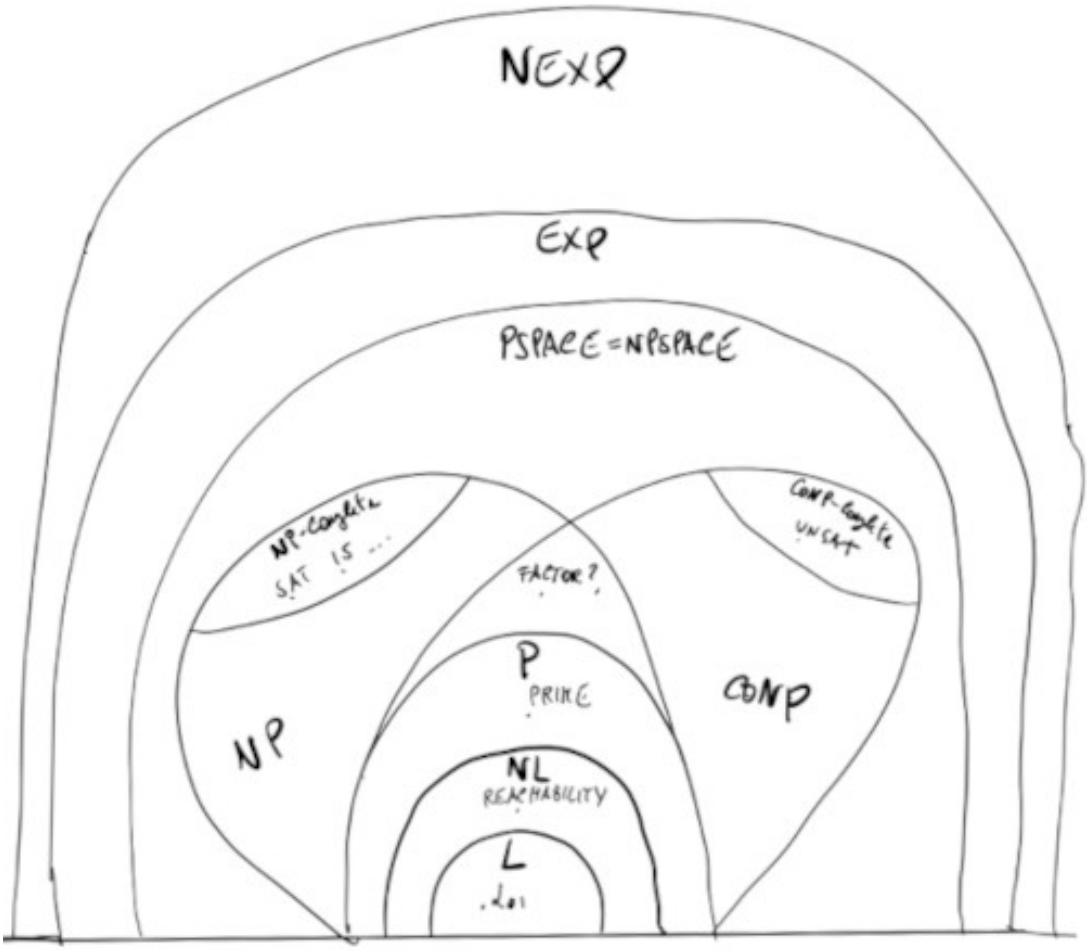
Computational Complexity

June 27, 2021

Contents

1 Overview



2 DTIME and NTIME, P and NP, EXP and NEXP

From now on we will focus our attention on the class of decidable problems $R = \{L|\exists$ Turing machine M that decides $L\}$. Considering the languages that can be decided in $O(f(n)), f : \mathbb{N} \to \mathbb{N}$, we have

- $\text{DTIME}(f(n)) = \{L|L \in R, T_M(n) \in O(f(n))\}$, solvable by a deterministic TM.
- $\text{NTIME}(f(n)) = \{L|L \in R, NT_M(n) \in O(f(n))\}$, solvable by a nondeterministic TM.

Considering only the languages that can be decided in polynomial time, we have the following complexity classes

- $\mathbf{P} = \bigcup_{c \geq 1} \text{DTIME}(n^c)$, problems solvable by a deterministic TM in polynomial time.
- $\mathbf{NP} = \bigcup_{c \geq 1} \text{NTIME}(n^c)$, problems solvable by a nondeterministic TM in polynomial time.

Nondeterministic TMs are often devised using a *trail and error* (or guess and check) approach, where the resulting computation takes the form of a decision tree model. To check if $w \in L$, the NTM should

1. Guess a *certificate* (some data) on the tape, in polynomial time.
2. Check the certificate to proove $w \in L$, in polynomial time.

The class **PC** collects the languages that can be decided with such NTMs, and it is in fact an alternative definition of **NP** since **PC** = **NP**.

We can also define two more classes of problems that deal with higher time resource usage.

- **EXP** $= \bigcup_{c\geq 1} \text{DTIME}(2n^c)$, solvable by a deterministic Turing machine in exponential time.
- **NEXP** $= \bigcup_{c\geq 1} \text{NTIME}(2n^c)$, solvable by a non-deterministic Turing machine in exponential time.

3 P languages

3.1 Reachability

$$\text{REACHABILITY} = \{(G, s, t) | \exists \text{ a path from node } s \text{ to node } t \text{ in the directed graph } G\}$$

Data: Directed graph $G = (V, E)$ and nodes s, t
Result: Y if there is a path from s to t, N otherwise
`// Done in constant time w.r.t. input size`
init an empty queue Q;
mark node s as *visited*;
append s to Q;
`// At most` $|V|$ `iterations, visiting all the nodes in` G
while Q *is not empty* **do**
  extract node v, the first element of Q;
  if v *is* t **then**
    return Y;
  end
  `// At most` $|V|$ `iterations, all the the nodes are neighbours of` v
  forall $(v, u) \in E$ *s.t.* u *is not visited* **do**
    mark node u as *visited*;
    append u to Q;
  end
end
return N;

Algorithm 1: Breadth-first search, in $O(n^2)$ with $n = |V|$

3.2 Prime

The algorithm below is not polynomial in the *size of the input*: since the input has been encoded in binary, we use $n = \lceil \log_2 N \rceil$ bits to encode N. This means that $N \in O(2^n)$ and the algorithm is exponential w.r.t to the input size. In the early 2000s a far more complex algorithm has been designed to prove that PRIME $\in$ **P**.

$$\text{PRIME} = \{< N > | N \text{ is prime}\}, \quad < N > \text{ binary encoding of } N \in \mathbb{N}$$

Data: $< N >$
Result: Y if N is prime, N otherwise
`// At most` N `iterations`
forall $k = 2$ *to* $N - 1$ **do**
  if k *divides* N **then**
    return N;
  end
end
return Y;

Algorithm 2: Naive iteration, in $O(N)$

4 NP – complete languages

A language L is said to be **NP** – **complete** if and only by providing a polynomial-time algorithm to decide L one proves that every language in **NP** can be decided in polynomial time. For the study of **NP** – **completeness** we need to introduce some useful concepts.

- Language L_1 reduces to language L_2 in polynomial time ($L_1 \leq_p L_2$) if there is a *polynomial-time reduction* M such that $T_M(n) \in O(n^c),\ c > 0$.
- A language L is $\mathbf{NP-hard}$ if $L' \leq_p L\ \forall L' \in \mathbf{NP}$.
- $L \in \mathbf{NP-complete} \Rightarrow L \in \mathbf{NP} \wedge L \in \mathbf{NP-hard}$.
- $L \in \mathbf{NP-complete} \wedge L \in \mathbf{P} \iff \mathbf{P} = \mathbf{NP}$. In other words, deciding an $\mathbf{NP-complete}$ problem in polynomial time proves that $\mathbf{P} = \mathbf{NP}$.
- $L_1 \in \mathbf{NP-hard} \wedge L_1 \leq_p L_2 \Rightarrow L_2 \in \mathbf{NP-hard}$.
- Cook's theorem states that SAT $\in \mathbf{NP-complete}$.

So to prove a language L is $\mathbf{NP-complete}$ it is enough to show that

1. $L \in \mathbf{NP}$.
2. $L \leq_p L'$, where $L' \in \mathbf{NP-hard}$.

4.1 SAT

A CNF (*Conjunctive Normal Form*) is a boolean formula of the form $C_1 \wedge C_2 \wedge \cdots \wedge C_n$, with clauses $C_i = (l_1 \vee l_2 \vee \cdots \vee l_k)$ and literals l_j. A CNF is satisfiable if there is a truth assignment τ that makes φ true.

Example: For $\varphi = (x_1 \vee x_2 \vee x_2 \vee \bar{x}_3) \wedge (x_4 \vee \bar{x}_5) \wedge (x_1 \vee \bar{x}_5)$ a truth assignment that satisfies φ is $\tau : x_1 \to 1, x_2, x_3, x_4, x_5 \to 0$.

Consider the decision problem SAT $= \{\varphi | \varphi$ is a satisfiable CNF$\}$. Given a CNF φ and a truth assignment τ, we can provide an algorithm that checks if τ satisfies φ in polynomial time.

Data: A CNF boolean formula φ and a truth assignment τ
Result: Y if τ satisfies φ N otherwise
`// At most` m `iterations, checking all the clauses`
forall C_i *of* φ **do**
  $cvalue = False$;
  `// At most` c `iterations, checking all the literals`
  forall l_j *of* C_i **do**
    if $l_j = True$ **then**
      $cvalue = True$;
      break;
    end
  end
  if *cvalue = False* **then**
    return N;
  end
end
return Y;

Algorithm 3: Naive iterations for τ, in $O(m \cdot n)$ with m clauses and c maximum number of literals in a clause

However, a CNF with n boolean variables has a total of 2^n possible truth assignment, making the overall procedure exponential in time. No polynomial procedure has been devised to date.

Can we get a better result by relying on non-determinism? We can use a *trial and error* approach with a NTM that, given a CNF φ with variables $x_1, \cdots, x_n$

1. Makes a non-deterministic guess τ for $x_1, \cdots, x_n$, written on the tape using n bits (the i-th bit represents the assignment for x_i). Each guess takes a different path on the computational tree and writes polynomially many symbols in the tape.
2. Verifies in polynomial time that τ satisfies φ, if so accept (otherwise reject).

By providing a NTM that decides SAT, we proved that SAT $\in \mathbf{NP}$.

As already said, Cook's theorem states that $SAT \in \mathbf{NP-complete}$, but even simplified versions are still in $\mathbf{NP-complete}$.

- 3-SAT = $\{\varphi|\varphi$ is a boolean formula in 3-CNF that is satisfiable$\}$, where 3-CNF is a CNF whose clauses have at most three literals.
- EXACT-3-SAT = $\{\varphi|\varphi$ is a boolean formula in EXACT-3-CNF that is satisfiable$\}$, where EXACT-3-CNF is a CNF whose clauses have exactly three literals.

4.2 Independent set

Given an undirected graph $G = (V, E)$, S is an *independent set* in G if and only if there is no edge connecting any pair of nodes in S. We are interested in the decision problem IS = $\{(G, K)|G$ undirected graph, $\exists S$ independent set $k\}$ is also **NP-complete**. While we can provide an algorithm that checks if S is an independent set of G in polynomial time, the total number of possible sets of nodes for a graph G is $\sum_{i=k}^{n} \binom{n}{i}$. Our procedure is not polynomial in time and no polynomial procedure has been devised to date.

Data: Undirected graph G, cardinality k
Result: Y if G contains an indipendent set S s.t. $|S| \geq k$, N otherwise
`// At most` $\sum_{i=k}^{n} \binom{n}{i}$ `iterations, checking all the possible sets with at least` k `nodes`
forall *Set of nodes* S *s.t.* $|S| \geq k$ **do**
  `// At most` $n(n-1)/2$ `iterations, checking all the pairs when` $|S| = n$
  forall $(u, v) \in S$ **do**
    if (u, v) *are connected* **then**
      break;
    end
  end
  return Y;
end
return N;

Algorithm 4: Naive iterations, in $O(n^2 \sum_{i=k}^{n} \frac{n!}{i!(n-i)!})$ with n nodes

Similarly to SAT, we can also devise a NTM that uses a trial and error strategy. Given an undirected graph G with n nodes and a number k

1. For each node v in G, non-deterministically decide to write or not v in the tape.
2. Accept if at least k nodes have been written and no two of them are connected (otherwise reject).

By providing a NTM that decides IS, we proved that IS $\in$ **NP**. To proove that IS $\in$ **NP** $-$ **hard** (and **NP** $-$ **complete**) we can provide a polynomial-time reduction for EXACT-3-SAT $\leq_p$ IS.

4.3 Vertex cover

Given an undirected graph $G = (V, E)$, $VC \subseteq V$ is a *vertex cover* if and only if all the vertices of G are touched by some node in VC. We set the decision problem VCOVER = $\{(G, k)|G$ undirected graph $, \exists VC$ vertex cover s.t $k\}$ and we want to prove VCOVER $\in$ **NP** $-$ **complete**.

For this problem we can rely on one important observation: let $G = (V, E)$ be an undirected graph and $S \subseteq V$ a set of nodes, then S is an independent set in G if and only if $V \setminus S$ is a vertex cover of G.

We first proove that VCOVER $\in$ **NP** by devising a NTM that

- Makes a non-deterministic guess of at most k nodes in its tape in polynomial time.
- Verifies in polynomial time that all edges are covered.

To proove that VCOVER $\in$ **NP** $-$ **hard** we can define a polynomial reduction for IS $\leq_p$ VCOVER. We must devise a reduction that converts a pair (G, k) to a pair (G', k'): if G has an independent set with at least k nodes, then G' has a vertex cover with at most k' nodes (and vice versa). So the reduction constructs $G' = G$ and $k' = |V| - k$ in a way that

- If G has an independent set S with $|S| \geq k$, then (for the previous observation) $V \setminus S$ is a vertex cover of G (and thus of G'). And since $|S| \geq k$, then $|VC| = |V \setminus S| \leq |V| - k = k'$.
- If G' has a vertex cover VC with $|VC| < k'$, then $S = V \setminus VC$ is an independent set of G' (and thus of G). And since $|VC| < k'$, then $|S| = |v \setminus VC| \geq |V| - k' = k$.

Then we have also proven that VCOVER $\in$ **NP** $-$ **complete**.

4.4 Clique

Given an undirected graph $G = (V, E)$, $C \in V$ is a *clique* if and only if the nodes in C form a fully connected subgraph in G. We consider the decision problem CLIQUE $= \{(G, k) | G$ undirected graph $, \exists C$ clique s.t. $|VC| \geq k\}$.

For this problem we can exploit the fact that an independent set S in G corresponds to a clique in a graph $\overline{G}$, with $\overline{G}$ being the complement of G (any pair of nodes G is adjacent iff it's not adjacent in $\overline{G}$).

We first proove that CLIQUE $\in$ **NP** by devising a NTM that

- Makes a non-deterministic guess of at lest k nodes in its tape in polynomial time.
- Verifies in polynomial time that all pairs are connected.

To proove that $CLIQUE \in \mathbf{NP} - \mathbf{hard}$ we can define a polynomial reduction for $IS \leq_p CLIQUE$. We must devise a reduction that converts a pair (G, k) to a pair (G', k'): if G has an independent set with at least k nodes, then G' has a clique with at least k' nodes (and vice versa). So the reduction constructs $G' = \overline{G}$, by copying $|V|$ nodes and adding at most $|V|^2$, and $k' = k$ in a way that

- If G has an independent set S with $|S| \geq k$, then (for the previous observation) S is a clique of G' with at least $k = k'$ nodes in G'.
- If G' has a clique C with at least k' nodes, then no distinct nodes in G are connected in G' and C is an independent set with at least $k' = k$ nodes in G.

4.5 Binary integer programming

We consider now an optimization problem. Consider a system of inequalities of the form

$$\begin{cases} a_{11} \cdot x_1 + a_{12} \cdot x_2 + \cdots a_{1n} \cdot x_n \leq b_1 \\ a_{21} \cdot x_1 + a_{22} \cdot x_2 + \cdots a_{2n} \cdot x_n \leq b_2 \\ \cdots \\ a_{m1} \cdot x_1 + a_{m2} \cdot x_2 + \cdots a_{mn} \cdot x_n \leq b_m \end{cases}$$

Does it have a solution with $x_1, \cdots, x_n \in \{0, 1\}$? In other words, given a matrix $A \in \mathbb{Z}^{m \times n}$ and a vector $\overline{b} \in Z^m$ is there are vector $\overline{x} \in \{0, 1\}^n$ such that $A \cdot \overline{x} \leq \overline{b}$? In this setting we define the problem BIP (Binary Integer Programming)

$$\text{BIP} = \{(A, \overline{b}) | A \in \mathbb{Z}^{m \times n},\ \overline{b} \in Z^m,\ \exists \overline{x} \in \{0, 1\}^n s.t. A \cdot \overline{x} \leq \overline{b}\}$$

We first prove that BIP $\in NP$ by devising a NTM that

- Makes a nondeterministic guess of $0, 1$ values for each variable in $\overline{x}$, each guess takes n steps (done in polynomial time).
- Checks that all m inequalities are satisfied in polynomial time.

To proove BIP $\in$ **NP-hard** we provide a reduction for EXACT-3-SAT $\leq_p$ BIP. Such a reduction should convert, in polynomial time, an EXACT-3-CNF to a system of inequalities (and viceversa) in a way that the CNF is satisfiable iff the system admits solutions. We procede with an example to explain the logic.

Consider a EXACT-3-CNF $\varphi = (x_1 \vee x_2 \vee \bar{x}_3)(\bar{x}_2 \vee x_3 \vee \bar{x}_4)$.

- From the boolean variables x_i of φ we can set corresponding arithmetic variables y_i for the system, such that if $x_i = True \rightarrow y_i = 1$, $x_i = False \rightarrow y_i = 0$. Then we will have the arithmetic variables y_1, y_2, y_3, y_4.
- φ is satisfiable iff each clause is true, so at least one literal in each clause must be true. We can construct for each clause an inequality such that if its literal is of the form $x_i \rightarrow y_i$, $\bar{x}_i \rightarrow (1 - y_i)$, and the resulting sum should be at least 1. The result is the following system of inequalities

$$\begin{cases} y_1 + y_2 + (1 - y_3) \geq 1 \\ (1 - y_2) + y_3 + y_4 \geq 1 \end{cases}$$

The instances of BIP take the form of $\leq$ inequalities with one constant on the right, but it is a simple matter of arithmetics to set the new system in such a form. The system is constructed in polynomial time: m inequalities are constructed with at most $n + 1$ coefficients.

Such a procedure is indeed a reduction that works in both directions.

- If φ is satisfiable then there is a truth assignment τ that makes φ true: each clause of τ has at least one true literal, so at least one of the expressions of the form y_i or $(1 - y_i)$ must be one and the inequality is satisfied. This holds for every inequality in the system.
- If our system has a solution, then there is an assignment for each y_i that satisfies all the inequalities. In such an assignment, at least one of the expressions of the form y_i or $(1 - y_i)$, and so its corresponding literal in its clause. This holds for all the clauses of τ.

Also the more general version of the problem IP $= \{(A, \bar{b}) | A \in \mathbb{Z}^{m \times n}, \bar{b} \in \mathbb{Z}^m\} \in$ **NP-complete**.

4.6 Vertex coloring

Given an undirected graph $G = (V, E)$ and $k \in \mathbb{Z}$, a *k-coloring function* $f : V \to \{1, 2, \ldots, k\}$ assigns a color to each node so that $f(u) \neq f(v), \forall \{u, v\} \in E$: the result would be a graph where no pair of nodes with the same color are touching. We are interested in solving the following problem

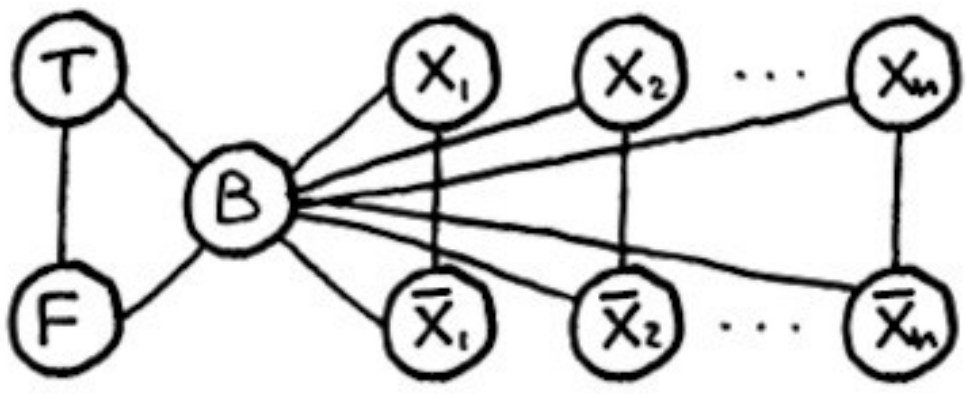
$$\text{VCOL} = \{(G, k) | G \text{ is an undirected graph admitting a k-coloring}\}$$

First, VCOL $\in NP$ since we can devise a NTM that

- Guesses a coloring for each node. If the NTM encodes k colors using $||k||$ bits, then for n nodes the guess takes $O(||k|| \cdot n)$.
- Checks that the guess provides a k-coloring in $O(|E|)$.

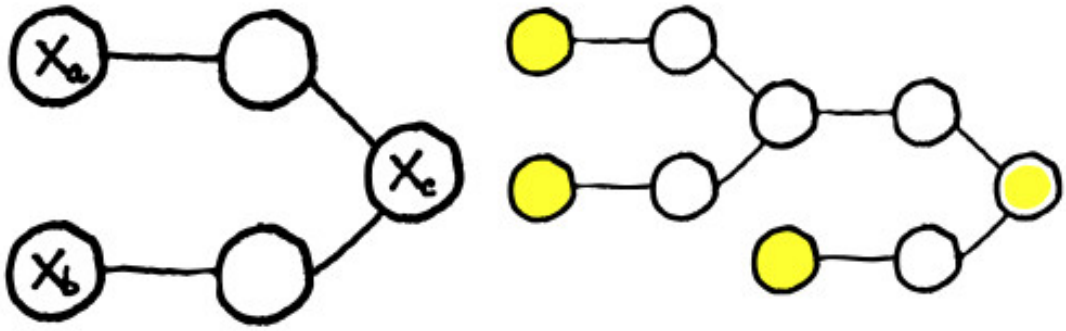
To proove VCOL $\in$ **NP-hard** we provide a reduction for EXACT-3-SAT $\leq_p$ VCOL. Such a reduction should convert, in polynomial time, an EXACT-3-CNF to a pair (G, k) (and viceversa) in a way that the CNF is satisfiable iff G admits a k coloring. We procede with an example to explain the logic.

We first construct a graph G with three nodes: T (with color *true*), F (with color $false$) and B (with color *base*). For each variable x_i of φ, we add two nodes and connect them to B: one labeled x_i, the other labeled $\bar{x}_i$. The color x_i and x_i must be either *true* or $false$, and since they can't have the same color they are also connected.

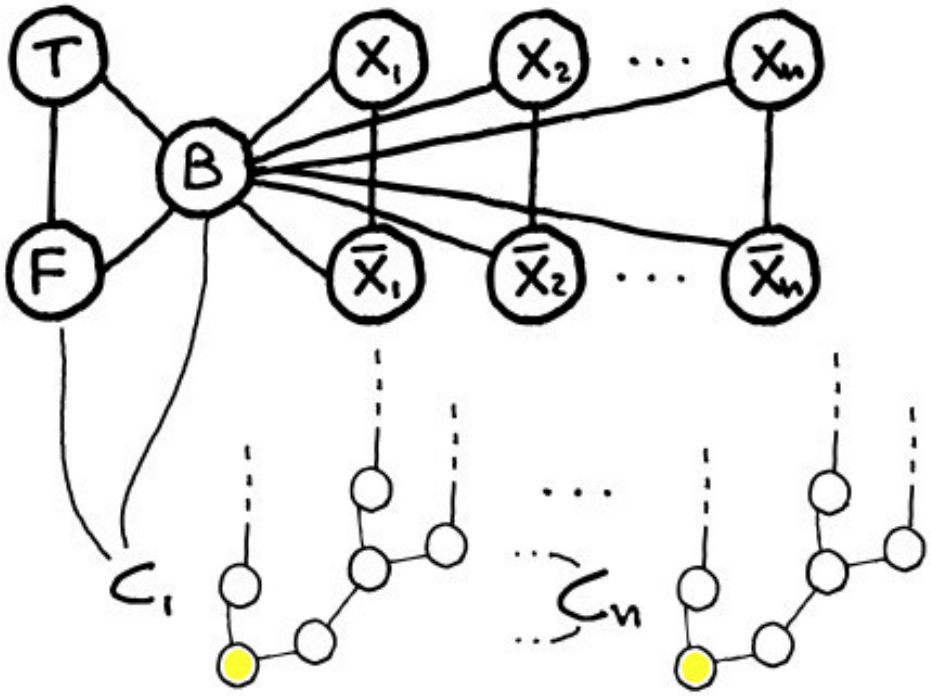


We need to model the fact that some literals make certain clauses true, and to do so we introduce the OR operation through a triangle scheme. The scheme receives as input two literals x_a and x_b and gives the result in the node x_c

- If both x_a and x_b are colored $false$, then the intermediate nodes must be $true$ and $base$. So x_c must be $false$.
- If either x_a or x_b is colored $true$, then there is a coloring for the intermediate nodes with one $false$ and the other $base$. So x_c must be true.



Such a construction can be combined to compute the value of a clause of three literals. To force each clause to be $true$, we just need to connect its output node to F and B.



The described procedure is a reduction in both directions.

- If φ is satisfiable then there is a truth assignment τ that makes φ true. For every assignment $\tau(x_i)$, nodes x_i and $\bar{x}_i$ are colored accordingly to the assigned value and respect the 3-coloring. Finally, since each clause in φ must be true, the output of a 3-input OR can be colored in such a way that the output note is colored $true$.
- If G has a 3-coloring then nodes x_i and $\bar{x}_i$ have different colors (either $true$ or $false$). Also, for each clause the output of the circuit must be $true$ since it is connected to F and B. Finally, the clause output is $true$ only if one of its input nodes is $true$ (and vice versa, if its input are only $false$ then the output is $false$, which is impossible).

The reduction that we just described uses only 3 colors, in fact also 3-VCOL $\in$ **NP − complete**.

$$\text{3-VCOL} = \{G | G \text{ is an undirected graph admitting a 3-coloring}\}$$

5 coNP and FACTOR

We consider now the problem UNSAT = $\{\varphi | \varphi$ is a non-satisfiable boolean formula in CNF$\}$. By trying to prove that UNSAT we might be tempted to follow the method used for SAT and devise a NTM that

- Guesses a truth assignment τ for φ in polynomial time.
- Checks that τ *does not* satisfy φ, in which case accepts (or rejects).

However, the NTM we just provide doesn't decide UNSAT: the CNF φ might be

Not satisfiable In which case any branch of the computational tree leads to an accepting state, and the NTM correctly accepts φ.

Satisfiable In which case some branches still lead to a τ that doesn't satisfy φ, and the NTM wrongly accepts φ.

In fact, the devised NTM guesses a certificate to answer "No" rather than "Yes": while acceptance requires the *exististence* of a path to accept, rejection requires *all* paths to be rejecting. This asymmety between acceptance and rejection is somehow similar to the one in RE languages and their complements.

After this introduction, we can define the class of languages **coNP** $= \{L | \bar{L} \in \mathbf{NP}\}$ with an important observation: **coNP** is *not* the complement of **NP**, is the set of *complements* of **NP** languages. In fact, there are languages L such that $L \in \mathbf{NP} \wedge L \in \mathbf{coNP}$.

Another open question is if **NP** = **coNP**, and we can have some insights on this claim knowing that

- $\mathbf{NP} = \mathbf{coNP} \Leftrightarrow \exists L \in \mathbf{NP} - \mathbf{complete} \wedge L \in \mathbf{coNP}$, no such language has been found yet.
- $\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}$, this and previous statement imply that if **P** = **NP** then **NP** = **coNP** (not the opposite).

An example of language both in **NP** and **coNP** is FACTOR. The common belief is that FACTOR $\notin$ **P** (and FACTOR $\notin$ **NP-complete**).

$$\text{FACTOR} = \{(N, k) | N \in \mathbb{N} \wedge M > k, \forall M \text{prime factor of } N\}$$

6 DSPACE and NSPACE, LOGSPACE and NL

We are now consider the *space* requirement of TMs. For convenience, we are going to consider only TMs (or NTMs) with just a read-only *input tape* and a read-write *working tape*. For such a TM we define $S_M(n)$ as the maximum number of cells visited on the working tap when executed with input of lenght n (for NTM whole the max among all computation branches). Considering the languages that can be decided in $O(f(n)), f : \mathbb{N} \to \mathbb{N}$, we have

- DSPACE$(f(n)) = \{L | \exists T_M$ deciding $L, S_M(n) \in O(f(n))\}$, solvable by a deterministic TM using space in the order of $f(n)$.
- NSPACE$(f(n)) = \{L | \exists NT_M$ deciding $L, S_M(n) \in O(f(n))\}$, solvable by a non-deterministic TM sing space in the order of $f(n)$.

Considering the languages that can be decided with a logarithmic amount of space, we have

- **LOGSPACE** = DSPACE$(\log n)$, solvable by a deterministic Turing machine
- **NL** = NSPACE$(\log n)$, solvable by a non-deterministic Turing machine

For sure **LOGSPACE** $\subseteq$ **NL**, but is **LOGSPACE** = **NL**? This is still an open question

- A language $L \in$ **NL-complete** $\iff L \in \mathbf{NL} \wedge L' \leq_L L, \forall L' \in \mathbf{NL}$, where $\leq_L$ stands for logspace reduction.
- We know that **LOGSPACE** = **NL** $\iff \exists L | L \in \mathbf{LOGSPACE} \wedge L \in$ **NL-complete**.

For example $L_{01} = \{0^n 1^n | n \geq 0\} \in \textbf{LOGSPACE}$. We can devise a TM that takes an input w and accepts if w is empty, otherwise scans left-to-right and increments a counter for each 0 found. When a 1, the TM keeps scanning but increments a second counter for each 1 found: if at the two counters are the same, accept, otherwise reject. For an input of lenght $n = |w|$ each counter, encoded in binary, takes $O(\log_2 n)$ cells, so $S_M(n) \in O(\log_2 n)$.

Data: Input string w
Result: Y if $w = 0^n 1^n$, N otherwise
if w={} **then**
  return Y;
end
zero_counter $= 0$;
while *input_cell is 0* **do**
  zero_counter $=$ *zero_counter* $+ 1$;
  read next *input_cell*;
end
one_counter $= 0$;
while *input_cell is 1* **do**
  one_counter $=$ *one_counter* $+ 1$;
  read next *input_cell*;
end
if *input_cell* $=$ _ *and zero_counter* $=$ *one_counter* **then**
  return Y;
else
  return N;
end
// Only two counters are used, each at most $n = |w|$, encoded in binary so $O(\log_2 n)$

Algorithm 5: Turing machine for L_{01}, $S_M(n) \in O(\log_2 n) \Rightarrow L_{01} \in \textbf{LOGSPACE}$

An example of **NL** language (in fact $\textbf{NL} - \textbf{complete}$) is REACHABILITY $= \{(G, s, t) |$ directed graph G has a path f
We can devise a NTM that stores into the working tape a pointer with the position of the input string in the input tape: this pointer would take only $O(\log_2 |w|)$ cells instead of holding the entire string in $O(|w|)$.

Data: Directed graph $G = (V, E)$ and nodes s, t
Result: Y if there is a path from s to t, N otherwise
p points to s, store p in the working tape;
counter $= 1$, store *counter* in the working tape;
#LABEL;
if *p points to t* **then**
  return Y;
end
Guess a point v in G, p' points to v;
if *p points to a node with no edge to the node pointed by* p' **then**
  return N;
end
$p = p'$;
counter $=$ *counter* $+ 1$;
if *counter* $\leq |V|$ **then**
  ***goto** #LABEL;*
else
  return N;
end

Algorithm 6: Algorithm for REACHABILITY $\in$ **NL**

7 Savitch's Theorem

We have previously devised a TM that decides REACHABILITY $= \{(G, s, t)|$ directed graph G has a path from s to $t\}$ in $O(\log_2 n)$. However, we can proove that REACHABILITY $\in$ DSPACE$((\log_2 n)^2)$ using recursion: note that a path from nod s to node t exists iff there is a node u between them such that s and t reach u in half the steps.

Data: Directed graph $G = (V, E)$, nodes s, t and an integer k
Result: Y if there is a path from s to t requiring at most k steps, N otherwise
// Does s reach t in zero steps?
if $k = 0$ **then**
  if $s = t$ **then**
    return Y;
  else
    return N;
  end
end
// Does s reach t in at most one step?
if *k=1* **then**
  if $s = t$ *or* $(s, t) \in E$ **then**
    return Y;
  else
    return N;
  end
end
// Look for a middle node
foreach $u \in V$ **do**
  if *exists-path*$(G, s, u, \lfloor k/2 \rfloor)$ *and exists-path*$(G, s, u, \lceil k/2 \rceil)$ **then**
    return Y;
  else
    return N;
  end
end
return N;

Algorithm 7: exists-path

Each calls requires at most $\log_2 |V| \in O(log_2 n)$ cells, since the space used by the previous two calls is freed up, and no more than $\log_2 |V|$ calls are made so overall the algorithm performs in $O(\log_2 n)$.

This reasoning is generalized to any language in **NSPACE** by the *Savitch's theorem*. When considering space, there is in fact not much difference between deterministic and non-deterministic Turing machines: unlike time, space can be reused.

$$\text{NSPACE}(f(n)) \subseteq \text{DSPACE}(f^2(n)) \quad \forall n \in \mathbb{N}, \forall f : \mathbb{N} \to \mathbb{N} | f(n) \geq \log n$$

As a corollary of Savitch's theorem, the difference between space required by TMs and NTMs is none in polynomial space.

$$\textbf{PSPACE} = \bigcup_{c \geq 1} \text{DSPACE}(n^c), \quad \textbf{NSPACE} = \bigcup_{c \geq 1} \text{DSPACE}(n^c), \quad \textbf{PSPACE} = \textbf{NPSPACE}$$

8 Oracles

It is quite common for algorithms to rely on *subroutines* to solve problems, and in fact TMs can rely on *oracles* to solve specific types of problems. Considering the problem VCOVER, we might want to find the optimal (smallest) vertex cover for a given graph.

$$\text{MIN-COVER} = \{(G, k) | G \text{ is an undirected graph and } k \text{ is the size of its smallest vertex cover}\}$$

This means devising a procedure that verifies that there is

1. A vertex cover of size at most k (VCOVER $\in$ **NP**).
2. No vertex cover of size $k-1$ ($\overline{\text{VCOVER}} \in$ **coNP**). If we believe that **NP** $\neq$ **coNP** then $\overline{\text{VCOVER}} \notin$ **NP**.

So we can rephrase the problem as MIN-COVER $= \{(G,k) | (G,k) \in \text{VCOVER} \wedge (G,k-1) \notin \text{VCOVER}\}$. Given an oracle `check-vcover` that decides VCOVER, we following procedure requires only a polynomial number of steps to decide MIN-COVER.

1. Let `result1 = check-vcover(G,k);`
2. Let `k' = k-1;`
3. Let `result2 = check-vcover(G,k');`
4. If `result1 = True` and `result2 = True=` then `Accept`, otherwise `Reject`

A TM with an oracle for L has an additional oracle tape (read-write), and additional states $q_?, q_{yes}, q_{not}$: to ask to the oracle if $w \in L$, the TM writes w in the oracle tape and moves to $q_?$: if the next state is q_{yes}, then $w \in L$.

We can now define additional complexity classes: let C be some complexity class

- $\mathbf{P}^C$ contains the languages that can be decided in polynomial time by a TM with an oracle for some language $L \in C$.
- $\mathbf{NP}^C$ contains the languages that can be decided in polynomial time by a NTM with an oracle for some language $L \in C$.

So MIN-VCOVER $\in P^{\mathbf{NP}}$. Note that

- **NP** $\subseteq \mathbf{P}^{\mathbf{NP}}$ and **coNP** $\subseteq \mathbf{P}^{\mathbf{NP}}$.
- A *polynomial time hierarchy* is an infinite hierarchy of complexity classes that generalizes **NP** and $\mathbf{co}-\mathbf{NP}$ using oracles.

9 Search problems

Oracles are exactly the kind of tools needed to undestand the complexity of *search problems*: we can transfer the knowledge we have about a decision problem to its search version. Consider search problem

$$\text{FMIN-VCOVER} = \min\{|VC| | VC \text{ is a vertex cover of } G\}$$

We can devise a $O(|V|)$ algorithm that uses the VCOVER oracle to solve the problem. Let **FP** be the class of search problems solvable by a TM with an output tape in polynomial time, than

$k = |V| - 1$;
while *(G,k)* $\in$ *VCOVER* **do**
  $k = k$ *-1*
end
write k in the output tape

We can actually improve the previous algorithm by employing binary search and reducing the number of steps from $O(|V|)$ to $O(\log_2 |V|)$. This improved algorithm proves thats FMIN-VCOVER $\in \mathbf{FP}^{\mathbf{NP}[\log_2 n]}$, that is the class of search problems that can be solved in polynomial time with $O(\log_2 n)$ calls to an oracle for a **NP** language.

9.1 The Travelling Salesman Problem

Given a weighted undirected graph $G = (V, E, \lambda)$, with a weight function $\lambda : E \to \mathbb{N}$, we consider the *cost* of a path to be the sum of its traversed edges. In the *Functional Travelling Salesman Problem* (FTSP) the goal is to visit all the nodes of G exactly ones, through what is called an *Hamiltonian path* that has a minimum cost.

To solve this search problem we might consider using an oracle for the related decision problem $TSP \in$ **NP**.

TSP $= \{(G, K) |$ weighted, undirected graph G with an Hamilton cycle costing at most $k\}$

// m bits to encode each cost + $|E|$ bits to encode each edge $\rightarrow k \in O(2^{m+|E|})$
k is the sum of the cost of all edges in G;
if *(G,k)* $\notin$ *TSP* **then**
  write $\perp$ *in the output tape and halt;*
end
while *(G,k)* $\in$ *TSP* **do**
  $k = k - 1$;
end
write $k + 1$ *in the output tape;*

A better algorithm relies again on binary search, resulting in FTSP $\in \mathbf{FP^{NP}}$.

$a = 0$;
// m bits to encode each cost + $|E|$ bits to encode each edge $\rightarrow b \in O(2^{m+|E|})$
b is the sum of the cost of all edges in G;
// binary search, so number of iterations $\in O(\log_2 2^{m+|E|}) = O(m + |E|)$
while $a \leq b$ **do**
  $k = \lfloor (a + b)/2 \rfloor$;
  if $(G, k - 1) \in$ *TSP* **then**
    // Cost is striclty smaller than middle point, move interval to the left
    $b = k - 1$;
  else
    if *(G,k)* $\in$ *TSP* **then**
      // k is the minimum cost
      write k *in the output tape and halt;*
    else
      // Cost is striclty larger than middle point, move interval to the right
      $a = k + 1$;
    end
  end
end
write $\perp$ *in the output tape;*

Finally, since TSP $\in$ **NP-complete** also the specific FTSP $\in$ **NP-complete**.

9.2 Restaurant

Given a set of k tables $T = \{1, \cdots, k\}$, a set of n people $P = \{1, \cdots, n\}$ and for each person a list $L_i, i \in P$ of guests that person i doesn't like. The problem of verifying if a given configuration is valid can be formalized as

RESTAURANT $= \{(P, T, L_1, \cdots, L_n) | (P, T, L_1, \cdots, L_n)$ is a valid input which has a table assignment$\}$

For the problem RESTAURANT we can devise a NTM that

- Guesses for each person i a table t_i in polynomial time.
- Checks for each guess that no person in L_i is in t_i in polynomial time, if so accepts (otherwise rejects).

Recall the Vertex Coloring problem. We can devise a procedure that constructs from $(G, k), G = (V, E)$ a string $(P, T, L_1, \cdots, L_n)$ such that (G, k) has a k-coloring iff there is a table assignment for $(P, T, L_1, \cdots, L_n)$.

- $P = V$.

- For each i, construct L_i as the set of nodes of G connected to node i.
- $T = \{1, \cdots, k'\}$ for $k' = k$ colors, that requires linear time in k. However k is an integer encoded with m bits, so in the worst case $k = 2^m - 1$ and the procedure requires $O(2^m)$.

Still we can devise a polynomial procedure that, before constructing the string as mentioned sets $k' = |V|$ if $k \geq |V|$. It is equivalent to the case in which we might have more colors than nodes a trivial k-coloring (one color per node): we can simply reduce the colors to the number of nodes, and consequently the resulting list L_i will have a maximum size $|V|$.

Such a procedure is indeed a reduction

- If G has a k-coloring μ (as said, with $k \leq |V|$), consider the function $\tau = \mu$: while μ assigns a color to each node, τ assigns a table to each guest. We have then that $\mu(i) \neq \mu(j), \forall \{i, j\} \in E$ and $\tau(i) \neq \tau(j), \forall i, j \in P$, since $\exists \{i, j\} \in E \Leftrightarrow i \in L_j \wedge j \in L_i$.
- If $(P, T, L_1, \cdots, L_n)$ has a table coloring τ, then $i \in L_j, j \in L_i \Rightarrow \{i, j\} \in E \Rightarrow \tau(i) \neq \tau(j)$ so $\mu = \tau$ is also a k-coloring for G.

9.3 Labels

Given an undirected graph $G = (V, E)$, two integers k, k' and a set integer *labels* X, can we assign a label $p_v \in X$ to each node $v \in V$ such that

- $\sum_v p_v < k$, so the sum of the labels is strictly smaller than k.
- $\sum_{e \in E} C_e \geq k'$, so the cost of each edge is at least k', where the cost of an edge $C_e = \max\{p_u, p_v\}, = \{u, v\} \in E$.

An input for such a problem can be encoded as (G, k, k', X), and labeling as a function $l : V \to X$ such that $\sum_{v \in V} l(v) < k$ and $\sum_{\{u,v\} \in E} \max\{l(u), l(v)\} \geq k'$. The resulting problem can be formalized as

$$\text{LABELS} = \{(G, k, k', X) | (G, k, k', X) \text{ is a valid input that has a labeling}\}$$

For the problem LABELS we can devise a NTM that decides the language in polynomial time.

- Guesses for each node $v \in V$ a label $p_v \in X$, if X is encoded using m bits then labels take $|V| \cdot m$ cells.
- Check $\sum_{v \in V} l(v) < k$, that requires summing $|V|$ numbers.
- Check $\sum_{\{u,v\} \in E} \max\{l(u), l(v)\} \geq k'$, that requires summing $|E|$ numbers.

So LABELS $\in$ **NP**, but is LABELS $\in$ **NP** $-$ **complete**? To prove it, we can show that VCOVER $\leq$ LABELS.

- The first sum $\sum_v p_v < k$ can be seen as counting the chosen nodes of a vertex cover.
- The second sum $\sum_{e \in E} \max\{p_u, p_v\}, = \{u, v\} \in E$ can be seen as counting how many edges are "touched" by the chosen nodes.

Thus, given a string $G = (V, E)$ we can construct a string $(\bar{G}, \bar{k}, \bar{k}', X)$ such that G has a vertex cover of size at most k iff $(\bar{G}, \bar{k}, \bar{k}', X)$ has a labeling, so

- $\bar{G} = G$, $X = \{0, 1\}$
- $\bar{k} = k + 1$, since the first sum is strictly smaller than $\bar{k}$.
- $\bar{k}' = |E|$, since the second sum counts all the edges.

Such a procedure is polynomial, since we are just copying G and counting nodes and edges of G, and a proper reduction since

- If $(G, k) \in VCOVER$, then G has a vertex cover of size at most k. Let the labeling function $l(v) = 1$ if $v \in VC$ ($l(v) = 0$ otherwise), the first sum is strictly less than $\bar{k}$: $\sum_{v \in V} l(v) = |VC| \leq k < \bar{k}$. The second sum $\sum_{\{u,v\} \in E} \max\{l(u), l(v)\} = \bar{k}' = |E|$, since the cost of each edge is 1.

- If l is a labeling of $(\bar{G}, \bar{k}, \bar{k}', X)$, then let VC be the set of all nodes $v \in \bar{G}$ such that $l(v) = 1$. With $X = \{0, 1\}$, the first sum $\sum_{v \in V} l(v) = |VC| \leq k < \bar{k}$. And since $X = \{0, 1\}$, $\sum_{\{u,v\} \in E} \max\{l(u), l(v)\} = \bar{k}' = |E|$. So VC is a vertex cover.

Consider now the search problem MAX-LABELS: given an undirected graph G an integer k and a set integer *labels* X, what is the largest k' such that $(G, k, k', X) \in$ LABELS? Using an oracle LABELS, we can solve the problem in polynomial-time by devising a TM that takes (G, k, X).

$x = \max(X)$;
$a = 0$;
$b = |E| \cdot x$;
while $a <b$ **do**
  $k' = \lfloor (a + b)/2 \rfloor$;
  if $(G, k, k' + 1, X) \in LABELS$ **then**
    a = k'+1;
  else
    if $(G, k, k') \in LABELS$ **then**
      write k and halt;
    else
      b = k' -1;
    end
  end
end
write $\perp$

The algorithm works in $O(log_2 C)$, where $C = |E| \cdot x$. The number C requires at most $|E| + m$ bits, where m bits are used to encode x, so $C \in O(2^{|E|+m})$. We conclude, since the algorithm works in $O(|E| + m)$ wrt the size of the input, that MAX-LABELS $\in \mathbf{FP^{NP}}$.