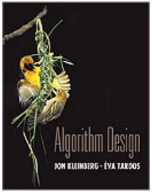
Complexity: Problems and Classes

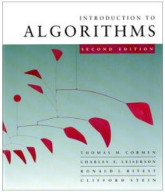
Maria Serna

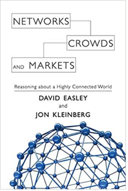
Spring 2024

Algorithmics: Basic references

- Kleinberg, Tardos. Algorithm Design, Pearson Education, 2006.
- Cormen, Leisserson, Rivest and Stein. Introduction to algorithms. Second edition, MIT Press and McGraw Hill 2001.
- Easley, Kleinberg. Networks, Crowds, and Markets: Reasoning About a Highly Connected World, Cambridge University Press, 2010

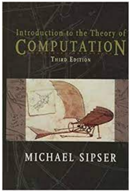


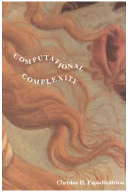


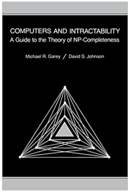


Computational Complexity: Basic references

- Sipser Introduction to the Theory of Computation 2013.
- Papadimitriou Computational Complexity 1994.
- Garey and Johnson Computers and Intractability: A Guide to the Theory of NP-Completeness 1979







Growth of functions: Asymptotic notations

We consider only functions defined on the natural numbers.

$$f, g : \mathbb{N} \to \mathbb{N}$$

O-notation

For a given function $g(n)$

$$O(g(n)) = \{f(n) \mid \text{there exists a positive constant } c \text{ and } n_0 \geq 0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$$

Equivalently, the set of functions that verify

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

$$\begin{aligned} 5n^3 + 2n^2 &= O(2^n) \\ 5n^3 + 2n^2 &= O(n^4) \\ 2^n &= O(2^{2n}) \\ 2^n &= O(2^{n \log n}) \end{aligned}$$

It is used for asymptotic upper bound.

Although $O(g(n))$ is a set we write $f(n) = O(g(n))$ to indicate that $f(n)$ is a member of $O(g(n))$

Ω-notation

For a given function $g(n)$

$$\Omega(g(n)) = \{f(n) \mid \text{there exists positive constants } c \text{ and } n_0 \text{ such that} \\ 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}$$

Equivalently, the set of functions that verify

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$$

$$\begin{aligned} 5n^3 + 2n^2 &= \Theta(n^3) \\ 5n^3 + 2n^2 &= \Omega(n^3) \\ 5n^3 + 2n^2 &= \Omega(n^2) \\ 2^n &= \Omega(2^{n/2}) \end{aligned}$$

It is used for asymptotic lower bound.

Θ-notation

For a given function $g(n)$

$$\Theta(g(n)) = \{f(n) \mid \text{there are positive constants } c_1,\ c_2, \text{ and } n_0 \geq 0 \ \}$$
$$\text{such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0\}$$

Equivalently, the set of functions that verify

$$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

$$5n^3 + 2n^2 = \Theta(n^3)$$

$$5n^3 + 2n^2 \notin \Theta(n^2)$$

It is used for asymptotic equivalence

o-notation

For a given function $g(n)$

$$o(g(n)) = \{f(n) \mid \text{for any positive constant } c \text{ there is } n_0 \geq 0 \text{ such that} \\ 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$$

Note that $f(n) = O(n)$ implies $f(n) \leq cg(n)$ asymptotically for some c but $f(n) = o(n)$ implies $f(n) \leq cg(n)$ asymptotically for any c and when $f(n) = o(g(n))$ it holds that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

It is used for asymptotic upper bounds that are not asymtotically tight.

ω-notation

$f(n) \in \omega(g(n))$ iff $g(n) \in o(f(n))$

Algorithm's analysis

- Time
- Space

Algorithm $\mathcal{A}$ on input x takes time $t(x)$.
$|x|$ denotes the size of input x.

Definition

The cost function of algorithm $\mathcal{A}$ is a function from $\mathbb{N}$ to $\mathbb{N}$ defined as

$$\mathcal{C}_{\mathcal{A}}(n) = \max_{|x|=n} t(x)$$

Fundamental growth functions

- Polynomial time
- Exponential time

Fundamental growth functions

- Polynomial time
  $\mathcal{C}_{\mathcal{A}}(n) = O(n^c)$, for some constant c.
- Exponential time

Fundamental growth functions

- Polynomial time
  $\mathcal{C}_{\mathcal{A}}(n) = O(n^c)$, for some constant c.
- Exponential time
  $\mathcal{C}_{\mathcal{A}}(n) = 2^{O(n^c)}$, for some constant c.

Fundamental growth functions

- Polynomial time
  $\mathcal{C}_{\mathcal{A}}(n) = O(n^c)$, for some constant c.
- Exponential time
  $\mathcal{C}_{\mathcal{A}}(n) = 2^{O(n^c)}$, for some constant c.
- Quasi-polynomial time
- Pseudo polynomial time

Fundamental growth functions

- Polynomial time
  $\mathcal{C}_{\mathcal{A}}(n) = O(n^c)$, for some constant c.
- Exponential time
  $\mathcal{C}_{\mathcal{A}}(n) = 2^{O(n^c)}$, for some constant c.
- Quasi-polynomial time
  $\mathcal{C}_{\mathcal{A}}(n) = 2^{O(\log^c n)}$, for some constant c.
- Pseudo-polynomial time
  $\mathcal{C}_{\mathcal{A}}(n) = O((mW)^c)$, for some constant c, but input size is $O(m + \log W)$

Fundamental growth functions

- Polynomial time
  $\mathcal{C}_{\mathcal{A}}(n) = O(n^c)$, for some constant c.
- Exponential time
  $\mathcal{C}_{\mathcal{A}}(n) = 2^{O(n^c)}$, for some constant c.
- Quasi-polynomial time
  $\mathcal{C}_{\mathcal{A}}(n) = 2^{O(\log^c n)}$, for some constant c.
- Pseudo-polynomial time
  $\mathcal{C}_{\mathcal{A}}(n) = O((mW)^c)$, for some constant c, but input size is $O(m + \log W)$
- Similar definitions replacing time by space
  Most used PSPACE polynomial space

Problem types

- Decision
  Input x
  Property $P(x)$

  Example: Given a graph and two vertices, is there a path joining them?

- Function
  Input x
  Compute y *such that* $Q(x, y)$

  Example: Given a graph and two vertices, compute the minimum distance between them.

Problem types

- Decision
  Input x
  Property $P(x)$

Problem types

- Decision

  Input x

  Property $P(x)$

Coding inputs on alphabet Σ a problem is a set

Problem types

- Decision

  Input x

  Property $P(x)$

Coding inputs on alphabet Σ a problem is a set $\{x \mid P(x)\} \in \mathcal{P}(\Sigma^*)$

Problem types

- Decision

  Input x

  Property $P(x)$

  Coding inputs on alphabet Σ a problem is a set $\{x \mid P(x)\} \in \mathcal{P}(\Sigma^*)$

- Function

  Input x

  Compute y *such that* $Q(x, y)$

Problem types

- Decision

  Input x

  Property $P(x)$

  Coding inputs on alphabet Σ a problem is a set $\{x \mid P(x)\} \in \mathcal{P}(\Sigma^*)$

- Function

  Input x

  Compute y *such that* $Q(x, y)$

  Coding inputs/outputs on alphabet Σ a deterministic algorithm solving a problem determines a function

Problem types

- Decision

  Input x

  Property $P(x)$

  Coding inputs on alphabet Σ a problem is a set
  $\{x \mid P(x)\} \in \mathcal{P}(\Sigma^*)$

- Function

  Input x

  Compute y *such that* $Q(x, y)$

  Coding inputs/outputs on alphabet Σ a deterministic algorithm solving a problem determines a function
  $f : \Sigma^* \to \Sigma^*$ s.t., for any x, $Q(x, f(x))$ is true.

Decision problem classes

- Undecidable
  No algorithm can solve the problem.
- Decidable
  There is an algorithm solving them.
  - P:
    There is an algorithm solving it with polynomial cost.
  - EXP
    There is an algorithm solving it with exponential cost.
  - PSPACE
    There is an algorithm solving it within polynomial space.

NP: non-deterministic polynomial time

It is possible to define a certificate y and a property $P(x, y)$ such that

- If x is an input with answer yes, there is y such that $P(x, y)$ is true,
- $P(x, y)$ can be decided in polynomial time, given x and y.
- y has polynomial size with respect to $|x|$.

Problems with a polynomial time verifier

NP: non-deterministic polynomial time

It is possible to define a certificate y and a property $P(x, y)$ such that

- If x is an input with answer yes, there is y such that $P(x, y)$ is true,
- $P(x, y)$ can be decided in polynomial time, given x and y.
- y has polynomial size with respect to $|x|$.

Problems with a polynomial time verifier
$\{x \mid \exists y P(x, y)\}$

Some decision problems

Bipartiteness (BIP)
Given a graph determine whether it is bipartite.

Perfect matching (PMATCH)
Given a graph determine whether it has a perfect matching.

Hamiltonian Cycle (HC)
Given a graph determine whether it has a Hamiltonian circuit.

In which classes?

NP-hardness

- It is an open question whether P = NP or NP = EXP. Most believed is that P $\neq$ NP
- Π is NP-hard means that a polynomial time algorithm for Π can be reused to solve in polynomial time any problem in P.
- Decision problem A is NP-complete iff $A \in$ NP and A is NP-hard. Look at Garey and Johnson's book for a big list of NP-hard/complete problems.

NP-hardness

- It is an open question whether P $=$ NP or NP $=$ EXP. Most believed is that P $\neq$ NP
- Π is NP-hard means that a polynomial time algorithm for Π can be reused to solve in polynomial time any problem in P.
- Decision problem A is NP-complete iff $A \in$ NP and A is NP-hard. Look at Garey and Johnson's book for a big list of NP-hard/complete problems.
- The NP-hardness of a problem is assessed through reductions.

Reductions

- Let A and B be decision problems
- A function $f : \mathbb{N} \to \mathbb{N}$ is a reduction from A to B if $x \in A$ iff $f(x) \in B$

Reductions

- Let A and B be decision problems
- A function $f : \mathbb{N} \to \mathbb{N}$ is a reduction from A to B if $x \in A$ iff $f(x) \in B$
- f is a polynomial time reduction if in addition f can be computed in polynomial time.

Reductions

- Let A and B be decision problems
- A function $f : \mathbb{N} \rightarrow \mathbb{N}$ is a reduction from A to B if $x \in A$ iff $f(x) \in B$
- f is a polynomial time reduction if in addition f can be computed in polynomial time.
- A is reducible to B ($A \leq B$) if there is a polynomial time reduction from A to B.

Reductions

- Let A and B be decision problems
- A function $f : \mathbb{N} \to \mathbb{N}$ is a reduction from A to B if $x \in A$ iff $f(x) \in B$
- f is a polynomial time reduction if in addition f can be computed in polynomial time.
- A is reducible to B ($A \leq B$) if there is a polynomial time reduction from A to B.

Theorem

If $A \leq B$ and $B \in P$ then $A \in P$

Reductions

- Let A and B be decision problems
- A function $f : \mathbb{N} \to \mathbb{N}$ is a reduction from A to B if $x \in A$ iff $f(x) \in B$
- f is a polynomial time reduction if in addition f can be computed in polynomial time.
- A is reducible to B ($A \leq B$) if there is a polynomial time reduction from A to B.

Theorem

If $A \leq B$ and $B \in P$ then $A \in P$

This type of reduction is calle many-one polynomial reduction, sometiomes we use $\leq_m^p$ to distinguish from other reducibilities.

Completeness

- Let $\leq$ be a reducibility among problems and $\mathcal{C}$ a class of problems. $\mathcal{C}$ is closed under $\leq$ if $A \leq B$ and $B \in \mathcal{C}$ implies $A \in \mathcal{C}$.

Completeness

- Let $\leq$ be a reducibility among problems and $\mathcal{C}$ a class of problems. $\mathcal{C}$ is closed under $\leq$ if $A \leq B$ and $B \in \mathcal{C}$ implies $A \in \mathcal{C}$.
- Let $\leq$ be a reducibility among problems and $\mathcal{C}$ a class of problems closed under $\leq$

Completeness

- Let $\leq$ be a reducibility among problems and $\mathcal{C}$ a class of problems. $\mathcal{C}$ is closed under $\leq$ if $A \leq B$ and $B \in \mathcal{C}$ implies $A \in \mathcal{C}$.
- Let $\leq$ be a reducibility among problems and $\mathcal{C}$ a class of problems closed under $\leq$
  - Problem B is hard for class $\mathcal{C}$ under $\leq$ if, for each $A \in \mathcal{C}$, $A \leq B$.

Completeness

- Let $\leq$ be a reducibility among problems and $\mathcal{C}$ a class of problems. $\mathcal{C}$ is closed under $\leq$ if $A \leq B$ and $B \in \mathcal{C}$ implies $A \in \mathcal{C}$.
- Let $\leq$ be a reducibility among problems and $\mathcal{C}$ a class of problems closed under $\leq$
  - Problem B is hard for class $\mathcal{C}$ under $\leq$ if, for each $A \in \mathcal{C}$, $A \leq B$.
  - Problem B is complete for class $\mathcal{C}$ under $\leq$ if, B is hard for $\mathcal{C}$ and $B \in \mathcal{C}$.

Completeness

- Let $\leq$ be a reducibility among problems and $\mathcal{C}$ a class of problems. $\mathcal{C}$ is closed under $\leq$ if $A \leq B$ and $B \in \mathcal{C}$ implies $A \in \mathcal{C}$.
- Let $\leq$ be a reducibility among problems and $\mathcal{C}$ a class of problems closed under $\leq$
  - Problem B is hard for class $\mathcal{C}$ under $\leq$ if, for each $A \in \mathcal{C}$, $A \leq B$.
  - Problem B is complete for class $\mathcal{C}$ under $\leq$ if, B is hard for $\mathcal{C}$ and $B \in \mathcal{C}$.
- P, NP, PSPACE, EXP are closed under $\leq$.

Completeness

- Let $\leq$ be a reducibility among problems and $\mathcal{C}$ a class of problems. $\mathcal{C}$ is closed under $\leq$ if $A \leq B$ and $B \in \mathcal{C}$ implies $A \in \mathcal{C}$.
- Let $\leq$ be a reducibility among problems and $\mathcal{C}$ a class of problems closed under $\leq$
  - Problem B is hard for class $\mathcal{C}$ under $\leq$ if, for each $A \in \mathcal{C}$, $A \leq B$.
  - Problem B is complete for class $\mathcal{C}$ under $\leq$ if, B is hard for $\mathcal{C}$ and $B \in \mathcal{C}$.
- P, NP, PSPACE, EXP are closed under $\leq$.
- A NP-complete problem is a problem complete for NP under $\leq$.

Completeness

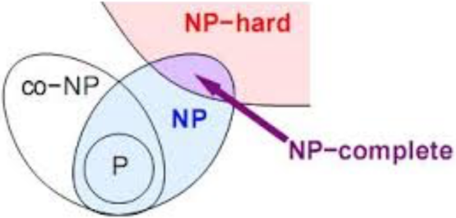
- Let $\leq$ be a reducibility among problems and $\mathcal{C}$ a class of problems. $\mathcal{C}$ is closed under $\leq$ if $A \leq B$ and $B \in \mathcal{C}$ implies $A \in \mathcal{C}$.
- Let $\leq$ be a reducibility among problems and $\mathcal{C}$ a class of problems closed under $\leq$
  - Problem B is hard for class $\mathcal{C}$ under $\leq$ if, for each $A \in \mathcal{C}$, $A \leq B$.
  - Problem B is complete for class $\mathcal{C}$ under $\leq$ if, B is hard for $\mathcal{C}$ and $B \in \mathcal{C}$.
- P, NP, PSPACE, EXP are closed under $\leq$.
- A NP-complete problem is a problem complete for NP under $\leq$.
- $\leq$ is a transitive relation.

NP-completeness

A problem A is NP-complete if:

1. $A \in$ NP, and
2. for every $B \in$ NP, $B \leq A$.

If for every $B \in$ NP, $B \leq A$ but $A \notin$ NP then A is said to be NP-hard.



Lemma

If A is NP-complete, then $A \in$P iff P=NP.

Lemma

If A is NP-complete, then $A \in$P iff P=NP.

- Once we prove that a problem is NP-complete, either A has no efficient algorithm or all NP problems are in P.

Lemma

If A is NP-complete, then $A \in$P iff P=NP.

- Once we prove that a problem is NP-complete, either A has no efficient algorithm or all NP problems are in P.
- *Majority conjecture:* P $\neq$ NP

Lemma

If A is NP-complete, then $A \in$P iff P=NP.

- Once we prove that a problem is NP-complete, either A has no efficient algorithm or all NP problems are in P.
- *Majority conjecture:* P $\neq$ NP
- To prove a problem is NP-complete, we just have to find a reduction from a problem known to be NP-complete.

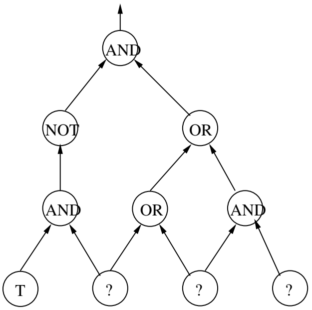
Lemma

If A is NP-complete, then $A \in$P iff P=NP.

- Once we prove that a problem is NP-complete, either A has no efficient algorithm or all NP problems are in P.
- *Majority conjecture:* P $\neq$ NP
- To prove a problem is NP-complete, we just have to find a reduction from a problem known to be NP-complete.
- We need as a seed a first NP-complete problem.

CIRCUIT SAT.

CIRCUIT SAT: Given a Boolean circuit with gates *AND*, *OR*, *NOT*, and the input gates *T*, *F* and *?*, and one output gate. Is there an an assignment to the input gates (?), such that the circuit evaluates to T?



For example if the input to ? is T,F,T, the output is F
if the input is F,T,T, the output is T

Any NP problem can be "expressed" as CIRCUIT SAT.

- We want to show that if $A \in \mathrm{NP}$, then $A \leq$ CIRCUIT SAT.

Any NP problem can be "expressed" as CIRCUIT SAT.

- We want to show that if $A \in \mathrm{NP}$, then $A \leq$ CIRCUIT SAT.
- A is a decision NP problem $\Rightarrow$ there is a polynomial-time algorithm $\mathcal{A}$ which given an instance x and a witness solution c of A, checks in polynomial time (in the length of $|x|$) if c is a valid certificate for x.

Any NP problem can be "expressed" as CIRCUIT SAT.

- We want to show that if $A \in$ NP, then $A \leq$ CIRCUIT SAT.
- A is a decision NP problem $\Rightarrow$ there is a polynomial-time algorithm $\mathcal{A}$ which given an instance x and a witness solution c of A, checks in polynomial time (in the length of $|x|$) if c is a valid certificate for x.
- Any polynomial-time algorithm (TM) can be expressed as a polynomial-size circuit, whose input gates encode the input to the algorithm, and the ? input gates are feeding the witness c. If the algorithm solves a decision problem (Y/N), the output of the circuit will be 1/0.

Any NP problem can be "expressed" as CIRCUIT SAT.

- We want to show that if $A \in \mathrm{NP}$, then $A \leq$ CIRCUIT SAT.
- A is a decision NP problem $\Rightarrow$ there is a polynomial-time algorithm $\mathcal{A}$ which given an instance x and a witness solution c of A, checks in polynomial time (in the length of $|x|$) if c is a valid certificate for x.
- Any polynomial-time algorithm (TM) can be expressed as a polynomial-size circuit, whose input gates encode the input to the algorithm, and the ? input gates are feeding the witness c. If the algorithm solves a decision problem (Y/N), the output of the circuit will be $1/0$.
- There is a way to feed c and get output 1 iff there is a valid cerificate.

The seminal theorem: Cook-Levin's Theorem

Therefore, given any instance x for A, we can construct in poly-time instance C of CIRCUIT SAT whose known inputs are the bits of x, and whose unknown inputs are the bits of x, and such that the output of C is 1 iff $\mathcal{A}$ outputs YES on input (x, c).

The seminal theorem: Cook-Levin's Theorem

Therefore, given any instance x for A, we can construct in poly-time instance C of CIRCUIT SAT whose known inputs are the bits of x, and whose unknown inputs are the bits of x, and such that the output of C is 1 iff $\mathcal{A}$ outputs YES on input (x, c).

Theorem (Cook-Levin's theorem)

CIRCUIT-SAT is NP-complete.

Boolean formulas

- A Boolean variable x can take values 0,1.

Boolean formulas

- A Boolean variable x can take values 0,1.
- A Boolean formula is an expression constructed from Boolean variables and connectives, negation ($\neg\phi$ or $\overline{\phi}$), disjunction ($\vee$), and conjunction ($\wedge$).

Boolean formulas

- A Boolean variable x can take values 0,1.
- A Boolean formula is an expression constructed from Boolean variables and connectives, negation ($\neg\phi$ or $\overline{\phi}$), disjunction ($\vee$), and conjunction ($\wedge$).
- A Boolean formula ϕ is satisfiable if there is a truth assignment $T : X \rightarrow \{0,1\}$ to the variables in ϕ such that $T(\phi) = 1$.

Boolean formulas

- A Boolean variable x can take values 0,1.
- A Boolean formula is an expression constructed from Boolean variables and connectives, negation ($\neg\phi$ or $\overline{\phi}$), disjunction ($\vee$), and conjunction ($\wedge$).
- A Boolean formula ϕ is satisfiable if there is a truth assignment $T : X \to \{0,1\}$ to the variables in ϕ such that $T(\phi) = 1$.
- For example, for $X = \{x_1, x_2, x_3\}$,

$$\phi = (x_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_2 \vee x_3)$$

is satisfiable, take $T(x_1) = T(x_2) = 0$, $T(x_3) = 1$ then $T(\phi) = 1$.

Boolean formulas: normal forms

Boolean formulas: normal forms

- A literal is a Boolean variable x or a negation of a Boolean variable $\bar{x}$.

Boolean formulas: normal forms

- A literal is a Boolean variable x or a negation of a Boolean variable $\bar{x}$.
- A clause is a disjunction (conjunction) of literals.

Boolean formulas: normal forms

- A literal is a Boolean variable x or a negation of a Boolean variable $\bar{x}$.
- A clause is a disjunction (conjunction) of literals.
- A Boolean formula ϕ in Conjunctive Normal Form (CNF) is a conjunction of clauses, $\phi = \bigwedge_{i=1}^{m}(C_i)$, where each clause $C_i = \bigvee_{j=1}^{k_i}\{l_j\}$.

  For example, for $X = \{x_1, x_2, x_3\}$, a CNF formula is

$$\phi = (x_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_2 \vee x_3)$$

Boolean formulas: normal forms

- A literal is a Boolean variable x or a negation of a Boolean variable $\bar{x}$.
- A clause is a disjunction (conjunction) of literals.
- A Boolean formula ϕ in Conjunctive Normal Form (CNF) is a conjunction of clauses, $\phi = \bigwedge_{i=1}^{m}(C_i)$, where each clause $C_i = \bigvee_{j=1}^{k_i}\{l_j\}$.

  For example, for $X = \{x_1, x_2, x_3\}$, a CNF formula is

  $$\phi = (x_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_2 \vee x_3)$$

- A Boolean formula ϕ in Disjunctive Normal Form (DNF) is expressed as a disjunction of clauses, $\phi = \bigvee_{i=1}^{m}(C_i)$, where each clause $C_i = \bigwedge_{j=1}^{k_i}\{l_j\}$.

SAT problem and variations

Given a formula CNF formula ϕ on a set X of n variables,

- SAT: Is ϕ satisfiable?

SAT problem and variations

Given a formula CNF formula ϕ on a set X of n variables,

- SAT: Is ϕ satisfiable?
- k-SAT. Given a boolean formula in CNF $\phi = \bigwedge_{i=1}^{m}(C_i)$ in where each clause is a disjunction of exactly k literals, is ϕ satisfiable?
  Ex. 3-SAT
  $\phi = (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \bar{x_2} \vee x_4) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_4}) \wedge (x_1 \vee \bar{x_3} \vee \bar{x_4}) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_3}) \wedge (\bar{x_2} \vee x_3 \vee x_4) \wedge (\bar{x_1} \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_2} \vee x_3 \vee \bar{x_4}) \wedge (x_2 \vee \bar{x_3} \vee \bar{x_4}) \wedge (\bar{x_2} \vee \bar{x_3} \vee \bar{x_4})$
  $T(x_1) = 1$, $T(x_2) = T(x_3) = T(x_4) = 0$, satisfies the previous instance.

SAT problem and variations

Given a formula CNF formula ϕ on a set X of n variables,

- SAT: Is ϕ satisfiable?
- k-SAT. Given a boolean formula in CNF $\phi = \bigwedge_{i=1}^{m}(C_i)$ in where each clause is a disjunction of exactly k literals, is ϕ satisfiable?
  Ex. 3-SAT
  $\phi = (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \bar{x_2} \vee x_4) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_4}) \wedge (x_1 \vee \bar{x_3} \vee \bar{x_4}) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_3}) \wedge (\bar{x_2} \vee x_3 \vee x_4) \wedge (\bar{x_1} \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_2} \vee x_3 \vee \bar{x_4}) \wedge (x_2 \vee \bar{x_3} \vee \bar{x_4}) \wedge (\bar{x_2} \vee \bar{x_3} \vee \bar{x_4})$
  $T(x_1) = 1$, $T(x_2) = T(x_3) = T(x_4) = 0$, satisfies the previous instance.

Given a DNF formula ϕ on a set X of n variables,

- DNF-SAT: Is ϕ satisfiable?

SAT problem and variations

Given a formula CNF formula ϕ on a set X of n variables,

- SAT: Is ϕ satisfiable?
- k-SAT. Given a boolean formula in CNF $\phi = \bigwedge_{i=1}^{m}(C_i)$ in where each clause is a disjunction of exactly k literals, is ϕ satisfiable?
  Ex. 3-SAT
  $\phi = (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \bar{x_2} \vee x_4) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_4}) \wedge (x_1 \vee \bar{x_3} \vee \bar{x_4}) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_3}) \wedge (\bar{x_2} \vee x_3 \vee x_4) \wedge (\bar{x_1} \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_2} \vee x_3 \vee \bar{x_4}) \wedge (x_2 \vee \bar{x_3} \vee \bar{x_4}) \wedge (\bar{x_2} \vee \bar{x_3} \vee \bar{x_4})$
  $T(x_1) = 1$, $T(x_2) = T(x_3) = T(x_4) = 0$, satisfies the previous instance.

Given a DNF formula ϕ on a set X of n variables,

- DNF-SAT: Is ϕ satisfiable?
- k-DNF-SAT: Given a boolean formula in DNF $\phi = \bigvee_{i=1}^{m}(C_i)$ in where each clause is a conjunction of exactly k literals, is ϕ satisfiable?

CIRCUIT SAT $\leq_m^p$ SAT

CIRCUIT SAT $\leq_m^p$ SAT

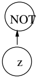
Given any circuit C , we can rewrite it as a CNF formula ϕ_C: for each gate we associate a variable to the output connection. We model the effect of the gate using at most three clauses.

CIRCUIT SAT $\leq_m^p$ SAT

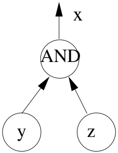
Given any circuit C , we can rewrite it as a CNF formula ϕ_C: for each gate we associate a variable to the output connection. We model the effect of the gate using at most three clauses.

(x)

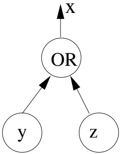
$(\bar{x})$



$(x \vee z) \wedge (\bar{x} \vee \bar{z})$

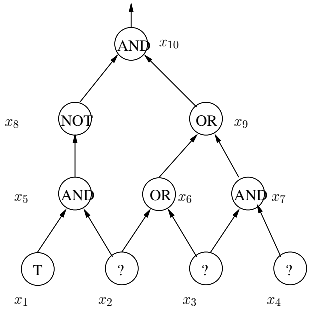


$(\bar{x} \vee y) \wedge (\bar{x} \vee z) \wedge (x \vee \bar{y} \vee \bar{z})$



$(\bar{y} \vee x) \wedge (\bar{z} \vee x) \wedge (z \vee y \vee \bar{x})$

Example.



$(x_1)\wedge$
$(\bar{x}_5 \vee x_1) \wedge (\bar{x}_5 \vee x_2) \wedge (x_5 \vee \bar{x}_1 \vee \bar{x}_2)\wedge$
$(\bar{x}_2 \vee x_6) \wedge (\bar{x}_3 \vee x_6) \wedge (x_2 \vee x_3 \vee \bar{x}_6)\wedge$
$(\bar{x}_7 \vee x_3) \wedge (\bar{x}_7 \vee x_4) \wedge (x_7 \vee \bar{x}_3 \vee \bar{x}_4)\wedge$
$(x_8 \vee x_5) \wedge (\bar{x}_8 \vee \bar{x}_5)\wedge$
$(\bar{x}_6 \vee x_9) \wedge (\bar{x}_7 \vee x_9) \wedge (x_6 \vee x_7 \vee \bar{x}_9)\wedge$
$(\bar{x}_{10} \vee x_8) \wedge (\bar{x}_{10} \vee x_9) \wedge (x_{10} \vee \bar{x}_8 \vee \bar{x}_9)\wedge$
(x_{10})

CIRCUIT SAT $\leq_m^p$ SAT

- Any truth assignment to the ? gates of C determines a truth assignment to the variables of ϕ_C

CIRCUIT SAT $\leq_m^p$ SAT

- Any truth assignment to the ? gates of C determines a truth assignment to the variables of ϕ_C
- To insure that the satisfying truth assignment of the resulting SAT formula is in 1-to-1 correspondence with the assignments on the gates in the given instance of CIRCUIT-SAT, we add a clause holding the variable corresponding to the output gate. Call this CNF formula $f(C)$.

CIRCUIT SAT $\leq_m^p$ SAT

- Any truth assignment to the ? gates of C determines a truth assignment to the variables of ϕ_C
- To insure that the satisfying truth assignment of the resulting SAT formula is in 1-to-1 correspondence with the assignments on the gates in the given instance of CIRCUIT-SAT, we add a clause holding the variable corresponding to the output gate. Call this CNF formula $f(C)$.
- C is satisfiable iff there is an assignment T of truth values such that the circuit outputs true iff the assignment obtained extending the truth values of T to coincide with the output of the gates in C is a satisfying assignment to $f(C)$.

CIRCUIT SAT $\leq_m^p$ SAT

- Any truth assignment to the ? gates of C determines a truth assignment to the variables of ϕ_C
- To insure that the satisfying truth assignment of the resulting SAT formula is in 1-to-1 correspondence with the assignments on the gates in the given instance of CIRCUIT-SAT, we add a clause holding the variable corresponding to the output gate. Call this CNF formula $f(C)$.
- C is satisfiable iff there is an assignment T of truth values such that the circuit outputs true iff the assignment obtained extending the truth values of T to coincide with the output of the gates in C is a satisfying assignment to $f(C)$.

Therefore, CIRCUIT SAT $\leq$ SAT.

CIRCUIT SAT $\leq_m^p$ SAT

- It remains to show that the formula $f(C)$ can be obtained in polynomial time.

CIRCUIT SAT $\leq_m^p$ SAT

- It remains to show that the formula $f(C)$ can be obtained in polynomial time.
- A circuit is directed acyclic graph, we can compute a topological sort in $O(|V| + |E|)$.

CIRCUIT SAT $\leq_m^p$ SAT

- It remains to show that the formula $f(C)$ can be obtained in polynomial time.
- A circuit is directed acyclic graph, we can compute a topological sort in $O(|V| + |E|)$.
- The topological sort provides an order to assign numbers to the variables.

CIRCUIT SAT $\leq_m^p$ SAT

- It remains to show that the formula $f(C)$ can be obtained in polynomial time.
- A circuit is directed acyclic graph, we can compute a topological sort in $O(|V| + |E|)$.
- The topological sort provides an order to assign numbers to the variables.
- We can obtain, from each gate, its number and the numbers of their input gates.

CIRCUIT SAT $\leq_m^p$ SAT

- It remains to show that the formula $f(C)$ can be obtained in polynomial time.
- A circuit is directed acyclic graph, we can compute a topological sort in $O(|V| + |E|)$.
- The topological sort provides an order to assign numbers to the variables.
- We can obtain, from each gate, its number and the numbers of their input gates.
- Then write, for each gate in order, the corresponding set of clauses.

CIRCUIT SAT $\leq_m^p$ SAT

- It remains to show that the formula $f(C)$ can be obtained in polynomial time.
- A circuit is directed acyclic graph, we can compute a topological sort in $O(|V| + |E|)$.
- The topological sort provides an order to assign numbers to the variables.
- We can obtain, from each gate, its number and the numbers of their input gates.
- Then write, for each gate in order, the corresponding set of clauses.
- All this process can be done in polynomial time.

CIRCUIT SAT $\leq_m^p$ SAT

- It remains to show that the formula $f(C)$ can be obtained in polynomial time.
- A circuit is directed acyclic graph, we can compute a topological sort in $O(|V| + |E|)$.
- The topological sort provides an order to assign numbers to the variables.
- We can obtain, from each gate, its number and the numbers of their input gates.
- Then write, for each gate in order, the corresponding set of clauses.
- All this process can be done in polynomial time.
- Therefore, CIRCUIT SAT $\leq_m^p$ SAT.

SAT is NP-complete

Theorem

SAT is NP-complete

Proof.

- As CIRCUIT SAT $\leq_m^p$ SAT, SAT is NP-hard
- It remains to show that $SAT \in NP$

SAT is NP-complete

Theorem

SAT is NP-complete

Proof.

- As CIRCUIT SAT $\leq_m^p$ SAT, SAT is NP-hard
- It remains to show that $SAT \in NP$
- The certificate is a truth assignment.

SAT is NP-complete

Theorem

SAT is NP-complete

Proof.

- As CIRCUIT SAT $\leq_m^p$ SAT, SAT is NP-hard
- It remains to show that $SAT \in NP$
- The certificate is a truth assignment.
- Evaluating a CNF formula on a particular truth assignment, requires to check that each clause gets at least one literal that evaluates to true.

SAT is NP-complete

Theorem

SAT is NP-complete

Proof.

- As CIRCUIT SAT $\leq_m^p$ SAT, SAT is NP-hard
- It remains to show that $SAT \in NP$
- The certificate is a truth assignment.
- Evaluating a CNF formula on a particular truth assignment, requires to check that each clause gets at least one literal that evaluates to true.
- This can be done in polynomial time.

$\square$

The 3-SAT

- Recall that the 3-SAT problem is a restricted version of SAT where each clause has exactly 3 literals.

The 3-SAT

- Recall that the 3-SAT problem is a restricted version of SAT where each clause has exactly 3 literals.
- Let $\phi = \{C_i\}_{i=1}^{m}$ be a CNF formula on a set X of variables. let z_i be the literal x_i or $\bar{x}_i$.
- We construct a formula $\phi' = f(\phi)$ on a set X' of variables ($X \subseteq X'$) having all clauses with 3 literals.

The 3-SAT

- Recall that the 3-SAT problem is a restricted version of SAT where each clause has exactly 3 literals.
- Let $\phi = \{C_i\}_{i=1}^{m}$ be a CNF formula on a set X of variables. let z_i be the literal x_i or $\bar{x}_i$.
- We construct a formula $\phi' = f(\phi)$ on a set X' of variables ($X \subseteq X'$) having all clauses with 3 literals.
- For each clause in ϕ, f determines a set of equivalent clauses, in the sense that the clause can be satisfied iff the formula corresponding to the set is satisfiable, to be included.
- We add some variables when needed.

The 3-SAT

- Recall that the 3-SAT problem is a restricted version of SAT where each clause has exactly 3 literals.
- Let $\phi = \{C_i\}_{i=1}^{m}$ be a CNF formula on a set X of variables. let z_i be the literal x_i or $\bar{x}_i$.
- We construct a formula $\phi' = f(\phi)$ on a set X' of variables ($X \subseteq X'$) having all clauses with 3 literals.
- For each clause in ϕ, f determines a set of equivalent clauses, in the sense that the clause can be satisfied iff the formula corresponding to the set is satisfiable, to be included.
- We add some variables when needed.
- The replacements depend on the size k of clause C_j.

SAT $\leq$ 3-SAT

- If $k = 1$, $C_j = z$, we add variables $\{y_{j1}, y_{j2}\}$ and clauses

$$C'_j = \{(z \vee y_{j1} \vee y_{j2}), (z \vee \bar{y}_{j1} \vee y_{j2}), (z \vee y_{j1} \vee \bar{y}_{j2}), (z \vee \bar{y}_{j1} \vee \bar{y}_{j2})\}.$$

Observe that C_j is satisfiable iff C'_j is satisfiable.

SAT $\leq$ 3-SAT

- If $k = 1$, $C_j = z$, we add variables $\{y_{j1}, y_{j2}\}$ and clauses

$$C'_j = \{(z \vee y_{j1} \vee y_{j2}), (z \vee \bar{y}_{j1} \vee y_{j2}), (z \vee y_{j1} \vee \bar{y}_{j2}), (z \vee \bar{y}_{j1} \vee \bar{y}_{j2})\}.$$

  Observe that C_j is satisfiable iff C'_j is satisfiable.

- If $k = 2$, $C_j = z_1 \vee z_2$, we add variable y_j and clauses

$$C'_j = \{(z_1 \vee z_2 \vee y), (z_1 \vee z_2 \vee \bar{y})\}.$$

  Again, C_j is satisfiable iff C'_j is satisfiable.

SAT $\leq$ 3-SAT

- If $k = 1$, $C_j = z$, we add variables $\{y_{j1}, y_{j2}\}$ and clauses

  $$C'_j = \{(z \vee y_{j1} \vee y_{j2}), (z \vee \bar{y}_{j1} \vee y_{j2}), (z \vee y_{j1} \vee \bar{y}_{j2}), (z \vee \bar{y}_{j1} \vee \bar{y}_{j2})\}.$$

  Observe that C_j is satisfiable iff C'_j is satisfiable.
- If $k = 2$, $C_j = z_1 \vee z_2$, we add variable y_j and clauses

  $$C'_j = \{(z_1 \vee z_2 \vee y), (z_1 \vee z_2 \vee \bar{y})\}.$$

  Again, C_j is satisfiable iff C'_j is satisfiable.
- If $k = 3$, we add $C_j = (z_1 \vee z_2 \vee z_3)$ to ϕ'.

SAT $\leq$ 3-SAT

- If $k > 3$, $C_j = (z_1 \vee z_2 \vee \cdots \vee z_k)$, add variables $\{y_{j1}, y_{j2}, \ldots, y_{jk-3}\}$ and the clauses

$$C'_j = \{(z_1 \vee z_2 \vee y_{j1}), (\bar{y}_{j1} \vee z_3 \vee y_{j2}), \ldots, (\bar{y}_{jk-3} \vee z_{k-1} \vee z_k)\}$$

- A satisfying assignment for C_j must made $T(z_i) = 1$ for at least one z_i. Then the assignment T' such that, $T(z_i) = 1$, $T'(y_{j1}) = \cdots = T'(y_{ji-2}) = 1$ and $T'(y_{ji-1}) = \cdots = T'(y_{jk-3}) = 0$ satisfies C'_j.
- On the other hand, if T' satisfies C'_j, there is z_i such that $T'(z_i) = 1$ (otherwise there would be a $y_{j\ell}$ such that $T'(y_{j\ell}) = 1 = T'(\bar{y}_{j\ell})$)

Example

Input to SAT: $\phi = (\bar{x_1}) \wedge (\bar{x_1}, \bar{x_2}) \wedge (\bar{x_1}, x_3, \bar{x_4}) \wedge (x_1, x_2, \bar{x_3}, x_4, x_5)$
$C_1' = (\bar{x_1}, y_{11}, y_{12}) \wedge (\bar{x_1}, \bar{y}_{11}, y_{12}) \wedge (\bar{x_1}, y_{11}, \bar{y}_{12}) \wedge (\bar{x_1}, \bar{y}_{11}, \bar{y}_{12})$
$C_2' = (\bar{x_1}, \bar{x_2}, y_2) \wedge (\bar{x_1}, \bar{x_2}, \bar{y}_2)$
$C_3' = (\bar{x_1}, x_3, \bar{x_4})$
$C_4' = (x_1, x_2, y_{41}) \wedge (\bar{y}_{41}, x_3, y_{42}) \wedge (\bar{y}_{42}, x_4, x_5)$

Then $f(\phi) = C_1' \wedge C_2' \wedge C_3' \wedge C_4'$
with $X' = \{x_1, x_2, x_3, x_4, x_5, y_{11}, y_{12}, y_2, y_3, y_{41}, y_{42}\}$

3-SAT is NP-complete

Theorem

3-SAT is NP-complete

Proof.

- The above construction can be done in polynomial time, therefor SAT $\leq_m^p$ 3-SAT, 3-SAT is NP-hard
- On the other hand 3-SAT is a subproblem of SAT, do 3-SAT$\in$ NP

□

The k-SAT problem

Theorem

For $k \geq 3$, k-SAT is NP-complete

Proof.

- We have just show that 3-SAT is NP-complete.
- Assume that ℓ-SAT is NP-complete.
- To reduce ℓ-SAT $\leq_m^p$ $(\ell + 1)$-SAT, for each clause C_j (with ℓ literals)
    - Add variable $\{y_j\}$
    - and add clauses
$$C_j' = \{(C_j \vee y_j), (C_j \vee \bar{y}_j)\}.$$
    - C_j is satisfiable iff C_j is satisfiable.

□

The 2-SAT problem

2-SAT: Given a Boolean formula ϕ, where each clause has exactly 2 literals, is ϕ satisfiable?

- A clause in a 2-SAT instance has the form $(y \vee z)$ for some literals y, z.

The 2-SAT problem

2-SAT: Given a Boolean formula ϕ, where each clause has exactly 2 literals, is ϕ satisfiable?

- A clause in a 2-SAT instance has the form $(y \vee z)$ for some literals y, z.
- Recall that $p \rightarrow q$ is equivalent to $\bar{p} \vee q$.

The 2-SAT problem

2-SAT: Given a Boolean formula ϕ, where each clause has exactly 2 literals, is ϕ satisfiable?

- A clause in a 2-SAT instance has the form $(y \vee z)$ for some literals y, z.
- Recall that $p \to q$ is equivalent to $\bar{p} \vee q$.
- So, $(y \vee z)$ is equivalent to $\bar{y} \to z$ and to $\bar{z} \to y$.

The 2-SAT problem

2-SAT: Given a Boolean formula ϕ, where each clause has exactly 2 literals, is ϕ satisfiable?

- A clause in a 2-SAT instance has the form $(y \vee z)$ for some literals y, z.
- Recall that $p \rightarrow q$ is equivalent to $\bar{p} \vee q$.
- So, $(y \vee z)$ is equivalent to $\bar{y} \rightarrow z$ and to $\bar{z} \rightarrow y$.
- A 2-SAT formula can be seen as a collection of implications.

The 2-SAT problem

2-SAT: Given a Boolean formula ϕ, where each clause has exactly 2 literals, is ϕ satisfiable?

- A clause in a 2-SAT instance has the form $(y \vee z)$ for some literals y, z.
- Recall that $p \rightarrow q$ is equivalent to $\bar{p} \vee q$.
- So, $(y \vee z)$ is equivalent to $\bar{y} \rightarrow z$ and to $\bar{z} \rightarrow y$.
- A 2-SAT formula can be seen as a collection of implications.
- To show that 2-SAT $\in$ P, we construct a kind of reduction to a problem related to the strongly connected components of a digraph.

Strongly connected components of a digraph

- A digraph G is strongly connected iff $\forall u, v \in V$, there is a path from u to v ($u \rightsquigarrow v$) and there is a path from v to u ($v \rightsquigarrow u$).
- We can determine if G is strongly connected in $O(n+m)$ time.
- When $\vec{G}$ not strongly connected, we can find its strongly connected components in $O(n+m)$ steps.

Strongly connected components of a digraph

- A digraph G is strongly connected iff $\forall u, v \in V$, there is a path from u to v ($u \rightsquigarrow v$) and there is a path from v to u ($v \rightsquigarrow u$).
- We can determine if G is strongly connected in $O(n + m)$ time.
- When $\vec{G}$ not strongly connected, we can find its strongly connected components in $O(n + m)$ steps.
- Recall, that by collapsing each strongly connected component of $\vec{G}$ to a node, and removing multiple edges and loops, the remaining digraph is acyclic.

2-SAT $\leq$ P

Let ϕ be a 2-SAT instance on a set X of n variables and with m clauses ($|\phi| = 2m$).

2-SAT$\leq$ P

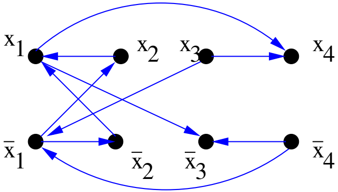
Let ϕ be a 2-SAT instance on a set X of n variables and with m clauses ($|\phi| = 2m$).
Define G_ϕ as follows:

- V: $2n$ nodes, two for each variable (x and $\bar{x}$).
- $\vec{E}$, has $2m$ edges, for each $C_i = (\alpha \vee \beta)$, add edges $\bar{\alpha} \to \beta$ and $\bar{\beta} \to \alpha$.

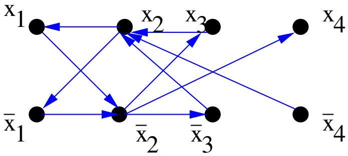
Notice that G_ϕ collects all implications in ϕ.

Examples

$(x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_1 \vee x_2)$
$\wedge(\bar{x}_3 \vee x_4) \wedge (\bar{x}_1 \vee x_4)$
which is satisfiable.



$\phi = (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_2)\wedge$
$(x_3 \vee x_2) \wedge (\bar{x}_3 \vee x_2) \wedge (x_2 \vee x_4)$
which is not satisfiable.



Correctness of the reduction

Exercise

ϕ is satisfiable iff no strongly connected component in G_ϕ contains nodes x and $\bar{x}$, for $x \in X$

Correctness of the reduction

Exercise

ϕ is satisfiable iff no strongly connected component in G_ϕ contains nodes x and $\bar{x}$, for $x \in X$

Theorem

2-Sat $\in P$

Proof.

To build G_ϕ takes $O(m)$.

The strongly connected components of G_ϕ can be obtained in $O(m+n)$. □