МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт искусственного интеллекта
Базовая кафедра 252

КУРСОВАЯ РАБОТА

по дисциплине
«Математическая логика и теория алгоритмов»

Тема курсовой работы
«NP-полные задачи»

Студент: Никишина Анна Александровна
Группа: ККСО–03–19

Руководитель
курсовой работы: Кузнецов Владимир Сергеевич

Москва
2022

Содержание

1. Введение

Далеко не все задачи можно решить за разумное время. Возможно, в будущем нам предстоит столкнулся именно с одной из таких задач. Задачи из класса NP постоянно встречаются в различных сферах жизнедеятельности: восстановление поврежденных файлов, оптимизация маршрутов, сложные вычисления в биоинформатике. Криптография открытых ключей основывается на предположении, что $NP \neq P$. Если найдется способ решать задачи этого класса за полиномиальное время, то многие методы защиты больше не будут иметь смысла. В связи с этим, изучение данного класса задач, представляется крайне важным и актуальным в будущей специальности.

1.1. Цели и зачи

Целью моей курсовой работы является изучение понятия NP-полных задач.
Для достижения поставленной цели, я выделила задачи, поэтапно выполняя которые, будет проходить исследование данной темы.

Задачи:

1. Изучить понятие NP-полных задач.
2. Рассмотреть примеры NP-полных задач.
3. Изучить классы P и NP, а также понятие NP-полноты.
4. Определить понятие NP-полноты и как её доказывать.
5. В качестве практической части доказать NP-полноту на примере конкретной задачи.
6. Сделать вывод о проделанной работе.

1.2. Актуальность

Чтобы стать квалифицированным разработчиком алгоритмов, необходимо понимать основы теории NP-полноты. Если установлено, что задача NP-полная, это служит достаточно надежным указанием на то, что она трудноразрешимая. Как инженер, вы можете эффективнее потратить время, если займетесь разработкой приближенного алгоритма или решением легкого особого случая, вместо того, чтобы искать быстрый алгоритм, выдающий точное решение задачи. Более того, многие естественно возникающие интересные задачи, которые на первый взгляд не сложнее задачи сортировки, поиска в графе или определения потока в сети, фактически являются NP-полными. Таким образом, важно ознакомиться с этим замечательным классом задач.

2. Теоретическая часть

2.1. Общее теоретическое введение

Многие известные нам задачи, например алгоритм быстрой сортировки, все базовые арифметические операции, максимальное паросочетание в графах и другие из изученных имеют полиномиальное время работы: для входных данных размера n их время работы в наихудшем случае равно $O(n^k)$, где k — некоторая константа. Возникает естественный вопрос: все ли задачи можно решить в течение полиномиального времени? Ответ отрицательный. В качестве примера можно привести знаменитую «задачу останова», предложенную Тьюрингом. Эту задачу невозможно решить ни на одном компьютере, каким бы количеством времени мы не располагали. Существуют также задачи, которые можно решить, но не удается сделать это за время $O(n^k)$, где k — некоторая константа. Например, процедура вычисления для арифметики Пресбургера, вычисление базиса Грёбнера (в худшем случае) – эти задачи выполняются за дважды экспоненциальное время ($T(n)$ ограничено сверху значением $2^{2^{poly(n)}}$, где $poly(n)$ – некоторый многочлен от n).

NP-полные задачи – это интересный класс задач. Их статус пока что неизвестен. Для решения NP-полных задач до настоящего времени не разработано алгоритмов с полиномиальным временем работы, но и не доказано, что для какой-то из них таких алгоритмов не существует. Этот так называемый вопрос $P \neq NP$ с момента своей постановки в 1971 году стал одним из самых трудных в теории вычислительных систем.

2.2. Примеры NP-полных задач

Особо интригующим аспектом NP-полных задач является то, что некоторые из них на первый взгляд аналогичны задачам, для решения которых существуют алгоритмы с полиномиальным временем работы. В каждой из описанных ниже пар задач одна из них разрешима в течение полиномиального времени, а другая является NP-полной. При этом различие между задачами кажется совершенно незначительным.

Поиск самых коротких и самых длинных простых путей. Даже при отрицательных весах ребер кратчайшие пути от отдельно взятого источника в ориентированном графе $G = (V, E)$ можно найти в течение времени $O(VE)$. Однако поиск самого длинного пути между двумя вершинами оказывается сложным. Задача по определению того, содержит ли граф простой путь, количество ребер в котором не меньше заданного числа, является NP-полной.

Эйлеров и Гамильтонов циклы. Эйлеров цикл связанного ориентированного графа $G = (V, E)$ — это цикл, в котором переход по каждому ребру G осуществляется ровно один раз, хотя допускается неоднократное посещение некоторых вершин. Определить наличие Эйлерова цикла (а также найти составляющие его ребра) можно в течение времени $O(E)$. Гамильтонов цикл ориентированного графа $G = (V, E)$ — это простой цикл, содержащий все вершины из множества V . Задача по определению того, содержится ли в ориентированном графе Гамильтонов цикл, является NP-полной.

2-CNF- и 3-CNF-выполнимость. Булева формула содержит переменные, принимающие значения 0 и 1, булевы операторы, такие как $\wedge$ (И), $\vee$ (ИЛИ) и $\neg$ (НЕ), а также скобки. Булева формула называется выполнимой, если входящим в ее состав переменным можно присвоить такие значения 0 и 1, чтобы в результате вычисления формулы получилось значение 1. Булева формула

представлена в k-конъюнктивной нормальной форме, или k-CNF, если она имеет вид конъюнкции (И) взятых в скобки выражений, являющихся дизъюнкцией (ИЛИ) ровно k переменных или их отрицаний (НЕ). Например, формула $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$ представлена в 2-CNF. (Для нее существует выполнимый набор $x_1 = 1, x_2 = 0, x_3 = 1$.) Можно сформулировать алгоритм с полиномиальным временем работы, позволяющий определить, является ли 2-CNF формула выполнимой. Однако, определение того, является ли 3-CNF формула выполнимой — это NP-полная задача.

2.3. NP-полнота и классы P и NP

Будем ссылаться на три класса задач: P, NP и NPC (класс NP- полных задач (NP-complete)).

Класс P состоит из задач, разрешимых в течение полиномиального времени работы. Точнее говоря — это задачи, которые можно решить за время $O(n^k)$, где k — некоторая константа, а n — размер входных данных задачи.

Определение 2.3.1 (Определение класса сложности P).
$P = \{L \subseteq \{0,1\} :$ *существует алгоритм A, разрешающий язык L за полиномиальное время*$\}$.

Класс NP состоит из задач, которые поддаются проверке в течение полиномиального времени. Имеется в виду, что если мы каким-то образом получаем «сертификат» решения, то в течение времени, полиномиальным образом зависящего от размера входных данных задачи, можно проверить корректность такого решения. Например, в задаче о гамильтоновом цикле с заданным ориентированным графом $G = (V, E)$ сертификат имел бы вид последовательности $(v_1, v_2, \ldots, v_{|V|})$ из $|V|$ вершин. В течение полиномиального времени легко проверить, что $(v_i, v_{i+1} \in E$ при $i = 1, 2, \ldots, |V| - 1$ и что $(v_{|V|}, v_1) \in E$. Приведем другой пример: в задаче о 3-CNF-выполнимости в сертификате должно быть указано, какие значения следует присвоить переменным. В течение полиномиального времени легко проверить, удовлетворяет ли такое присваивание булевой формуле.

Определение 2.3.2 (Определение класса сложности NP).
Класс сложности NP *– это класс языков, которые можно верифицировать с помощью алгоритма с полиномиальным временем работы. Точнее говоря, язык принадлежит классу NP тогда и только тогда, когда существует алгоритм A с двумя входными параметрами и полиномиальным временем работы, а также константа c, такая что*
$L = \{x \in \{0,1\} :$ *существует сертификат* y $(|y| = O(|x|^c))$ *такой, что* $A(x, y) = 1\}$.
При этом говорят, что алгоритм A верифицирует язык L за полиномиальное время.

Неформально задача принадлежит классу NPC, если она принадлежит классу NP и является такой же «сложной», как и любая задача из класса NP. Если любую NP-полную задачу можно решить в течение полиномиального времени, то для каждой задачи из класса NP существует алгоритм с полиномиальным временем работы. Большинство ученых, занимающихся теорией вычислительных систем, считают NP-полные задачи очень трудноразрешимыми, потому что при огромном разнообразии изучавшихся до настоящего времени NP-полных задач ни для одной из них пока так и не найдено решение в виде алгоритма с полиномиальным временем работы. Таким образом, было бы крайне удивительно, если бы все они оказались разрешимыми в течение полиномиального времени.

2.4. Как показать, что задача является NP-полной

Методы, позволяющие доказать, что та или иная задача является NP-полной, отличаются от методов, которые использовались в большей части этой книги для разработки и

анализа алгоритмов. Имеется фундаментальная причина такого различия: чтобы показать, что задача является NP-полной, делается утверждение о том, насколько она сложна (или, по крайней мере, насколько она сложна в нашем представлении), а не о том, насколько она проста. Не предпринимается попыток доказать, что существуют эффективные алгоритмы. Скорее, мы пытаемся доказать, что эффективных алгоритмов, вероятнее всего, не существует. В этом смысле доказательство NP-полноты несколько напоминает доказательство того, что нижняя граница любого алгоритма, работающего по методу сравнений, равна $\Omega(n \lg(n))$. Однако методы, использующиеся для доказательства NP-полноты, отличаются от методов с применением дерева решений. При доказательстве NP-полноты задачи используются три основные концепции, описанные ниже.

2.4.1 Задачи принятия решения и задачи оптимизации

Многие представляющие интерес задачи являются задачами оптимизации, каждому допустимому ("законному") решению которых можно сопоставить некоторое значение и для которых нужно найти допустимое решение с наилучшим значением. Например, в задаче, получившей название Shortest_Path, задается неориентированный граф G, а также вершины u и v, и нужно найти путь из вершины u к вершине v, в котором содержится наименьшее количество ребер. (Другими словами, Shortest_Path — это задача поиска кратчайшего пути между парой вершин невзвешенного неориентированного графа.) Однако NP-полнота непосредственно применима не к задачам оптимизации, а к задачам принятия решения, в которых ответ может быть положительным или отрицательным (говоря более формально, принимать значения "1" или "0").

Хотя при доказательстве NP-полноты задачи приходится ограничиваться задачами принятия решения, между ними и задачами оптимизации существует удобная взаимосвязь. Наложив ограничение на оптимизируемое значение, поставленную задачу оптимизации можно свести к соответствующей задаче принятия решения. Например, задаче Shortest_Path соответствует задача принятия решения под названием Path: существует ли для заданных исходных данных, в число которых входит направленный граф G, вершины u и v, и целое число k, путь из вершины u к вершине v, состоящий не более чем из k ребер.

Взаимосвязь между задачей оптимизации и соответствующей ей задачей принятия решения полезна для нас, если мы пытаемся показать, что задача оптимизации является «сложной». Причина этого заключается в том, что задача принятия решения в некотором смысле «проще» или, по крайней мере, «не сложнее». Например, задачу Path можно решить, решив задачу Shortest_Path, а затем сравнивая количество ребер в найденном кратчайшем пути со значением параметра k в задаче принятия решения. Другими словами, если задача оптимизации простая, соответствующая ей задача принятия решения тоже простая. Формулируя это утверждение так, чтобы оно имело большее отношение к NP-полноте, можно сказать, что если удается засвидетельствовать сложность задачи принятия решения, это означает, что соответствующая задача оптимизации тоже сложная. Таким образом, хотя и приходится ограничиваться рассмотрением задач принятия решения, теория NP-полноты зачастую имеет следствия и для задач оптимизации.

2.4.2 Приведение

Сделанное выше замечание о том, что одна задача не сложнее или не легче другой, применимо, даже если обе задачи являются задачами принятия решения. Эта идея используется почти во всех доказательствах NP-полноты. Это делается следующим образом. Рассмотрим задачу принятия решения, скажем, задачу A, которую хотелось бы решить в течение полиномиального времени. Назовем входные данные отдельно взятой задачи экземпляром этой

задачи. Например, экземпляром задачи Путь является некоторый граф G, некоторые его вершины u и v, а также некоторое целое число k. А теперь предположим, что существует другая задача принятия решения, скажем, B, для которой заранее известно, как решить ее в течение полиномиального времени. Наконец, предположим, что имеется процедура с приведенными ниже характеристиками, преобразующая любой экземпляр α задачи A в некоторый экземпляр β задачи B.

1. Это преобразование занимает полиномиальное время.
2. Ответы являются идентичными, т.е. в экземпляре α ответ «да» выдается тогда и только тогда, когда в экземпляре β тоже выдается ответ «да».

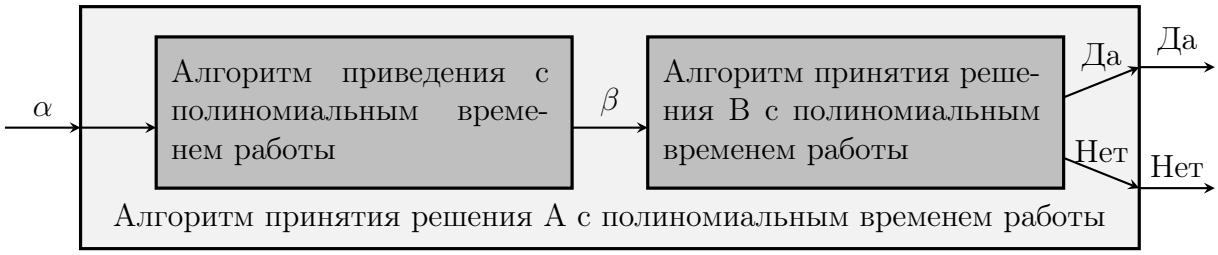

Рис. 2.4.1: Решение задачи принятия решения A в течение полиномиального времени с помощью алгоритма приведения с полиномиальным временем работы и известного алгоритма с полиномиальным временем работы, предназначенного для решения другой задачи B

Назовем такую процедуру алгоритмом приведения с полиномиальным временем. Как видно из Рис. 2.4.1, эта процедура предоставляет описанный ниже способ решения задачи A в течение полиномиального времени.

1. Заданный экземпляр α задачи A с помощью алгоритма приведения с полиномиальным временем преобразуется в экземпляр β задачи B.
2. Запускается алгоритм, решающий экземпляр β задачи принятия решения B в течение полиномиального времени.
3. Ответ для экземпляра β используется в качестве ответа для экземпляра α.

Поскольку для выполнения каждого из перечисленных выше этапов требуется полиномиальное время, это относится и ко всему процессу в целом, что дает способ решения экземпляра α задачи в течение полиномиального времени. Другими словами, путем сведения задачи A к задаче B «простота» задачи B используется для доказательства «простоты» задачи A.

Если вспомнить, что при доказательстве NP-полноты требуется показать, насколько сложной является задача, а не насколько она простая, доказательство NP-полноты с помощью приведения с полиномиальным временем работы выполняется обратно описанному выше способу. Продвинемся в разработке этой идеи еще на шаг и посмотрим, как с помощью приведения с полиномиальным временем показать, что для конкретной задачи B не существует алгоритмов с полиномиальным временем работы. Предположим, что имеется задача принятия решения A, относительно которой заранее известно, что для нее не существует алгоритма с полиномиальным временем работы. Предположим также, что имеется преобразование, позволяющее в течение полиномиального времени преобразовать экземпляры задачи A в экземпляры задачи B. Теперь с помощью простого доказательства «от противного» можно показать, что для решения задачи B не существует алгоритмов с полиномиальным временем работы.

Доказательство. Предположим обратное, т.е. что существует решение задачи B в виде алгоритма с полиномиальным временем работы. Тогда, воспользовавшись методом, проиллюстрированным на Рис. 2.4.1, можно получить способ решения задачи A в течение полиномиального времени, а это противоречит предположению о том, что таких алгоритмов для задачи A не существует. □

Если речь идет о NP-полноте, то здесь нельзя предположить, что для задачи A вообще не существует алгоритмов с полиномиальным временем работы. Однако методология аналогична в том отношении, что доказательство NP-полноты задачи B основывается на предположении о NP-полноте задачи A.

2.4.3 Первая NP-полная задача

Поскольку метод приведения базируется на том, что для какой-то задачи заранее известна ее NP-полнота, то для доказательства NP-полноты различных задач нам понадобится «первая» NP-полная задача. В качестве таковой мы воспользуемся задачей, в которой задана булева комбинационная схема, состоящая из логических элементов И, ИЛИ и НЕ. В задаче спрашивается, существует ли для этой схемы такой набор входных булевых величин, для которого будет выдано значение 1.

2.5. NP-полнота и приводимость

По-видимому, одна из веских причин, по которым специалисты в области теории вычислительных систем полагают, что $P \neq NP$, — наличие класса «NP-полных» задач. Этот класс обладает замечательным свойством, которое состоит в том, что если хоть одну (любую) NP-полную задачу можно решить в течение полиномиального времени, то и все задачи этого класса обладают полиномиальновременным решением, т.е. $P = NP$. Однако, несмотря на многолетние исследования, до сих пор не обнаружено ни одного алгоритма с полиномиальным временем работы ни для одной NP-полной задачи.

В определенном смысле NP-полные языки — «самые сложные» в классе NP.

2.5.1 Приводимость

Интуитивно понятно, что задачу Q можно свести к другой задаче Q', если любой экземпляр задачи Q «легко перефразируется» в экземпляр задачи Q', решение которого позволяет получить решение соответствующего экземпляра задачи Q. Например, задача решения линейных уравнений относительно неизвестной величины x сводится к задаче решения квадратных уравнений. Если задан экземпляр $ax + b = 0$, его можно преобразовать в уравнение $0x^2 + ax + b = 0$, решение которого совпадает с решением уравнения $ax + b = 0$. Таким образом, если задача Q сводится к другой задаче Q', то решить задачу Q в некотором смысле «не сложнее», чем задачу Q'.

Определение 2.5.1 (Определение приводимого языка, функции приведения и алгоритма приведения)**.**
Говорят, что язык L_1 ***приводим в течение полиномиальноro времени*** *к языку L_1 (что обозначается как $L_1 \leqslant_P L_2$), если существует вычислимая за полиномиальное время функция $f : \{0,1\} \to \{0,1\}$, такая что для всех $x \in \{0,1\}$*

$$x \in L_1 \text{ тогда и только тогда, когда } f(x) \in L_2. \tag{2.5.1}$$

Функцию f называют ***функцией приведения****, а алгоритм F с полиномиальным временем работы, вычисляющий функцию f , –* ***алгоритмом приведения****.*

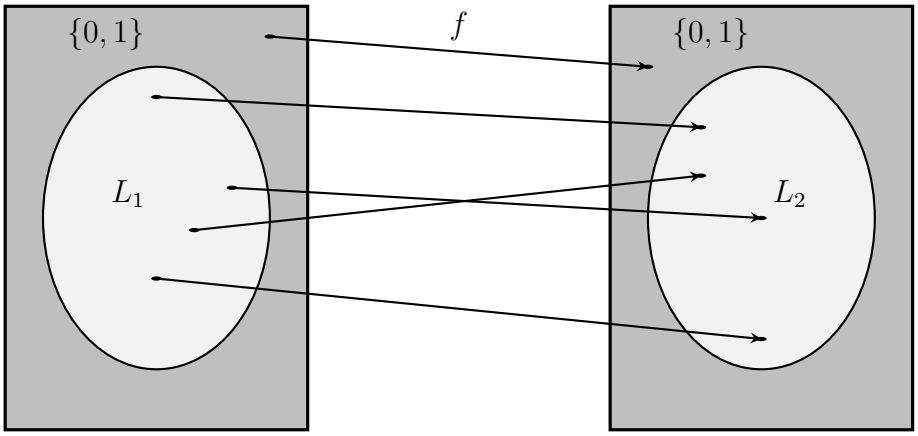

Рис. 2.5.2: Иллюстрация к приведению в течение полиномиального времени языка L_1 к языку L_2 с помощью функции приведения f

Рис. 2.5.2 иллюстрирует понятие приведения языка L_1 к другому языку L_2 в течение полиномиального времени. Каждый язык – это подмножество множества 0, 1. Функция приведения f обеспечивает такое отображение в течение полиномиального времени, что если $x \in L_1$, то $f(x) \in L_2$. Более того, если $x \notin L_1$, то $f(x) \notin L_2$. Таким образом, функция приведения отображает любой экземпляр x задачи принятия решения, представленной языком L_1, на экземпляр $f(x)$ задачи, представленной языком $L2$. Ответ на вопрос о том, принадлежит ли экземпляр $f(x)$ языку L_2, непосредственно позволяет ответить на вопрос о том, принадлежит ли экземпляр x языку L_1.

Лемма 2.5.3. *Если $L_1, L_2 \subseteq \{0,1\}$ — языки, такие что $L_1 \leqslant_P L_2$, то из $L_2 \in P$ следует $L_1 \in P$.*

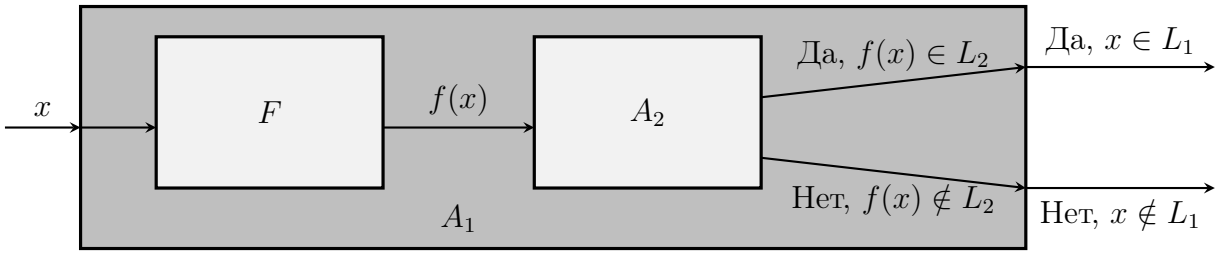

Рис. 2.5.4: Иллюстрация к доказательству Леммы 2.5.3

Доказательство. Пусть A_2 — алгоритм с полиномиальным временем выполнения, который распознает язык L_2, а F — полиномиально-временной алгоритм приведения, вычисляющий приводящую функцию f. Построим алгоритм A_1 с полиномиальным временем выполнения, который распознает язык L_1.

На Рис. 2.5.4 проиллюстрирован процесс построения алгоритма A_1. Для заданного набора входных данных $x \in \{0,1\}$ в алгоритме A_1 с помощью алгоритма F данные x преобразуются в $f(x)$, после чего с помощью алгоритма A_2 проверяется, является ли $f(x) \in L_2$. Результат работы алгоритма A_2 выводится алгоритмом A_1 в качестве ответа.

Корректность алгоритма A_1 следует из условия 2.5.1. Алгоритм выполняется в течение полиномиального времени, поскольку и алгоритм F, и алгоритм A_2 завершают свою работу за полиномиальное время. □

2.5.2 NP-полнота

Приведения в течение полиномиального времени служат формальным средством, позволяющим показать, что одна задача такая же по сложности, как и другая, по крайней мере с точностью до полиномиально-временного множителя. Другими словами, если $L_1 \leqslant_P L_2$, то сложность языка L_1 превышает сложность языка L_2 не более чем на полиномиальный множитель. Вот почему отношение «меньше или равно» для приведения является мнемоническим. Теперь можно определить множество NP-полных языков, являющихся самыми сложными задачами класса NP.

Определение 2.5.5 (Определение множества NP-полных языков)**.**
Язык $L \subseteq \{0,1\}$ является NP-полным, если выполняются перечисленные ниже условия:

1. $L \in NP$, *и*
2. $L' \leqslant_P L$ *для всех* $L' \in NP$.

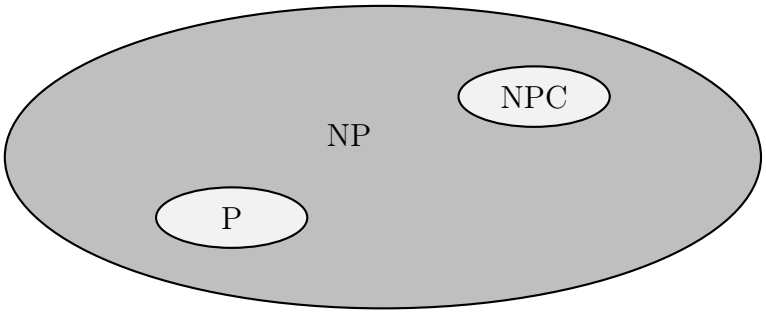

Рис. 2.5.6: Представление большинства специалистов в области вычислительных систем об отношении между классами P, NP и NPC; классы P и NPC полностью содержатся в классе NP, и $\mathrm{P} \cap \mathrm{NPC} = \varnothing$

Если язык L удовлетворяет свойству 2, но не обязательно удовлетворяет свойству 1, говорят, что L – NP-сложный (NP-hard). Определим также класс NPC как класс NP-полных языков.

Как показано в приведенной далее теореме, NP-полнота – ключевая проблема в разрешении вопроса о том, равны ли на самом деле классы P и NP.

Теорема 2.5.7. *Если некоторая NP-полная задача разрешима в течение полиномиального времени, то P = NP. Это эквивалентно утверждению, что если какая-нибудь задача из класса NP не решается в течение полиномиального времени, то никакая из NP-полных задач не решается в течение полиномиального времени.*

Доказательство. Предположим, что $L \in$ P и что $L \in$NPC. Для любого языка $L' \in$ NP, согласно свойству 2 определения NP-полноты, справедливо $L' \leqslant_P L$. Таким образом, в соответствии с Леммой 2.5.3, мы также имеем $L' \in$ P, что и доказывает первое утверждение теоремы.
Чтобы доказать второе утверждение, заметим, что оно является переформулировкой первого. □

Вот почему при исследовании вопроса $\mathrm{P} \neq \mathrm{NP}$ внимание акцентируется на NP-полных задачах. Большинство специалистов по теории вычислительных систем полагают, что $\mathrm{P} \neq \mathrm{NP}$, так что отношения между классами P, NP и NPC являются такими, как показано на

Рис. 2.5.6. Но если кто-нибудь додумается до алгоритма с полиномиальным временем работы, способного решить NP-полную задачу, тем самым будет доказано, что P = NP. Тем не менее, поскольку до сих пор такой алгоритм не обнаружен ни для одной NP-полной задачи, доказательство NP-полноты задачи является веским свидетельством ее трудноразрешимости.

2.6. Доказательство NP-полноты

Основой рассматриваемого в этом разделе метода, позволяющего доказать NP- полноту задачи, служит сформулированная ниже лемма.

Лемма 2.6.1. *Если язык L такой, что $L' \leqslant_P L$ для некоторого языка $L' \in NPC$, то язык L – NP-сложный. Более того, если $L \in NP$, то $L \in NPC$.*

Доказательство. Поскольку язык L' NP-полный, для всех $L'' \in$ NP выполняется соотношение $L'' \leqslant_P L'$. Согласно условию леммы $L' \leqslant_P L$, поэтому в силу транзитивности мы имеем $L'' \leqslant_P L$, что и показывает, что язык L – NP-сложный. Если $L \in$ NP, то мы также имеем $L \in$ NPC. □

Другими словами, если язык L', о котором известно, что он – NP-полный, удается свести к языку L, тем самым к этому языку неявно сводится любой язык класса NP. Таким образом, Лемма 2.6.1 позволяет сформулировать метод доказательства NP-полноты языка L, состоящий из перечисленных ниже этапов.

1. Доказывается, что $L \in$ NP.
2. Выбирается язык L', для которого известно, что он – NP-полный.
3. Описывается алгоритм, который вычисляет функцию f , отображающую каждый экземпляр $x \in \{0,1\}$ языка L' на экземпляр $f(x)$ языка L.
4. Доказывается, что для функции f соотношение $x \in L'$ выполняется тогда и только тогда, когда $f(x) \in L$ для всех $x \in \{0,1\}$.
5. Доказывается, что время работы алгоритма, вычисляющего функцию f, выражается полиномиальной функцией.

3. Практическая часть

3.1. NP-полные задачи

NP-полные задачи возникают в различных областях: в булевой логике, в теории графов, в арифметике, при разработке сетей, в теории множеств и разбиений, при хранении и поиске информации, при планировании вычислительных процессов, в математическом программировании, в алгебре и теории чисел, при создании игр и головоломок, в теории автоматов и языков, при оптимизации программ, в биологии, в химии, физике и т.п. В настоящем разделе с помощью методики приведения будет доказана NP-полнота различных задач, возникающих в теории графов и при разбиении множеств.

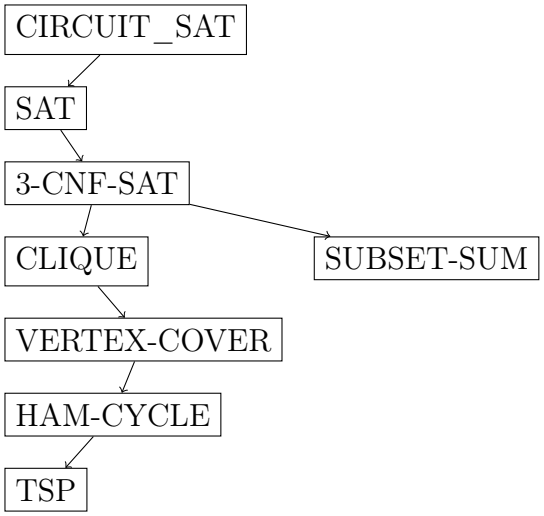

Рис. 3.1.1: Схема доказательств NP-полноты для задач

На Рис. 3.1.1 приводится схема доказательства NP-полноты, которая используется в этом разделе. Для каждого из представленных на рисунке языков NP-полнота доказывается путем приведения его к языку, на который указывает идущая от этого языка стрелка. В качестве корневого выступает NP-полный язык Circuit_Sat.

3.1.1 Задача о вершинном покрытии

Вершинное покрытие неориентированного графа $G = (V, E)$ – это такое подмножество $V' \subseteq V$, что если $(u, v) \in E$, то $u \in V'$ либо $v \in V'$ (либо справедливы оба эти соотношения). Другими словами, каждая вершина «покрывает» инцидентные ребра, а вершинное покрытие графа G — это множество вершин, покрывающих все ребра из множества E. ***Размером*** вершинного покрытия называется количество содержащихся в нем вершин.

Задача о вершинном покрытии заключается в том, что-бы найти в заданном графе вершинное покрытие минимального размера. Переформулируем эту задачу оптимизации в виде задачи принятия решения, в которой требуется определить, содержит ли граф вершинное покрытие заданного размера k. Определим язык

Vertex_Cover = $\langle G, k\rangle$: граф G имеет вершинное покрытие размера k.
В сформулированной ниже теореме доказывается, что эта задача является NP- полной.

Теорема 3.1.2. *Задача о вершинном покрытии является NP-полной.*

Доказательство. Докажем, что задача о вершинном покрытии NP-сложная, для чего покажем справедливость соотношения Clique $\leqslant_P$ Vertex_Cover. Это приведение основывается на понятии «дополнения» графа. Дополнение неориентированного графа $G = (V, E)$ определяется как $G = (V, E)$, где $E = \{(u, v) : u, v \in V, u \neq v$ и $(u, v) \notin E\}$. Другими словами, G — это граф, со держащий те ребра, которые не содержатся в графе G.

Алгоритм приведения в качестве входных данных получает экземпляр задачи о клике $\langle G, k\rangle$. В этом алгоритме вычисляется дополнение $\overline{G}$, что легко осуществить в течение полиномиального времени. Выходом алгоритма приведения является экземпляр $\langle \overline{G}, |V| - k\rangle$ задачи о вершинном покрытии. Чтобы завершить доказательство, покажем, что это преобразование действительно является приведением: граф G содержит клику размера k тогда и только тогда, когда граф $\overline{G}$ имеет вершинное покрытие размером $|V| - k$.

Справедливо и обратное. Предположим, что граф $\overline{G}$ имеет вершинное покрытие $V' \subseteq V$, где $|V'| = |V| - k$. Тогда для всех пар вершин $u, v \in V$ из $(u, v) \in \overline{E}$ следует, что или $u \in V'$, или $v \in V'$, или справедливы оба эти утверждения. Обращение следствия дает, что для всех пар вершин $u, v \in V$, если $u \notin V'$ и $v \notin V'$, то $(u, v) \in E$. Другими словами,$V - V'$ – это клика, а ее размер равен $|V| - |V'| = k$. □

Поскольку задача Vertex_Cover является NP-полной, маловероятным представляется то, что удастся разработать алгоритм поиска вершинного покрытия минимального размера за полиномиальное время. Размер вершинного покрытия, полученного в результате работы «приближенного алгоритма», с полиномиальным временем работы, позволяющий находить «приближенные» решения этой задачи, не более чем в 2 раза превышает размер минимального вершинного покрытия.

Таким образом, не стоит лишать себя надежды только из-за того, что задача NP-полная. Может оказаться, что для нее существует приближенный алгоритм с полиномиальным временем работы, позволяющий получать решения, близкие к оптимальным.

4. Заключение

В ходе выполнения курсовой работы мне удалось изучить понятие NP-полных задач и рассмотреть их примеры; узнать о классах P и NP, а также о понятии NP-полноты и как её доказывать; рассмотреть доказательство NP-полноты на примере конкретной задачи.

Изучив данную тему мы можем сделать вывод, что знание, относится ли задача к классу NP-полных, достаточно важно в анализе данных, поскольку в аналитических проектах приходится иметь дело с очень большими объемами исходных данных. Поэтому необходимо планировать вычислительные ресурсы и время для решения задач анализа, и если они неприемлемо возрастают, искать альтернативные пути решения (например, выполнять декомпозицию сложной задачи на ряд простых или искать приближённое решение).

5. Список литературы

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. «АЛГОРИТМЫ. ПОСТРОЕНИЕ И АНАЛИЗ», Глава 34. NP-полнота, 2-е изд. – М.: «Вильямс», 2006.

2. Джон Хопкрофт, Раджив Мотвани, Джеффри Ульман. «Введение в теорию автоматов, языков и вычислений», М.: «Вильямс», 2002.

3. Гэри М., Джонсон Д. «Вычислительные машины и труднорешаемые задачи», М.: Мир, 1982.

4. https://alexeykalina.github.io/technologies/np-completeness.html

5. https://wiki.loginom.ru/articles/np-complete.html