Algorithm Design XVII

NP Problem III

Guoqiang Li
School of Software

Shanghai Jiao Tong University

The Reductions

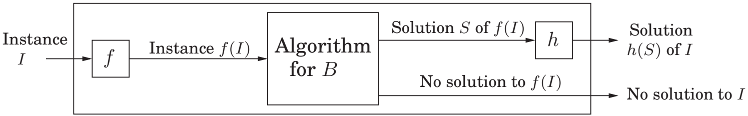
Reduction Between Search Problems

A reduction from A to B is a polynomial time algorithm f that transforms any instance I of A into an instance $f(I)$ of B

Together with another polynomial time algorithm h that maps any solution S of $f(I)$ back into a solution $h(S)$ of I.

If $f(I)$ has no solution, then neither does I.

These two translation procedures f and h imply that any algorithm for B can be converted into an algorithm for A.



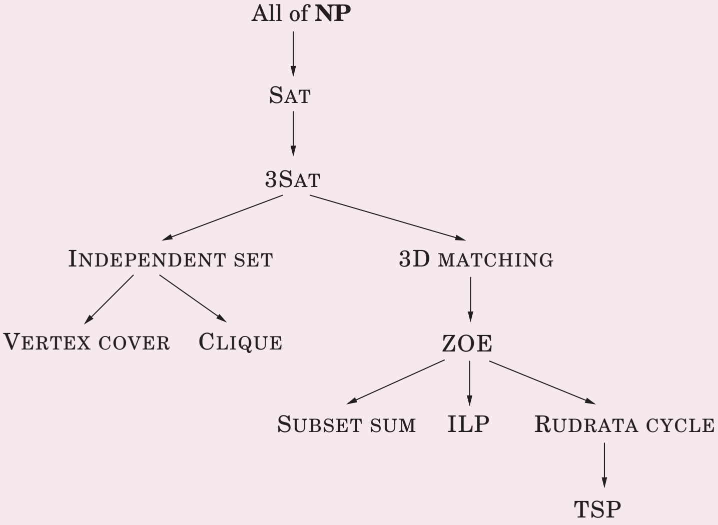
The Two Ways to Use Reductions

Assume there is a reduction from a problem A to a problem B.

$$A \to B$$

- If we can solve B efficiently, then we can also solve A efficiently.
- If we know A is hard, then B must be hard too.

If $A \to B$ and $B \to C$, then $A \to C$.


All of **NP**
SAT
3SAT
INDEPENDENT SET
3D MATCHING
VERTEX COVER
CLIQUE
ZOE
SUBSET SUM
ILP
RUDRATA CYCLE
TSP

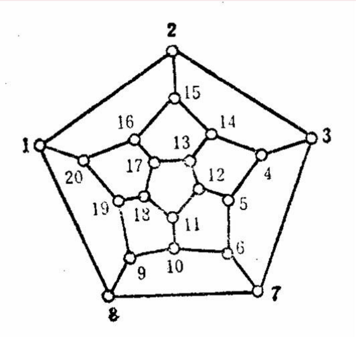
RUDRATA PATH $\rightarrow$ RUDRATA CYCLE

Rudrata Cycle

SHANGHAI JIAO TONG UNIVERSITY

RUDRATA CYCLE

Given a graph, find a cycle that visits each vertex exactly once.



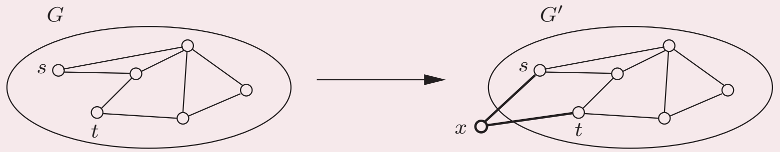
Rudrata (s,t)-path $\rightarrow$ Rudrata cycle

A Rudrata (s,t)-path problem specifies two vertices s and t and wants a path starting at s and ending at t that goes through each vertex exactly once.

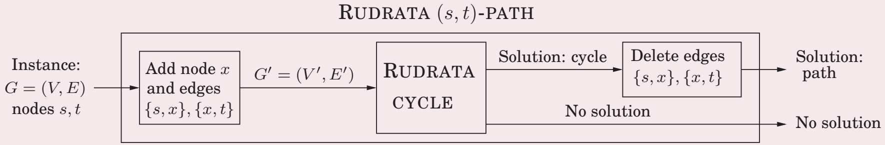
Q: Is it possible that Rudrata cycle is easier than Rudrata (s,t)-path?

The reduction maps an `instance` G of Rudrata (s,t)-Path into an `instance` G' of Rudrata cycle as follows: G' is G with an additional vertex x and two new edges $\{s,x\}$ and $\{x,t\}$.

Rudrata (s,t)-path $\rightarrow$ Rudrata cycle



Rudrata (s, t)-path $\rightarrow$ Rudrata cycle



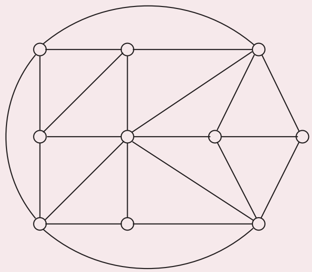
3SAT $\rightarrow$ Independent set

3 SAT

The instances of 3SAT, is set of clauses, each with three or fewer literals.

$$(x \vee y \vee z)(x \vee \overline{y})(y \vee \overline{z})(z \vee \overline{x})(\overline{x} \vee \overline{y} \vee \overline{z})$$

Independent Set



INDEPENDENT SET: Given a graph G and an integer g, find g vertices, no two of which have an edge between them.

True Assignment

To form a satisfying truth assignment we must pick one literal from each clause and give it the value `true`.

The choices must be consistent, if we choose $\overline{x}$ in one clause, we cannot choose x in another.

Solution: put an edge between any two vertices that correspond to opposite literals.

Clause

Represent a clause, say $(x \vee \overline{y} \vee z)$, by a triangle, with vertices labeled $x, \overline{y}, z$.

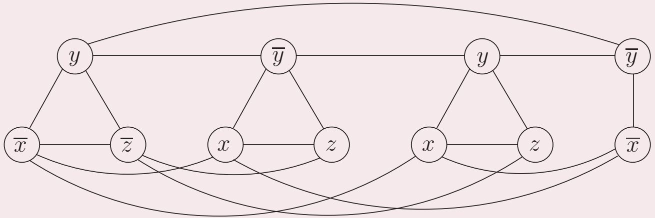
Because a triangle has its three vertices maximally connected, and thus forces to pick only one of them for the independent set.

3SAT $\rightarrow$ INDEPENDENT SET

Given an `instance` I of 3SAT, create an `instance` (G, g) of INDEPENDENT SET as follows,

- A triangle for each clause, with vertices labeled by the clause's literals.
- Additional edges between any two vertices that represent opposite literals.
- The goal g is set to the number of clauses.

3SAT $\rightarrow$ Independent set



$$(\overline{x} \vee y \vee \overline{z})(x \vee \overline{y} \vee z)(x \vee y \vee z)(\overline{x} \vee \overline{y})$$

SAT $\rightarrow$ 3SAT

SAT $\to$ 3SAT

This is an interesting and common kind of reduction, from a problem to a special case of itself.

Given an `instance` I of SAT, use exactly the same `instance` for 3SAT, except that any clause with more than three literals,

$$(a_1 \vee a_2 \vee \ldots \vee a_k)$$

is replaced by a set of clauses,

$$(a_1 \vee a_2 \vee y_1)(\overline{y_1} \vee a_3 \vee y_2)(\overline{y_2} \vee a_4 \vee y_3)\ldots(\overline{y_{k-3}} \vee a_{k-1} \vee a_k)$$

where the y_i's are new variables.

The reduction is in polynomial and I' is equivalent to I in terms of satisfiability.

SAT $\to$ 3SAT

$$\left\{ \begin{array}{c} (a_1 \vee a_2 \vee \cdots \vee a_k) \\ \text{is satisfied} \end{array} \right\} \Longleftrightarrow \left\{ \begin{array}{c} \text{there is a setting of the } y_i\text{'s for which} \\ (a_1 \vee a_2 \vee y_1)\ (\overline{y}_1 \vee a_3 \vee y_2)\ \cdots\ (\overline{y}_{k-3} \vee a_{k-1} \vee a_k) \\ \text{are all satisfied} \end{array} \right\}$$

Suppose that the clauses on the right are all satisfied. Then at least one of the literals $a_1, \ldots, a_k$ must be `true`. Otherwise y_1 would have to be `true`, which would in turn force y_2 to be `true`, and so on.

Conversely, if $(a_1 \vee a_2 \vee \ldots \vee a_k)$ is satisfied, then some a_i must be `true`. Set $y_1, \ldots, y_{i-2}$ to `true` and the rest to `false`.

SAT $\rightarrow$ 3SAT

3SAT remains hard even under the further restriction that no variable appears in more than three clauses.

Suppose that in the 3SAT `instance`, variable x appears in $k > 3$ clauses. Then replace its first appearance by x_1, its second by x_2, and so on, replacing each of its k appearances by a different new variable.

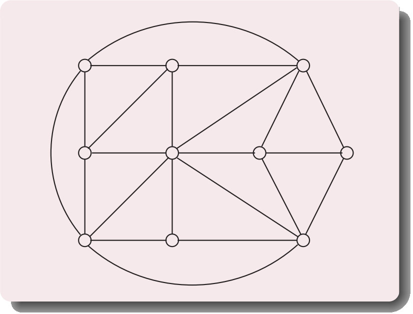
Finally, add the clauses

$$(\overline{x_1} \vee x_2)(\overline{x_2} \vee x_3)\ldots(\overline{x_k} \vee x_1)$$

In the new formula no variable appears more than three times (and in fact, no literal appears more than twice).

INDEPENDENT SET $\rightarrow$ VERTEX COVER

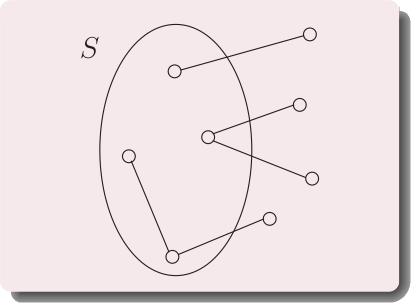
Vertex Cover



VERTEX COVER: Given a graph G and an integer b, find b vertices cover (touch) every edge.

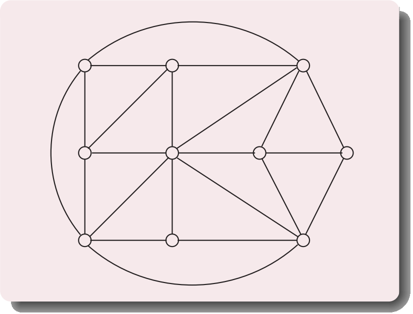
INDEPENDENT SET $\rightarrow$ VERTEX COVER

A set of nodes S is a vertex cover of graph $G = (V, E)$ iff the remaining nodes, $V - S$, are an independent set of G.



INDEPENDENT SET $\rightarrow$ CLIQUE

Clique



CLIQUE: Given a graph G and an integer g, find g vertices such that all possible edges between them are present.

INDEPENDENT SET $\to$ CLIQUE

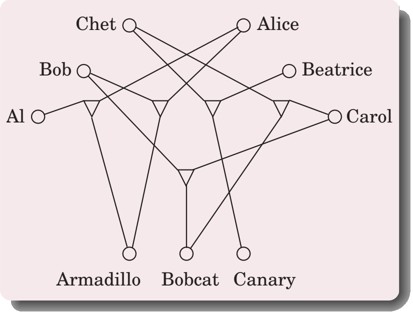
The complement of a graph $G = (V, E)$ is $\overline{G} = (V, \overline{E})$, where $\overline{E}$ contains precisely those unordered pairs of vertices that are not in E. A set of nodes S is an independent set of G iff S is a clique of $\overline{G}$.

Therefore, we can reduce INDEPENDENT SET to CLIQUE by mapping an instance (G, g) of INDEPENDENT SET to the corresponding instance $(\overline{G}, g)$ of CLIQUE.

3SAT $\rightarrow$ 3D MATCHING

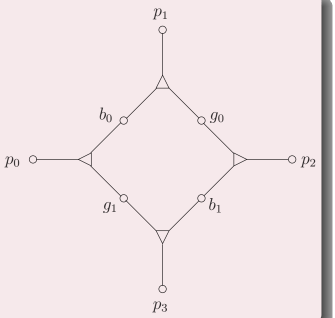
Three-Dimensional Matching

3D MATCHING: There are n boys, n girls, and n pets. The compatibilities are specified by a set of triples, each containing a boy, a girl, and a pet.
A triple (b, g, p) means that boy b, girl g, and pet p get along well together.
To find n disjoint triples and thereby create n harmonious households.



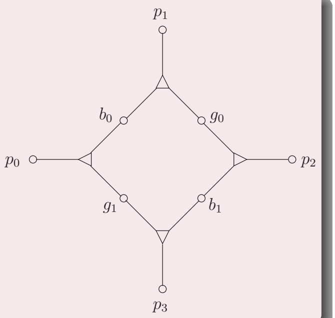
3SAT $\rightarrow$ 3D MATCHING

Consider a set of four triples, each represented by a triangular node joining a boy, girl, and pet.
Any matching must contain either the two triples $(b_0, g_1, p_0), (b_1, g_0, p_2)$ or $(b_0, g_0, p_1), (b_1, g_1, p_3)$.



3SAT $\rightarrow$ 3D MATCHING

Therefore, this "gadget" has two possible states: it behaves like a Boolean variable.
Transform an `instance` of 3SAT to one of 3D MATCHING, by creating a gadget for each variable x.



3SAT $\rightarrow$ 3D MATCHING

For each clause c introduce a new boy b_c and a new girl g_c.

E.g., $c = (x \vee \overline{y} \vee z)$, b_c, g_c will be involved in three triples, one for each literal in the clause.

And the pets in these triples must reflect the three ways whereby the clause can be satisfied:

1. $x = \texttt{true}$,
2. $y = \texttt{false}$,
3. $z = \texttt{true}$.

For $x = \texttt{true}$, we have the triple (b_c, g_c, p_{x1}), where p_{x1} is the pet p_1 in the gadget for x.

- If $x = \texttt{true}$, then b_{x0} is matched with g_{x1} and b_{x1} with g_{x0}, and so pets p_{x0} and p_{x2} are taken.
- If $x = \texttt{false}$, then p_{x1} and p_{x3} are taken, and so g_c and b_c cannot be accommodated.

We do the same thing for the other two literals, which yield triples involving b_c and g_c with either p_{y0} or p_{y2} and with either p_{z1} or p_{z3}.

3SAT $\to$ 3D MATCHING

We have to make sure that for every occurrence of a literal in a clause c there is a different pet to match with b_c and g_c.

This is easy: an earlier reduction guarantees that no literal appears more than twice, and so each variable gadget has enough pets, two for negated occurrences and two for positive.

The last problem remains: in the matching defined so far, some pets may be left unmatched.

If there are n variables and m clauses, then $2n - m$ pets will be left unmatched.

Add $2n - m$ new boy-girl couples that are “generic animal-lovers”, and match them by triples with all the pets!

3D MATCHING $\rightarrow$ ZOE

Zero-One Equations

ZOE

Given an $m \times n$ matrix A with $0-1$ entries, and find a $0-1$ vector $\mathbf{x} = (x_1, \ldots, x_n)$ such that the m equations $A\mathbf{x} = 1$; are satisfied.

3D MATCHING → ZOE

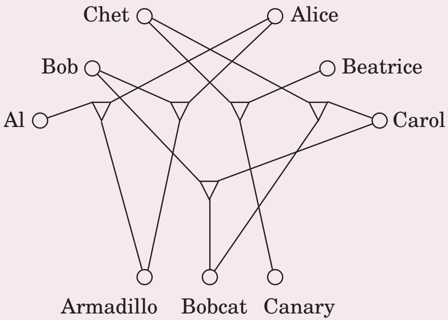
Assume in 3D MATCHING, there are m boys, m girls, m pets, and n boy-girl-pet triples.

We have $0-1$ variables, $x_1, \ldots, x_n$, one per triple, where $x_i = 1$ means that the i-th triple is chosen for the matching, and $x_i = 0$ means that it is not chosen.

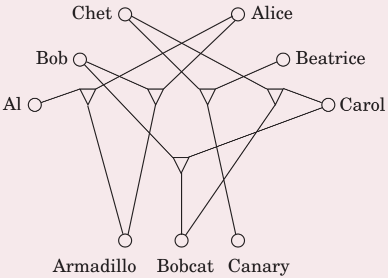
For each boy, girl, or pet, suppose that the triples containing him (or her, or it) are those numbered $j_1, j_2, \ldots, j_k$; the appropriate equation is then

$$x_{j_1} + x_{j_2} + \ldots + x_{j_k} = 1$$

3D MATCHING → ZOE



3D MATCHING $\to$ ZOE



$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

ZOE $\rightarrow$ SUBSET SUM

Subset Sum

SUBSET SUM

SUBSET SUM: Find a subset of a given set of integers that adds up to exactly W.

ZOE $\rightarrow$ SUBSET SUM

This is a reduction between two special cases of ILP:

- One with many equations but only $0-1$ coefficients;
- The other with a single equation but arbitrary integer coefficients.

The reduction is based on a simple and time-honored idea: $0-1$ vectors can encode numbers!

If the columns is regarded as binary integers, a subset of the integers corresponds to the columns of A that add up to the binary integer $11\ldots1$.

This is an `instance` of SUBSET SUM. The reduction seems complete!

An Example

SHANGHAI JIAO TONG UNIVERSITY

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

ZOE $\rightarrow$ Subset Sum

Except for one detail: carry.

E.g., 5-bit binary integers can add up to $11111 = 31$, for example, $5 + 6 + 20 = 31$ or, in binary,

$$00101 + 00110 + 10100 = 11111$$

even when the sum of the corresponding vectors is not $(1, 1, 1, 1, 1)$.

Solution: The column vectors not as integers in base 2, but as integers in base $n + 1$, one more than the number of columns.

At most n integers are added, and all their digits are 0 and 1 There is no carry anymore.

ZOE $\rightarrow$ ILP

Special Cases

3SAT is a special case of SAT, or, SAT is a generalization of 3SAT.

By special case we mean that the `instances` of 3SAT are a subset of the `instances` of SAT.

There is a reduction from 3SAT to SAT, where the input has no transformation, and the solution to the target instance also kept unchanged.

A useful and common way of establishing that a problem is NP-complete: it is a generalization of a known NP-complete problem.

E.g., the SET COVER problem is NP-complete because it is a generalization of VERTEX COVER.

ZOE $\to$ ILP

In ILP we are looking for an integer vector $\mathbf{x}$ that satisfies $A\mathbf{x} \leq b$, for given matrix A and vector b.

To write an `instance` of ZOE in this precise form, we need to rewrite each equation of the ZOE `instance` as two inequalities, and to add for each variable x_i the inequalities $x_i \leq 1$ and $-x_i \leq 0$.

ZOE $\rightarrow$ RUDRATA CYCLE

ZOE $\rightarrow$ RUDRATA CYCLE

In RUDRATA CYCLE, seek a cycle in a graph that visits every vertex exactly once.

In ZOE, given an $m \times n$ matrix A with $0-1$ entries, and find a $0-1$ vector $\mathbf{x} = (x_1, \ldots, x_n)$ such that the m equations $A\mathbf{x} = 1$; are satisfied.

ZOE $\rightarrow$ RUDRATA CYCLE

We will prove it NP-complete in two stages:

1. Firstly, reduce ZOE to a generalization of RUDRATA CYCLE, called RUDRATA CYCLE WITH PAIRED EDGES.
2. Secondly, get rid of the extra features of that problem and reduce it to the plain RUDRATA CYCLE.

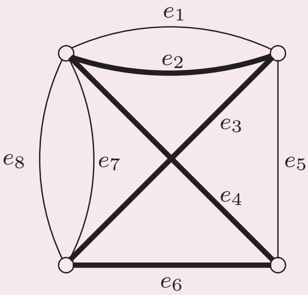
Rudrata Cycle with Paired Edges

Given a graph $G = (V, E)$ and a set $C \subseteq E \times E$ of pairs of edges. Find a cycle that,

1. visits all vertices once,
2. for every pair of edges (e, e') in C, traverses either edge e or edge e' exactly one of them.

Notice that two or more parallel edges between two nodes are allowed.

An Example



$$C = \{(e_1, e_3), (e_5, e_6), (e_4, e_5), (e_3, e_7), (e_3, e_8)\}$$

ZOE $\rightarrow$ RUDRATA CYCLE WITH PAIRED EDGES

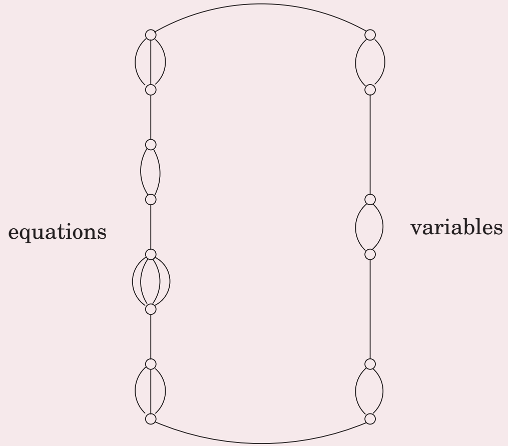
Given an `instance` of ZOE, $A\mathbf{x} = 1$, where A is an $m \times n$ matrix with $0 - 1$ entries, the graph is as follows

- A cycle that connects $m + n$ collections of parallel edges.
- Each variable x_i has two parallel edges, for $x_i = 1$ and $x_i = 0$).
- Each equation $x_{j_1} + \ldots + x_{j_k} = 1$ involving k variables has k parallel edges, one for every variable appearing in the equation.

Any RUDRATA CYCLE traverses the $m + n$ collections of parallel edges one by one, choosing one edge from each collection.

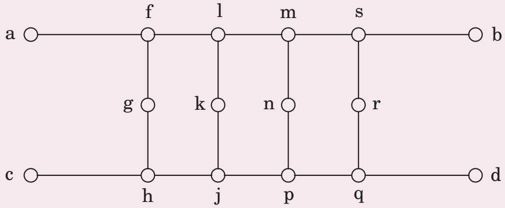
The cycle "chooses" for each variable a value 0 or 1 and, for each equation, a variable appearing in it.

ZOE $\rightarrow$ Rudrata Cycle with Paired Edges



Get Rid of Edge Pairs

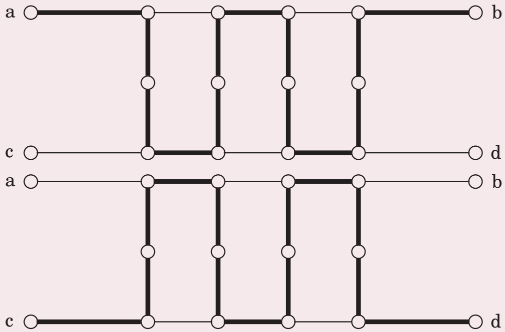
Consider the graph, and suppose it is a part of a larger graph G in such a way that only the four endpoints a, b, c, d touch the rest of the graph.



Get Rid of Edge Pairs

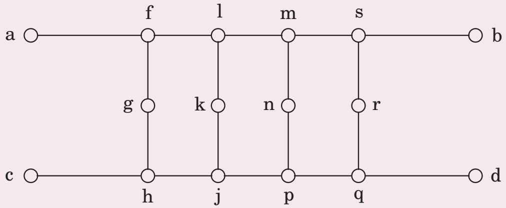
SHANGHAI JIAO TONG UNIVERSITY

We claim that this graph has the following important property: in any RUDRATA CYCLE of G the subgraph shown must be traversed in one of the two ways.



Get Rid of Edge Pairs

This gadget behaves just like two edges $\{a, b\}$ and $\{c, d\}$ that are paired up in the RUDRATA CYCLE WITH PAIRED EDGES.



$$C = \{(\{a, b\}, \{c, d\})\}$$

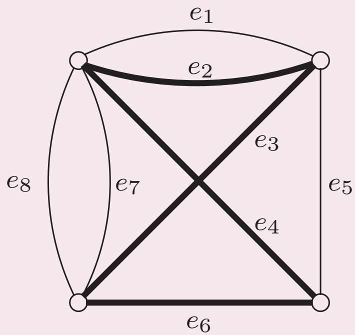
Go through the pairs in C one by one. To get rid of each pair $(\{a, b\}, \{c, d\})$ by replacing the two edges with the gadget.

For any other pair in C that involves $\{a, b\}$, replace the edge $\{a, b\}$ with the new edge $\{a, f\}$, where f is from the gadget.

Similarly, $\{c, h\}$ replaces $\{c, d\}$.

The RUDRATA CYCLES in the resulting graph will be in one-to-one correspondence with the RUDRATA CYCLES in the original graph that conform to the constraints in C.

An Example



$$C = \{(e_1, e_3), (e_5, e_6), (e_4, e_5), (e_3, e_7), (e_3, e_8)\}$$

Rudrata cycle $\rightarrow$ TSP

Rudrata cycle $\rightarrow$ TSP

Given a graph $G = (V, E)$, construct the `instance` of the TSP:

- The set of nodes is the same as V.
- The distance between cities u and v is 1 if $\{u, v\}$ is an edge of G and $1 + \alpha$ otherwise, for some $\alpha > 1$ to be determined.
- The budget of the TSP `instance` is $|V|$.

If G has a Rudrata cycle, then the same cycle is also a tour within the budget of the TSP `instance`.

If G has no Rudrata cycle, then there is no solution: the cheapest possible TSP tour has cost at least $n + \alpha$.

Rudrata cycle $\rightarrow$ TSP

If $\alpha = 1$, then all distances are either 1 or 2, and so this `instance` of the TSP satisfies the triangle inequality: if i, j, k are cities, then

$$d_{ij} + d_{jk} \geq d_{ik}$$

This is a special case of the TSP which is in a certain sense easier, since it can be efficiently approximated.

RUDRATA CYCLE $\rightarrow$ TSP

If α is large, then the resulting `instance` of the TSP may not satisfy the triangle inequality, and has another important property.

This important gap property implies that, unless P = NP, no approximation algorithm is possible.

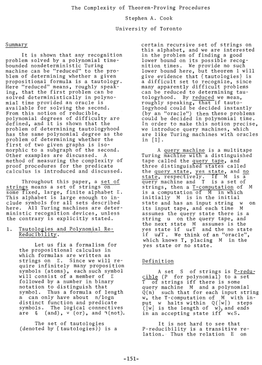
ANY PROBLEM $\rightarrow$ SAT

home reading!

The “first” NP-complete problem

Theorem (Cook 1971, Levin 1973)

*SAT is **NP**-complete.*



The Complexity of Theorem-Proving Procedures

Stephen A. Cook

University of Toronto

Summary

It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology. Here "reduced" means, roughly speaking, that the first problem can be solved deterministically in polynomial time provided an oracle is available for solving the second. From this notion of reducible, polynomial degrees of difficulty are defined, and it is shown that the problem of determining tautologyhood has the same polynomial degree as the problem of determining whether the first of two given graphs is isomorphic to a subgraph of the second. Other examples are discussed. A method of measuring the complexity of proof procedures for the predicate calculus is introduced and discussed.

Throughout this paper, a set of strings means a set of strings on some fixed, large, finite alphabet Σ. This alphabet is large enough to include symbols for all sets described here. All Turing machines are deterministic recognition devices, unless the contrary is explicitly stated.

1. Tautologies and Polynomial Re-Reducibility.

Let us fix a formalism for the propositional calculus in which formulas are written as strings on Σ. Since we will require infinitely many proposition symbols (atoms), each such symbol will consist of a member of Σ followed by a number in binary notation to distinguish that symbol. Thus a formula of length n can only have about n/logn distinct function and predicate symbols. The logical connectives are & (and), ∨ (or), and ¬(not).

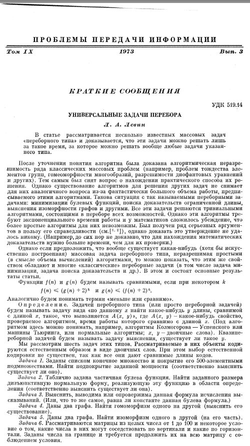
The set of tautologies (denoted by {tautologies}) is a certain recursive set of strings on this alphabet, and we are interested in the problem of finding a good lower bound on its possible recognition times. We provide no such lower bound here, but theorem 1 will give evidence that {tautologies} is a difficult set to recognize, since many apparently difficult problems can be reduced to determining tautologyhood. By reduced we mean, roughly speaking, that if tautologyhood could be decided instantly (by an "oracle") then these problems could be decided in polynomial time. In order to make this notion precise, we introduce query machines, which are like Turing machines with oracles in [1].

A query machine is a multitape Turing machine with a distinguished tape called the query tape, and three distinguished states called the query state, yes state, and no state, respectively. If M is a query machine and T is a set of strings, then a T-computation of M is a computation of M in which initially M is in the initial state and has an input string w on its input tape, and each time M assumes the query state there is a string u on the query tape, and the next state M assumes is the yes state if u∊T and the no state if u∉T. We think of an "oracle", which knows T, placing M in the yes state or no state.

Definition

A set S of strings is P-reducible (P for polynomial) to a set T of strings iff there is some query machine M and a polynomial Q(n) such that for each input string w, the T-computation of M with input w halts within Q(|w|) steps (|w| is the length of w), and ends in an accepting state iff w∊S.

It is not hard to see that P-reducibility is a transitive relation. Thus the relation E on



ПРОБЛЕМЫ ПЕРЕДАЧИ ИНФОРМАЦИИ

Том IX 1973 Вып. 3

КРАТКИЕ СООБЩЕНИЯ

УДК 519.14

УНИВЕРСАЛЬНЫЕ ЗАДАЧИ ПЕРЕБОРА

Л. А. Левин

В статье рассматривается несколько известных массовых задач «переборного типа» и доказывается, что эти задачи можно решать лишь за такое время, за которое можно решать вообще любые задачи указанного типа.

После уточнения понятия алгоритма была доказана алгоритмическая неразрешимость ряда классических массовых проблем (например, проблем тождества элементов групп, гомеоморфности многообразий, разрешимости диофантовых уравнений и других). Тем самым был снят вопрос о нахождении практического способа их решения. Однако существование алгоритмов для решения других задач не снимает для них аналогичного вопроса из-за фантастически большого объема работы, предписываемого этими алгоритмами. Такова ситуация с так называемыми переборными задачами: минимизации булевых функций, поиска доказательств ограниченной длины, выяснения изоморфности графов и другими. Все эти задачи решаются тривиальными алгоритмами, состоящими в переборе всех возможностей. Однако эти алгоритмы требуют экспоненциального времени работы и у математиков сложилось убеждение, что более простые алгоритмы для них невозможны. Был получен ряд серьезных аргументов в пользу его справедливости (см.[1, 2]), однако доказать это утверждение не удалось никому. (Например, до сих пор не доказано, что для нахождения математических доказательств нужно больше времени, чем для их проверки.)

Однако если предположить, что вообще существует какая-нибудь (хотя бы искусственно построенная) массовая задача переборного типа, неразрешимая простыми (в смысле объема вычислений) алгоритмами, то можно показать, что этим же свойством обладают и многие «классические» переборные задачи (в том числе задача минимизации, задача поиска доказательств и др.). В этом и состоят основные результаты статьи.

Функции $f(n)$ и $g(n)$ будем называть сравнимыми, если при некотором k

$$f(n) \leqslant (g(n)+2)^k \quad \text{и} \quad g(n) \leqslant (f(n)+2)^k.$$

Аналогично будем понимать термин «меньше или сравнимо».

Определение. Задачей переборного типа (или просто переборной задачей) будем называть задачу вида «по данному x найти какое-нибудь y длины, сравнимой с длиной x, такое, что выполняется $A(x, y)$», где $A(x, y)$ – какое-нибудь свойство, проверяемое алгоритмом, время работы которого сравнимо с длиной x. (Под алгоритмом здесь можно понимать, например, алгоритмы Колмогорова — Успенского или машины Тьюринга, или нормальные алгоритмы; x, y – двоичные слова). Квазипереборной задачей будем называть задачу выяснения, существует ли такое y.

Мы рассмотрим шесть задач этих типов. Рассматриваемые в них объекты кодируются естественным образом в виде двоичных слов. При этом выбор естественной кодировки не существен, так как все они дают сравнимые длины кодов.

Задача 1. Заданы списком конечное множество и покрытие его 500-элементными подмножествами. Найти подпокрытие заданной мощности (соответственно выяснить существует ли оно).

Задача 2. Таблично задана частичная булева функция. Найти заданного размера дизъюнктивную нормальную форму, реализующую эту функцию в области определения (соответственно выяснить существует ли она).

Задача 3. Выяснить, выводима или опровержима данная формула исчисления высказываний. (Или, что то же самое, равна ли константе данная булева формула.)

Задача 4. Даны два графа. Найти гомоморфизм одного на другой (выяснить его существование).

Задача 5. Даны два графа. Найти изоморфизм одного в другой (на его часть).

Задача 6. Рассматриваются матрицы из целых чисел от 1 до 100 и некоторое условие о том, какие числа в них могут соседствовать по вертикали и какие по горизонтали. Заданы числа на границе и требуется продолжить их на всю матрицу с соблюдением условия.