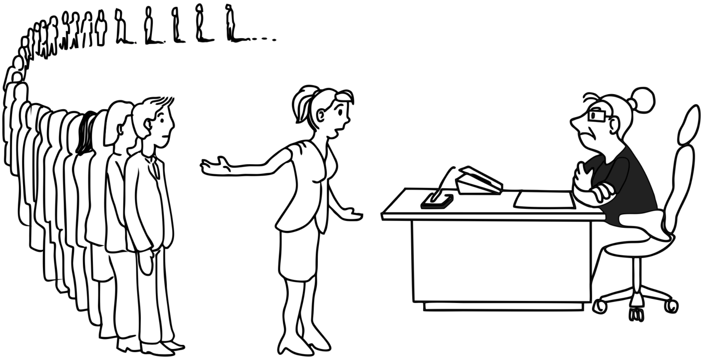
Automata, Languages and Computation

Chapter 10 : Intractability

Master Degree in Computer Engineering
University of Padua
Lecturer : Giorgio Satta

Lecture based on material originally developed by :
Gösta Grahne, Concordia University

Intractability



"I can't find an efficient algorithm, but neither can all these famous people."

Introduction

After investigating what can be decided, let us focus on what can be computed efficiently, that is, in **polynomial time**

The problems that can be solved in polynomial time on a computer coincide with the problems solvable on TMs in polynomial time

This follows from simulation of RAM architecture by a TM, which was not presented in our lectures

We introduce

- a new type of reduction
- the theory of intractability

All results are based on the $\mathcal{P} \neq \mathcal{NP}$, which has not yet been proven or falsified

1. Classes $\mathcal{P}$ and $\mathcal{NP}$: we introduce the theory of intractability and the notion of polynomial time reduction

2. An NP-complete problem : we introduce the SAT problem

3. Restricted version of the satisfiability problem : we introduce a very common variant of SAT

4. Other NP-complete problems : we investigate problems of practical importance that are difficult to solve

Computational complexity of a TM

A TM M has **time complexity** $T(n)$ if, for any input string w with $|w| = n$, M halts after at most $T(n)$ computational steps

Example : $T(n) = 5n^2 + 3n$, or $T(n) = 4^n + 3n^2$

A language (decision problem) L belongs to the **class $\mathcal{P}$** if there exists a **polynomial** $T(n)$ such that $L = L(M)$ for some (deterministic) TM M with time complexity $T(n)$

Computational complexity of a TM

If the time complexity is not polynomial, we usually say that the time complexity is **exponential**, even if $T(n)$ may not be an exponential function

Example : $T(n) = n^{\log_2 n}$ grows faster than any polynomial, but slower than any exponential $2^{c \cdot n}$

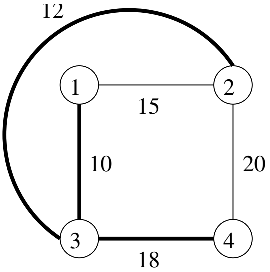
Polynomial algorithms

The **spanning tree** of a connected graph G is a subset of G's arcs without cycles, that connects all nodes of G

The **minimum weight spanning tree** problem (MWST) has as input a graph G with integer weights at its arcs. The problem is to find a spanning tree with the minimum sum of the weights on the arcs

Polynomial algorithms

Example :



The spanning tree with minimum weight is indicated by the arcs in boldface

Polynomial algorithms

The minimum weight spanning tree can be found using **Kruskal algorithm**

- for each node p, keep track of the connected component to which p belongs with respect to the partial spanning tree
- among all arcs that have not yet been processed, consider arc (p, q) with lowest weight; if (p, q) connects two separate connected components
  - add (p, q) to the partial spanning tree
  - merge the two connected components by updating the involved nodes

Polynomial algorithms

Let m be the number of nodes and e be the number of arcs in the graph

Kruskal algorithm has a very simple implementation on a computer, running in time $\mathcal{O}(e(e+m))$

- for each step : choose an arc in time $\mathcal{O}(e)$, merge the two components in time $\mathcal{O}(m)$
- there are at most $\mathcal{O}(e)$ steps

The execution time is therefore **polynomial** in the input size, which we can consider as $(e+m)$

Computational complexity analysis

Analyzing the computational complexity of a TM presents two difficulties, as compared to the analysis of a computer algorithm

Issue 1 : An algorithm can output a structure, while a TM just accepts or rejects its input

We can recast a search problem through a decision problem
Example : Given a graph G and an integer W, is there a spanning tree with weight not exceeding W ?

The decision problem usually provides a **lower bound** for the computational complexity of the search problem, which can be used for intractability to prove that a problem is difficult

Computational complexity analysis

Issue 2 : Algorithms have input alphabets of unlimited size, while TMs have finite input alphabet

Example : The set of nodes of a graph can be represented by **atomic** symbols $p_1, \ldots, p_{27}, \ldots, p_{225}$;
a TM instead requires some encoding of each symbol, such as $p_1 = p1$, $p_{27} = p11011, \ldots, p_{255} = p11111111$

Symbol encoding introduces a growth factor usually equal to the **logarithm** of the number of symbols. This factor is not relevant when we study the class of polynomial problems

Nondeterministic polynomial time

A language (decision problem) L belongs to the **class $\mathcal{NP}$** if there exists a polynomial function $T(n)$ such that $L = L(M)$ for some NTM M with time complexity $T(n)$

We can always assume that M performs exactly $T(n)$ moves **for every input** of length n: to this end, we can simulate a clock function on a special tape track

Nondeterministic polynomial time

$\mathcal{P} \subseteq \mathcal{NP}$: every TM is also a NTM

$\mathcal{P} \neq \mathcal{NP}$?

A polynomial NTM can perform an **exponential** number of computations "simultaneously". Therefore it is commonly assumed that $\mathcal{P} \neq \mathcal{NP}$, but no one has ever been able to prove this statement

Nondeterministic polynomial time algorithms

A **Hamiltonian circuit** in a graph G is a sorting of G's nodes that forms a cycle

The **traveling salesman problem**, TSP for short, takes as input a graph G with integer weights on the arcs and a weight limit W. The problem asks whether G has a Hamiltonian circuit with total weight not exceeding W

Nondeterministic polynomial time algorithms

In a graph with m nodes, the number of distinct cycles grows with $\mathcal{O}(m!)$, which grows faster than $2^{c \cdot m}$

Any deterministic algorithm for TSP *seems* to need to examine at least an **exponential** number of cycles and compute the associated weights

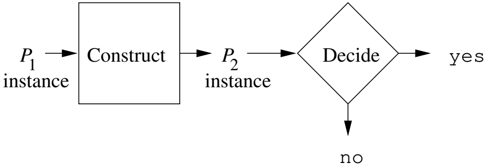
With a nondeterministic algorithm, we can

- **choose** in each branch of the computation a permutation π of the nodes of G
- verify the existence of an associated cycle p_π
- compute the associated weight of p_π and compare with W

This takes polynomial time, so the problem is in the class $\mathcal{NP}$

Polynomial time reductions

To show that a problem P_2 cannot be solved in polynomial time, we reduce a problem $P_1 \notin \mathcal{P}$ to P_2



We have two issues

- what if the reduction produces an instance of P_2 having exponential length in the size of P_1 ?
- what if the reduction takes exponential time ?

Polynomial time reductions

We impose the additional constraint that the reduction operates in **polynomial time**, and write $P_1 \leqslant_p P_2$

Theorem If $P_1 \leqslant_p P_2$ and $P_1 \notin \mathcal{P}$ then $P_2 \notin \mathcal{P}$

Proof If $P_2 \in \mathcal{P}$, we would have a polynomial time algorithm for P_1, which is a **contraddiction** $\square$

NP-complete problems

A language L is **NP-complete** if

- $L \in \mathcal{NP}$
- for each language $L' \in \mathcal{NP}$ we have $L' \leqslant_p L$

Example : TSP is NP-complete (to be proved later)

NP-complete problems are the **most difficult** problems among those in $\mathcal{NP}$

If $\mathcal{P} \neq \mathcal{NP}$ then the NP-complete problems are in $\mathcal{NP} \setminus \mathcal{P}$

NP-complete problems

Theorem If P_1 is NP-complete, $P_2 \in \mathcal{NP}$, and $P_1 \leqslant_p P_2$, then P_2 is NP-complete

Proof The polynomial time reduction has the transitive property. For any language $L \in \mathcal{NP}$ we have $L \leqslant_p P_1$ and $P_1 \leqslant_p P_2$, and therefore $L \leqslant_p P_2$ $\square$

NP-complete problems

Theorem If an NP-complete problem is in $\mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$

Proof Assume P is NP-complete and $P \in \mathcal{P}$. For any language $L \in \mathcal{NP}$ we have $L \leqslant_p P$ and therefore we can solve L in polynomial time $\square$

Assuming $\mathcal{P} \neq \mathcal{NP}$, we consider the proof of NP-completeness of a problem P as **evidence** that $P \notin \mathcal{P}$

NP-hard problems

A language L is **NP-hard** if, for each language $L' \in \mathcal{NP}$, we have $L' \leqslant_p L$

Note : We do not require membership in $\mathcal{NP}$. In other words, L could be much more difficult than the problems in $\mathcal{NP}$

Example : Some NP-hard problems take exponential time, even if it turns out that $\mathcal{P} = \mathcal{NP}$

Boolean expressions

We now prove that deciding whether a Boolean expression is satisfiable is an NP-complete problem

As this is our first NP-complete problem, we must **explicitly** reduce each problem in $\mathcal{NP}$ to it

Boolean expressions

Boolean expressions are composed by the following symbols

- an **infinite** set $\{x, y, z, x_1, x_2, \ldots\}$ of variables defined on Boolean values 1 (true) and 0 (false)
- binary operators $\wedge$ (logical AND) and $\vee$ (logical OR)
- unary operator $\neg$ (logical NOT)
- round brackets (to force precedence)

Boolean expressions

A **Boolean expression** E is recursively defined as

- $E = x$, for any Boolean variable x
- $E = E_1 \wedge E_2$ and $E = E_1 \vee E_2$
- $E = \neg E_1$
- $E = (E_1)$

Operator precedence (decreasing) : $\neg, \wedge, \vee$

Example : $x \wedge \neg(y \vee z)$

Satisfiability

A **truth assignment** T for a Boolean expression E assigns a Boolean value $T(x)$ (true or false) to each variable x in E

The Boolean value $E(T)$ of E under T is the result of the evaluation of E with each variable x replaced by $T(x)$.

T **satisfies** E if $E(T) = 1$

E is **satisfiable** if there exists at least one T that satisfies E

Example

$x \wedge \neg(y \vee z)$ is satisfiable : $T(x) = 1,\ T(y) = 0,\ T(z) = 0$

$x \wedge (\neg x \vee y) \wedge \neg y$ cannot be satisfied

- we must have $T(x) = 1$ and $T(y) = 0$
- therefore $(\neg x \vee y)$ must be false

The SAT problem

The **satisfiability problem**, SAT for short, is defined as follows

- the input is a Boolean expression E
- the output is "yes" if E is satisfiable, "no" otherwise

Boolean expression encoding

We rename the variables as $x_1, x_2, \ldots$ and encode them using symbol x followed by a binary representation of the **index**.
Example : $x_{13} = x1101$

Logical operators and parentheses are represented by themselves

We have the **alphabet** $\{\wedge, \vee, \neg, (,), 0, 1, x\}$ (eight symbols) for encoding of Boolean expressions

The SAT language is formed by the set of all Boolean expressions that are well-formed, properly coded, and satisfiable

Example

The Boolean expression

$$x \wedge \neg(y \vee z)$$

is encoded as

$$x1 \wedge \neg(x10 \vee x11)$$

Boolean expression encoding

A Boolean expression E with m occurrences of operators must have $\mathcal{O}(m)$ variable occurrences

A Boolean expression E of size m has an encoding, written $\text{enc}(E)$, of **length** $\mathcal{O}(m \log m)$, which is a polynomial function of m

Cook Theorem

Theorem SAT is an NP-complete language

Proof (First part) SAT $\in \mathcal{NP}$

There is a polynomial NTM that solves SAT

- verify that the input is a well formed Boolean expression
- using **nondeterminism**, guess a truth assignments T; this can be done in polynomial time in the length of $\text{enc}(E)$
- for the guessed T, check if $E(T) = 1$ and accept accordingly; this can be done in polynomial time in the length of $\text{enc}(E)$

Cook Theorem

(Second part) For each $L \in \mathcal{NP}$, $L \leqslant_p$ SAT

The reduction translates an instance w of the problem represented by L into an instance E of SAT, i.e., a string encoding a Boolean expression

Let us set a NTM M and a polynomial $p(n)$ such that $L(M) = L$ and M processes w with $|w| = n$ in at most $p(n)$ steps

In the following M is considered as fixed. The size of Q, Γ and δ is therefore considered as a **constant**

Cook Theorem

We can assume that

- M has semi-infinite tape and never writes B; proof similar to the case of general TM
- on input w with $|w| = n$, M executes exactly $p(n)$ steps on **each** of its computations; proof uses a clock and extends the M moves by with $\alpha \underset{M}{\vdash} \alpha$ for each accepting ID α
- all IDs have length $p(n) + 1$ ($p(n)$ symbols and one state); pad the tail of IDs with symbol B

Cook Theorem

Let $|w| = n$. Each computation of M on w has the form

$$\gamma = \alpha_0 \underset{M}{\vdash} \alpha_1 \underset{M}{\vdash} \cdots \underset{M}{\vdash} \alpha_{p(n)}$$

where

- α_0 is the initial ID on w
- all IDs have the same length
- γ **accepts** if and only if $\alpha_{p(n)}$ is an accepting ID

Each α_i is represented as a sequence

$$X_{i0} X_{i1} \cdots X_{i,p(n)}$$

where exactly one symbol X_{ij} is a state, and all of the others are tape symbols

Cook Theorem

$$\gamma = \alpha_0 \underset{M}{\vdash} \alpha_1 \underset{M}{\vdash} \cdots \underset{M}{\vdash} \alpha_{p(n)}$$

ID	0	1	$\cdots$	$j-1$	j	$j+1$	$\cdots$	$p(n)$
α_0	X_{00}	X_{01}						$X_{0,p(n)}$
α_1	X_{10}	X_{11}						$X_{1,p(n)}$
$\vdots$								
α_i				$X_{i,j-1}$	$X_{i,j}$	$X_{i,j+1}$	$\cdots$	
α_{i+1}				$X_{i+1,j-1}$	$X_{i+1,j}$	$X_{i+1,j+1}$	$\cdots$	
$\vdots$								
$\alpha_{p(n)}$	$X_{p(n),0}$	$X_{p(n),1}$						$X_{p(n),p(n)}$

Cook Theorem

We represent the computation γ using Boolean variables y_{ijZ}, where $0 \leqslant i, j \leqslant p(n)$, $Z \in (\Gamma \cup Q)$, and

$$y_{ijZ} = \begin{cases} 1, & \text{if } X_{ij} = Z \\ 0, & \text{otherwise} \end{cases}$$

The reduction produces a Boolean expression $E_{M,w}$ such that

- $E_{M,w}$ is satisfiable if and only if there exists an accepting computation of M on w
- $E_{M,w}$ can be built in polynomial time in n

Cook Theorem

$$E_{M,w} = U \wedge S \wedge N \wedge F$$

- U (uniqueness) : only one symbol at each cell
- N (next) : adjacent ID's represent a valid move of the TM
- S (start) : γ starts with the initial ID
- F (final) : γ halts with an accepting ID

Cook Theorem

Uniqueness :

$$U = \bigwedge_{\substack{0 \leqslant i,j \leqslant p(n) \\ Z_1, Z_2 \in (\Gamma \cup Q)}} \neg(y_{ijZ_1} \wedge y_{ijZ_2})$$

We have $\mathcal{O}(p(n)^2 \times |\Gamma \cup Q|^2)$ terms. Since we consider $|\Gamma \cup Q|^2$ as a constant, the number of terms is $\mathcal{O}(p(n)^2)$

$|U|$ is a polynomial function of n, where n is the length of the input instance w

Cook Theorem

Start : let $w = a_1 a_2 \cdots a_n$

$$\begin{aligned} S &= y_{00q_0} \wedge y_{01a_1} \wedge y_{02a_2} \wedge \cdots \wedge y_{0na_n} \wedge \\ &\quad y_{0,n+1,B} \wedge y_{0,n+2,B} \wedge \cdots \wedge y_{0,p(n),B} \end{aligned}$$

We have $\mathcal{O}(p(n))$ terms, and $|S|$ is a polynomial function of n

Cook Theorem

Final : let $s_1, s_2, \ldots, s_k$ be all the final states of M

$$F = \bigvee_{\substack{0 \leqslant j \leqslant p(n) \\ 1 \leqslant h \leqslant k}} y_{p(n) j s_h}$$

We have $\mathcal{O}(p(n))$ terms and $|F|$ is a polynomial function of n

Cook Theorem

Next :

$$N = \bigwedge_{0 \leqslant i \leqslant p(n)-1} N_i$$

Each expression N_i guarantees that $\alpha_i \underset{M}{\vdash} \alpha_{i+1}$

Cook Theorem

In order to check the validity of each relation $\alpha_i \underset{M}{\vdash} \alpha_{i+1}$ we always look into windows composed of three tape cells

$X_{i,j-1}$	$X_{i,j}$	$X_{i,j+1}$
$X_{i+1,j-1}$	$X_{i+1,j}$	$X_{i+1,j+1}$

On the basis of $X_{i,j-1}$, $X_{i,j}$, $X_{i,j+1}$ and of the move chosen by M

- it is **always** possible to check the validity of $X_{i+1,j}$
- in **some cases** ($X_{i,j} \in Q$) it is also possible to check the validity of $X_{i+1,j-1}$ and $X_{i+1,j+1}$

Cook Theorem

$$N_i \;=\; \bigwedge_{0 \leqslant j \leqslant p(n)} (A_{ij} \vee B_{ij})$$

The Boolean expression A_{ij} states that

- X_{ij} is the state of α_i
- M can choose any move in $\delta(X_{ij}, X_{i,j+1})$

The Boolean expression B_{ij} states that

- X_{ij} is not a state
- if the state of α_i is not $X_{i,j-1}$ or $X_{i,j+1}$, then $X_{i+1,j} = X_{ij}$

When the state of α_i is $X_{i,j-1}$ or $X_{i,j+1}$, the validity of $X_{i+1,j}$ is guaranteed by $A_{i,j-1}$ or $A_{i,j+1}$

Cook Theorem

Let $q_1, q_2, \ldots, q_m$ be all of the states of M and let $Z_1, Z_2, \ldots, Z_r$ be all of its tape symbols

$$
\begin{aligned}
B_{ij} \quad = \quad & (y_{i,j-1,q_1} \vee y_{i,j-1,q_2} \vee \cdots \vee y_{i,j-1,q_m}) \vee \\
& (y_{i,j+1,q_1} \vee y_{i,j+1,q_2} \vee \cdots \vee y_{i,j+1,q_m}) \vee \\
& ((y_{i,j,Z_1} \vee y_{i,j,Z_2} \vee \cdots \vee y_{i,j,Z_r}) \wedge \\
& ((y_{i,j,Z_1} \wedge y_{i+1,j,Z_1}) \vee (y_{i,j,Z_2} \wedge y_{i+1,j,Z_2}) \vee \cdots \vee \\
& \ (y_{i,j,Z_r} \wedge y_{i+1,j,Z_r})))
\end{aligned}
$$

- if the state of α_i is adjacent to X_{ij} we do not impose any condition
- if the state of α_i is X_{ij}, B_{ij} is false so that A_{ij} must be true
- if the state of α_i is not $X_{i,j-1}$ or $X_{i,j+1}$, then $X_{i+1,j} = X_{ij}$

Cook Theorem

Let us assume that $(p, C, L) \in \delta(q, A)$ and $D \in \Gamma$. Then

$$X_{i,j-1}X_{i,j}X_{i,j+1} = DqA$$
$$X_{i+1,j-1}X_{i+1,j}X_{i+1,j+1} = pDC$$

is a valid assignment for the logical variables in a 2×3 rectagle in the table representing a computation

We can represent the assignment by means of the term

$$y_{i,j-1,D} \wedge y_{i,j,q} \wedge y_{i,j+1,A} \wedge y_{i+1,j-1,p} \wedge y_{i+1,j,D} \wedge y_{i+1,j+1,C}$$

Cook Theorem

Let us assume that $(p, C, R) \in \delta(q, A)$. Then

$$X_{i,j-1}X_{i,j}X_{i,j+1} = DqA$$
$$X_{i+1,j-1}X_{i+1,j}X_{i+1,j+1} = DCp$$

is a valid assignment

The assignment is represent by means of the term

$$y_{i,j-1,D} \wedge y_{i,j,q} \wedge y_{i,j+1,A} \wedge y_{i+1,j-1,D} \wedge y_{i+1,j,C} \wedge y_{i+1,j+1,p}$$

Cook Theorem

An assignment for a 2×3 rectangle is **valid** if

- $X_{i,j} \in Q$ and $X_{i,j-1}, X_{i,j+1} \in \Gamma$
- there is a move by M that changes the values of $X_{i,j-1}X_{i,j}X_{i,j+1}$ into the values of $X_{i+1,j-1}X_{i+1,j}X_{i+1,j+1}$

The number of valid assignments depends on the size of Q and Γ and on the moves in δ. Since M is fixed, the number of valid assignments is a constant

A_{ij} is the logical OR of all terms representing valid assignments

Cook Theorem

Summarizing :

$$\begin{aligned} N &= \bigwedge_{0 \leqslant i \leqslant p(n)-1} N_i \\ N_i &= \bigwedge_{0 \leqslant j \leqslant p(n)} (A_{ij} \vee B_{ij}) \end{aligned}$$

$|A_{ij}|, |B_{ij}|$ are costants (M is fixed)
$|N_i|, |N|$ are polynomial functions of n

To conclude, $E_{M,w}$ is satisfiable if and only if $w \in L(M)$, and $|E_{M,w}|$ is a polynomial function of n $\square$

Normal forms for Boolean expressions

Boolean expressions have a fairly complex structure

We introduce a **simplified** version of SAT, called 3SAT

- 3SAT is an NP-complete problem
- 3SAT is particularly convenient to define reductions that we will investigate later

Normal forms for Boolean expressions

A **literal** is a variable or else the negation of a variable.
Example : x; $\overline{x} = \neg x$

A **clause** is the disjunction (logical OR) of literals.
Example : $x \vee \overline{y} \vee z$

A Boolean expression in **conjunctive normal form**, or CNF for short, is a conjunction (logical AND) of clauses.
Example : $(x \vee \overline{y}) \wedge (\overline{x} \vee z)$

Normal forms for Boolean expressions

We use $+$ in place of $\vee$ and we use $\times$ in place of $\wedge$. As for arithmetic expressions, we represent $\times$ by means of concatenation

Example :

- $(x \vee \overline{y}) \wedge (\overline{x} \vee z)$ is written as $(x + \overline{y})(\overline{x} + z)$
- $(x + y\overline{z})(\overline{x} + y + z)$ is not in CNF
- xyz is in CNF

Normal forms for Boolean expressions

A Boolean expression is in ***k*-conjunctive normal form**, or k-CNF for short, if

- it is in CNF
- every clause has **exactly** k literals

Example : $(x + \overline{y})(\overline{x} + z)$ is in 2-CNF

We introduce two new decision problems

- CSAT : is some CNF satisfiable ?
- kSAT : is some k-CNF satisfiable ?

Results

Theorem CSAT is NP-complete

Proof CSAT $\in \mathcal{NP}$; SAT $\leqslant_p$ CSAT

Theorem 3SAT is NP-complete

Proof 3SAT $\in \mathcal{NP}$; CSAT $\leqslant_p$ 3SAT

NP-completeness

Finding out that a decision problem is NP-complete indicates that there are **very few chances** to discover an efficient algorithm for its solution. It is therefore recommended to look for partial / approximate solutions, using heuristics

The large number of failed attempts to prove $\mathcal{P} = \mathcal{NP}$ provides evidence that every NP-complete problem requires **exponential time** for an exact solution

NP-completeness

Many collections of NP-complete problems have been published and are constantly updated

Typically, these decision problems are described according to the following scheme

- problem name and abbreviation
- problem input and its representation / encoding
- specification of positive instances of the problem
- problem used in the reduction for the NP-completeness result

Example

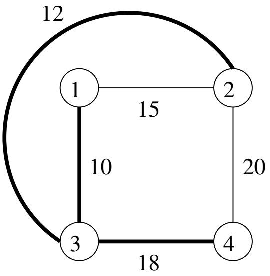
PROBLEM : satisfiability of Boolean expressions in 3-CNF (3SAT)
INPUT : Boolean expressions in 3-CNF
OUTPUT : "yes" if and only if the Boolean expressions is satisfiable
REDUCTION : from CSAT

Independent set

In a graph G, a subset I of the nodes is an **independent set** if no pair of nodes in I is connected by some arc of G

An independent set is **maximal** if any other independent set of G has a number of nodes smaller or equal than the former

Example



$I = \{1, 4\}$ is an independent set

I is maximal : any set of three nodes from the graph has some arc connection

Independent set

The problem of finding a maximal independent set is investigated in the area of **combinatorial optimization**

We consider here the **decision** version of this problem

Independent set

PROBLEM : independent set (IS)
INPUT : undirected graph G and lower bound k
OUTPUT : "yes" if and only if G has an independent set with k nodes
REDUCTION : from 3SAT

For small values of k, it can be easy to solve the problem. But if k is the size of the maximal independent set, then the solution of the problem is **difficult**

IS is NP-complete

Theorem IS is NP-complete

Proof (First part) IS $\in \mathcal{NP}$

Let us consider a NTM that

- arbitrarily chooses k nodes using **nondeterminism**
- verifies that the chosen set is independent, and accepts accordingly

The two phases described above can be performed in polynomial time in the size of the input data

IS is NP-complete

(Second part) 3SAT $\leqslant_p$ IS

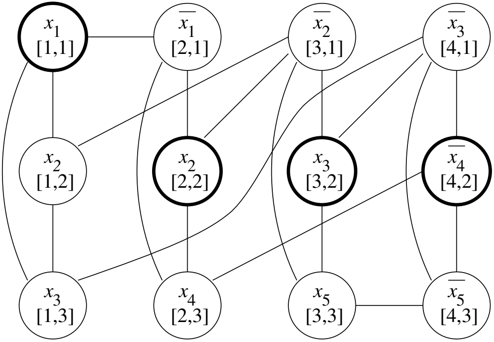
Let $E = e_1 \wedge e_2 \wedge \cdots \wedge e_m$ be a Boolean expression in 3-CNF, where each e_i is a clause

We build G with $3m$ nodes. Each node is identified by a pair $[i, j]$, with $1 \leqslant i \leqslant m$ and $j \in \{1, 2, 3\}$

The $[i, j]$ node represents the j-th literal in the i-th clause

Example

$$E = (x_1 + x_2 + x_3)(\overline{x_1} + x_2 + x_4)(\overline{x_2} + x_3 + x_5)(\overline{x_3} + \overline{x_4} + \overline{x_5})$$



IS is NP-complete

Construction of graph G

- one arc for each pair of nodes in the same column;
  then one can choose no more than one node per clause
- one arc for each pair of nodes $[i_1, j_1]$, $[i_2, j_2]$, if these represent the literals x and $\overline{x}$;
  then one cannot choose two literals in an independent set if they are one the negation of the other

We let $k = m$

IS is NP-complete

We can build G and k in **polynomial time** (quadratic) in the length of the representation of E

We prove that E is satisfiable if and only if G has an independent set with m elements

IS is NP-complete

(If part) Let I be an independent set with m elements. We define

- $T(x) = 1$ if the node for x is in I
- $T(x) = 0$ if the node for $\overline{x}$ is in I
- $T(x)$ can be arbitrarily defined if the nodes for x and $\overline{x}$ are not in I

Since nodes for x and $\overline{x}$ cannot **simultaneously** belong to I, the definition of T is consistent

An independent set I contains exactly one node per clause. It follows that the definition of T **satisfies** E

IS is NP-complete

(Only if part) Let T be an assignment that satisfies E. We arbitrarily choose a true literal for each clause, and we add to I the node associated with that literal

I has size m

I is an independent set

- if one arc connects two nodes from the same clause, the two nodes are not both in I by construction
- if the remaining arcs connect two nodes corresponding to a literal and its negation, then the two nodes are not both in I because we have chosen only literals that are true in T $\square$

Node cover

In a graph G, a subset C of the nodes is a **node cover** if each arc of G impinges upon **at least one** node in C

A node cover is **minimal** if its size is smaller or equal than the size of any other node cover of G

Node cover

PROBLEM : node cover (NC)
INPUT : undirected graph G and upper bound k
OUTPUT : "yes" if and only if G has a node cover with at most k nodes
REDUCTION : from IS

NC is NP-complete

Theorem NC is NP-complete

Proof (First part) NC $\in \mathcal{NP}$

Let us consider a NTM that

- arbitrarily chooses k nodes of the input graph G, using **nondeterminism**
- tests whether the chosen set is a node cover, and accepts accordingly

Both the above steps can be carried out in time polynomial in the size of the input

NC is NP-complete

(Second part) IS $\leqslant_p$ NC

Let G, k be an instance of IS, and let n be the number of nodes of G. We produce an instance of NC formed by G and $n - k$

Construction takes **polynomial time**

We prove that G has an independent set with k elements if and only if G has a node cover with $n - k$ elements

NC is NP-complete

(If part) Let N be the set of nodes of G and let C be a node cover with $n - k$ nodes. We argue that $N \smallsetminus C$ with k nodes is an independent set for G

Let us assume that $N \smallsetminus C$ is not independent. Then there are nodes $v, w \in (N \smallsetminus C)$ that are connected by some arc

Thus $v, w \notin C$ and then the arc (v, w) is uncovered. We have a **contraddiction** since C is a node cover

NC is NP-complete

(Only if part) Let I be an independent set with k nodes. We argue that $N \setminus I$ is a node cover with $n - k$ nodes

Let us assume that an arc (v, w) is not covered by $N \setminus I$. Then v, w are in I

Since (v, w) is an arc, we have a **contraddiction** because I is an independent set $\square$

Directed Hamiltonian circuit

Let G be an oriented graph. A **directed Hamiltonian circuit** in G is an oriented cycle that passes through each node of G **exactly once**

PROBLEM : directed Hamiltonian circuit (DHC)
INPUT : directed graph G
OUTPUT : "yes" if and only if G has a directed Hamiltonian circuit
REDUCTION : from 3SAT

Undirected Hamiltonian circuit

PROBLEM : undirected Hamiltonian circuit (HC)
INPUT : undirected graph G
OUTPUT : "yes" if and only if G has an undirected Hamiltonian circuit
REDUCTION : from DHC

Traveling salesman problem

PROBLEM : traveling salesman problem (TSP)
INPUT : undirected graph G with integer weights at every arc, and upper bound k
OUTPUT : "yes" if and only if G has an undirected Hamiltonian circuit such that the sum of the weights at each arc is smaller equal than k
REDUCTION : from HC

Summary of our reductions