содержит множество связных компонентов, объединение которых имеет размер k в том, и только в том случае, если некоторое подмножество чисел w_1, ..., w_n в сумме дает W. ■

Ошибка в этом «доказательстве» весьма коварна; в частности, утверждение в последнем предложении истинно. Проблема в том, что построение, описанное выше, не устанавливает, что Суммирование подмножеств $\leq_P$ Группировка компонентов, потому что эта задача требует более чем полиномиального времени.

Внимание: суммирование подмножеств с полиномиально ограничиваемыми числами

При построении входных данных для «черного ящика», решающего задачу группировки компонентов, нам пришлось построить систему кодирования графа с размером $w_1 + ... + w_n$, а это требует времени, экспоненциального по размеру входных данных экземпляра задачи о суммировании подмножеств. По сути, задача о суммировании подмножеств работает с числами w_1, ..., w_n в чрезвычайно компактном представлении, но задача группировки компонентов не получает «компактную» кодировку графа.

Проблема более фундаментальна, чем неправильность этого доказательства; на самом деле задача группировки компонентов может быть решена за полиномиальное время. Если n_1, n_2, ..., n_c — размеры связных компонентов G, мы просто используем наш алгоритм динамического программирования для задачи о суммировании подмножеств, чтобы принять решение о том, дает ли некоторое подмножество этих чисел $\{n_i\}$ в сумме k. Необходимое для этого время выполнения равно O(ck); и поскольку и c, и k ограничиваются n, получается время $O(n^2)$.

Таким образом, мы обнаружили новый алгоритм с полиномиальным временем посредством сведения в обратном направлении — к особому случаю задачи о суммировании подмножеств, решаемому за полиномиальное время.

Co-NP и асимметрия NP

Чтобы расширить свое представление об этом общем классе задач, вернемся к тем определениям, на которых базировался класс NP. Мы уже видели, что понятие эффективного сертифицирующего алгоритма не предполагает конкретного алгоритма для фактического решения задачи, который бы превосходил метод «грубой силы».

А теперь еще одно наблюдение: определение эффективного сертифицирования (а следовательно, и NP) по своей сути асимметрично. Входная строка s представляет экземпляр с положительной сертификацией в том, и только в том случае, если существует короткое значение t, для которого B(s, t) = да. Из отрицания этого утверждения мы видим, что входная строка представляет экземпляр с отрицательной сертификацией в том, и только в том случае, если для всех коротких t выполняется B(s, t) = нет.

Все это близко к нашим интуитивным представлениям о NP: если экземпляр сертифицируется положительно, мы можем привести короткое доказательство этого факта. Но для экземпляров с отрицательной сертификацией определение не гарантирует соответствующего короткого доказательства; ответ отрицателен просто потому, что не удается найти строку, которая бы служила доказательством. Если набор условий невыполним, какие «доказательства» могли бы быстро убедить вас в невыполнимости задачи?

Co-NP и асимметрия NP

Для каждой задачи X существует естественная дополняющая задача $\bar{X}$: для всех входных строк s мы говорим, что $s \in X$ в том, и только в том случае, если $s \notin \bar{X}$. Следует заметить, что если $X \in P$, то $\bar{X} \in P$, так как из алгоритма A, решающего X, мы можем просто построить алгоритм $\bar{A}$, который выполняет A, а затем «инвертирует» его ответ.

Но при этом совершенно не очевидно, что из $X \in NP$ должно следовать, что $\bar{X} \in NP$. Вместо этого задача $\bar{X}$ обладает другим свойством: для всех s выполняется $s \in \bar{X}$ в том, и только в том случае, если для всех t длины не более $p(|s|)$, B(s, t) = нет. Это принципиально иное определение, которое невозможно обойти простым «инвертированием» вывода эффективного сертифицирующего алгоритма B для получения $\bar{B}$. Проблема в том, что «существует t» в определении NP превращается в «для всех t», и это серьезное изменение.

Существует класс задач, параллельных NP, спроектированных для моделирования этой проблемы; ему присвоено достаточно естественное название co-NP. Задача X принадлежит co-NP в том, и только в том случае, если дополняющая задача $\bar{X}$ принадлежит NP.

Co-NP и асимметрия NP

Но мы не знаем наверняка, что NP и co-NP различны; можем только спросить:

Утверждение 1.25. Выполняется ли NP = co-NP? И снова принято считать, что NP ≠ co-NP: из того, что экземпляры задач с положительной сертификацией имеют короткие доказательства, совершенно не следует, что экземпляры с отрицательной сертификацией также должны иметь короткие доказательства.

Доказательство NP ≠ co-NP стало бы еще более важным шагом, чем доказательство P ≠ NP, по следующей причине:

Утверждение 1.26. Если NP ≠ co-NP, то P ≠ NP.

Доказательство. На самом деле мы докажем обратное утверждение: из P = NP следует NP = co-NP. Здесь важно то, что множество P замкнуто относительно дополнения; следовательно, если P = NP, то множество NP также будет замкнуто относительно дополнения. Или в более формальном изложении, начиная с предположения P = NP, имеем

$$X \in NP \Longrightarrow X \in P \Longrightarrow \bar{X} \in P \Longrightarrow \bar{X} \in NP \Longrightarrow X \in co - NP$$

и

$$X \in co - NP \Longrightarrow \bar{X} \in NP \Longrightarrow \bar{X} \in P \Longrightarrow X \in P \Longrightarrow X \in NP$$

Из этого следует, что $NP \subseteq co - NP$ и $co - NP \subseteq NP$, а значит, NP = co-NP. ■

Хорошая характеризация: класс NP ∩ co-NP

Если задача X принадлежит как NP, так и co-NP, то она обладает следующим полезным свойством: для ответа «да» существует короткое доказательство; для ответа «нет» также существует короткое доказательство. Задачи, принадлежащие этому пересечению NP ∩ co-NP, называются имеющими хорошую характеризацию, так как для решения всегда существует хороший сертификат.

Это понятие напрямую соответствует некоторым результатам, которые мы видели ранее. Например, возьмем задачу определения того, содержит ли потоковая сеть поток с величиной как минимум v, для некоторого значения v. Чтобы доказать, что ответ на этот вопрос положительный, достаточно продемонстрировать поток, достигающий этой величины; это свойство согласуется с принадлежностью задачи NP. Но также можно доказать и отрицательный ответ: можно продемонстрировать разрез, пропускная способность которого строго меньше v. Этот дуализм между экземплярами с положительной и отрицательной сертификацией является сутью теоремы о максимальном потоке и минимальном разрезе.

Если задача X находится в P, то она принадлежит как NP, так и co-NP; следовательно, P ⊆ NP ∩ co-NP.

этот вопрос также остается открытым:

Утверждение 1.27. Выполняется ли P = NP ∩ co-NP?