

# CS 455/555: Complexity theory

Stefan D. Bruda

Fall 2020

## ALTERNATIVE DEFINITION OF $\mathcal{NP}$ : CERTIFICATES



- $L \in \Sigma^*$ ;  $\Sigma^*$  is **polynomially balanced** iff there exists a polynomial  $p$  such that  $\forall x, y \in L : |y| \leq p(|x|)$
- $L \in \mathcal{NP}$  iff there exists a polynomially balanced language  $L'$  such that
  - 1  $L' \in \mathcal{P}$ , and
  - 2  $L = \{x \in \Sigma^* : \exists y \in \Sigma^* : x, y \in L'\}$
- $L'$  is the language of **succinct certificates** for  $L$  (every  $x \in L$  has a succinct certificate  $y$ )
- An  $\mathcal{NP}$  problem has solutions that are **easy to check**

## TIME MATTERS



- For some  $f : \mathbb{N} \rightarrow \mathbb{N}$ , a Turing machine  $M = (K, \Sigma, \Delta, s, \{h\})$  is  **$f$ -time bounded** iff for any  $w \in \Sigma^*$ : there is no configuration  $C$  such that  $(s, \#w\#) \vdash_M^{f(|w|)+1} C$
  - $M$  is **polynomially (time) bounded** iff  $M$  is  $p$ -time bounded for some polynomial  $p$
  - $L \subseteq \Sigma^*$  is **polynomially decidable** iff there is a **deterministic**, polynomially bounded Turing machine that decides  $L \Rightarrow$  **complexity class  $\mathcal{P}$** 
    - $\mathcal{P}$  is the class of exactly all the polynomially decidable languages
    - $\mathcal{P}$  is closed under complementation
    - There are recursive languages that are not in  $\mathcal{P}$  (page 277)
- $$E = \{\text{enc}(M)\#\text{enc}(w) : M \text{ accepts } w \text{ after at most } 2^{|w|} \text{ steps}\}$$
- $\mathcal{P}$  (as well as subsequent complexity classes) are based on **worst-case analysis**
  - **Complexity class  $\mathcal{NP}$** : the class of exactly all the languages decided by **nondeterministic**, polynomially bounded Turing machines
  - **Complexity class  $\mathcal{EXP}$** : exactly all the languages decided by exponentially-bounded, **deterministic** Turing machines
  - $\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{EXP}$

CS 455/555 (S. D. Bruda)

Fall 2020

1 / 19

## LANGUAGES? PROBLEMS?



- Given some computational problem that requires certain resource (time) bounds to solve, it is generally easy to find a language that requires the same resource bounds to decide
  - Sometime (but not always) finding an algorithm for deciding the language immediately implies an algorithm for solving the problem
- **Traveling salesman (TSP)**: Given  $n \geq 2$ , a matrix  $(d_{ij})_{1 \leq i, j \leq n}$  with  $d_{ij} > 0$  and  $d_{ii} = 0$ , find a permutation  $\pi$  of  $\{1, 2, \dots, n\}$  such that  $c(\pi)$ , the cost of  $\pi$  is minimal, where  $c(\pi) = d_{\pi_1 \pi_2} + d_{\pi_2 \pi_3} + \dots + d_{\pi_{n-1} \pi_n} + d_{\pi_n \pi_1}$ 
  - TSP the language (take 1):  $\{((d_{ij})_{1 \leq i, j \leq n}, B) : n \geq 2, B \geq 0, \text{ there exists a permutation } \pi \text{ such that } c(\pi) \leq B\}$
  - TSP the language (take 2), or the **Hamiltonian graphs**: Exactly all the graphs that have a (Hamiltonian) cycle that goes through all the vertices exactly once
  - **Note**: A cycle that uses all the **edges** exactly once is **Eulerian**; a graph  $G$  is Eulerian iff
    - 1 There is a path between any two vertices that are not isolated, and
    - 2 Every vertex has an in-degree equal to its out-degree

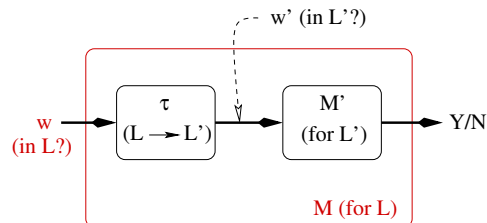


- **Clique:** Given an undirected graph  $G = (V, E)$ , find the maximal set  $C \subseteq V$  such that  $\forall v_i, v_j \in C : (v_i, v_j) \in E$  ( $C$  is a **clique** of  $G$ )
  - Clique, the language:  $\{(G = (V, E), K) : K \geq 2 : \text{there exists a clique } C \text{ of } V \text{ such that } |C| \geq K\}$
- **SAT:** Fix a set of **variables**  $X = \{x_1, x_2, \dots, x_n\}$  and let  $\bar{X} = \{\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n\}$ 
  - An element of  $X \cup \bar{X}$  is called a **literal**
  - A **formula** (or set/conjunction of **clauses**) is  $\alpha_1 \wedge \alpha_2 \wedge \dots \wedge \alpha_m$  where  $\alpha_i = x_{a_1} \vee x_{a_2} \vee \dots \vee x_{a_k}$ ,  $1 \leq i \leq m$ , and  $x_{a_j} \in X \cup \bar{X}$
  - An **interpretation** (or truth assignment) is a function  $I : X \rightarrow \{\top, \perp\}$
  - A formula  $F$  is **satisfiable** iff there exists an interpretation under which  $F$  evaluates to  $\top$ .
  - **SAT** =  $\{F : F \text{ is satisfiable}\}$
- **2-SAT, 3-SAT** are variants of SAT (with the number of literals in every clause restricted to a maximum of 2 and 3, respectively)

## REDUCTIONS, REVISITED



- The general idea of reductions:



- Reductions can be used in **proofs by contradiction**:
  - If  $L$  does not have property  $\mathbb{P}$  and reduction  $\tau$  from  $L$  to  $L'$  preserves  $\mathbb{P}$
  - Then  $L'$  does not have  $\mathbb{P}$
- Example: Turing reductions and undecidable problems

## 2-SAT



### Theorem

2-SAT  $\in \mathcal{P}$

- Algorithm **purge**( $F, x_i \in X$ ): Erase from  $F$   $\bar{x}_i$ , erase from  $F$  all the clauses that contain  $x_i$
- Algorithm **satisfy**( $F, X$ ):
  - 1 For every singleton clause  $x_i$ : Set  $I(x_i) = \top$ , **purge**( $F, x_i$ )
  - 2 For every singleton clause  $\bar{x}_i$ : Set  $I(x_i) = \perp$ , **purge**( $F, \bar{x}_i$ )
  - 3 If we have an empty clause then report  $F$  as unsatisfiable and stop
  - 4 Pick  $x_i \in X$ , set  $X$  to  $X \setminus \{x_i\}$ , and copy  $F$  into  $F'$
  - 5 Set  $I(x_i) = \top$ , **purge**( $F, x_i$ )
  - 6 If we have an empty clause, then
    - 1 Set  $I(x_i) = \perp$ , **purge**( $F', \bar{x}_i$ )
    - 2 If we have an empty clause then report  $F$  as unsatisfiable and stop
    - 3 Set  $F$  to  $F'$
  - 7 If  $x = \emptyset$  then report  $F$  as satisfiable and stop, otherwise repeat from Step 4

## POLYNOMIAL REDUCTIONS



- A function  $f : \Sigma^* \rightarrow \Sigma^*$  is **polynomially computable** iff there exists a polynomially time bounded, **deterministic** Turing machine that computes it
- Let  $L_1, L_2 \in \Sigma^*$ ; the function  $\tau : \Sigma^* \rightarrow \Sigma^*$  is a **polynomial reduction** if it is polynomially computable, and  $\forall x \in \Sigma^* : x \in L_1 \text{ iff } \tau(x) \in L_2$
- Polynomial reductions show that a problem is not harder to solve than another within a polynomial-time factor

### Lemma

$L_1$  is polynomially reducible to  $L_2$  and  $L_2 \in \mathcal{P}$  implies  $L_1 \in \mathcal{P}$

### Theorem

Polynomial reductions are closed under (functional) composition

- Direct, constructive proof



- A problem  $L$  is **NP-hard** iff for every language  $L' \in \mathcal{NP}$  there exists a polynomial reduction from  $L'$  to  $L$
- A problem  $L$  is **NP-complete** iff  $L$  is NP-hard and  $L \in \mathcal{NP}$

## Theorem

Let  $L$  be *some* NP-complete problem; then  $\mathcal{P} = \mathcal{NP}$  iff  $L \in \mathcal{P}$

- $\Rightarrow$ :  $L$  is NP-complete, so  $L \in \mathcal{NP}$ ; however,  $\mathcal{P} = \mathcal{NP}$  and so  $L \in \mathcal{P}$
- $\Leftarrow$ :  $L \in \mathcal{P}$ , so  $L$  is decided by a polynomially time bounded deterministic machine  $M$ 
  - For any  $L' \in \mathcal{NP}$  we have a polynomial reduction  $\tau$  from  $L$  to  $L'$ , decided by a polynomially time bounded, deterministic machine  $M_\tau$
  - Then  $L'$  is decided by the deterministic, polynomially time bounded machine  $M_\tau M$

# REDUCTION EXAMPLE (CONT'D)



- We have  $O(n^3)$  clauses with at most  $O(n)$  literals each
- Each clause may depend on  $G$  and  $n$  but nothing else
- The whole set is clearly polynomially computable, as desired
- Remains to prove that  $G$  has a Hamiltonian cycle iff  $\tau(G)$  is satisfiable
  - Suppose that some interpretation  $I$  satisfies  $\tau(G)$
  - Then for each  $i$  exactly one  $I(x_{ij})$  is  $\top$  and for each  $j$  exactly one  $I(x_{ij})$  is  $\top$  (because of 1-4)
  - This goes both ways
  - if
    - $\overline{x_{ij}} \vee \overline{x_{kj+1}}$  is true whenever  $(i, j) \notin G$
    - Whenever  $i = \pi_j$  and  $k = \pi_{j+1}$  we have  $I(x_{ij}) = \top$  and  $I(x_{kj+1}) = \top$
    - Therefore the clause  $\overline{x_{ij}} \vee \overline{x_{kj+1}}$  is false, so  $(i, k)$  **must** be an edge in  $G$
  - only if
    - Let  $\pi$  be a Hamiltonian cycle
    - We then set  $I(x_{ij}) = \top$  iff  $j = \pi_i$ , which makes  $\tau(G)$  true



- Reduction from Hamiltonian cycle to SAT
  - Graph  $G$  given as adjacency matrix:  $G = V \times V$ ,  $V = \{1, 2, \dots, n\}$
  - $G$  has a Hamiltonian cycle iff  $\tau(G)$  is satisfiable
- Variables:  $x_{ij}$ ,  $1 \leq i, j \leq n$ ;  $x_{ij} = \top$  iff vertex  $i$  is number  $j$  in the Hamiltonian cycle
- Clauses: need to specify that  $x_{ij}$  represent a permutation (or bijection) over  $V$ ; need then to specify that all the vertices in the cycle are actually connected
  - 1 at least one vertex is number  $j$   $\forall 1 \leq j \leq n : x_{1j} \vee x_{2j} \vee \dots \vee x_{nj}$
  - 2 no vertex can be in two places at once  $\forall 1 \leq i, j, k \leq n, j \neq k : \overline{x_{ij}} \vee \overline{x_{ik}}$
  - 3 every vertex must be in the cycle  $\forall 1 \leq i \leq n : x_{i1} \vee x_{i2} \vee \dots \vee x_{in}$
  - 4 a place in the cycle can only have one vertex  $\forall 1 \leq i, j, k \leq n, i \neq k : \overline{x_{ij}} \vee \overline{x_{kj}}$
  - 5 The permutation given by  $x_{ij}$  is a Hamiltonian cycle For all  $i$  and  $k$  such that  $(i, k) \notin G$  and assuming that  $x_{kn+1} = x_{k1}$ , we add  $\overline{x_{ij}} \vee \overline{x_{kj+1}}$

# NP-COMPLETENESS THEORY IN A NUTSHELL



- Are there NP-complete problems at all?
  - Yes, SAT is one (cf. Stephen Cook, 1971)
- The first is the hard one: we have to show that **every** problem in NP reduces to our problem
- Then in order to find other NP-complete problems all we need to do is to find problems such that **some** problem already known to be NP-complete reduces to them
  - This works because polynomial reductions are closed under composition = are transitive
- Then it is apparently easy to use the theorem stated earlier:
 

Let  $L$  be **some** NP-complete problem; then  $\mathcal{P} = \mathcal{NP}$  iff  $L \in \mathcal{P}$



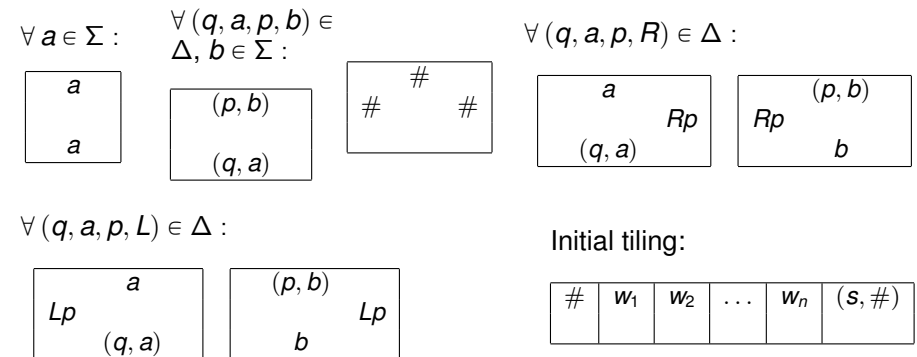
- **Tiling system:**  $\mathcal{D} = (D, d_0, H, V)$ 
  - $D$  is a finite set of tiles
  - $d_0 \in D$  is the initial corner tile
  - $H, V \in D \times D$  are the horizontal and vertical tiling restrictions
- **Tiling:**  $f : \mathbb{N}_s \times \mathbb{N}_s \rightarrow D$  such that
  - $f(0, 0) = d_0$
  - $\forall 0 \leq m < s, 0 \leq n < s - 1 : (f(m, n), f(m, n + 1)) \in V$
  - $\forall 0 \leq m < s - 1, 0 \leq n < s : (f(m, n), f(m + 1, n)) \in H$
- **The bounded tiling problem:**
  - Given a tiling system  $\mathcal{D}$ , a positive integer  $s$  and an initial tiling  $f_0 : \mathbb{N}_s \rightarrow D$
  - Find whether there exists a tiling function  $f$  that extends  $f_0$



- 1 **SAT  $\in \mathcal{NP}$** 
    - We nondeterministically guess an interpretation and we check that the interpretation satisfies the formula
    - Both of these take linear time
  - 2 **SAT is  $\mathcal{NP}$ -hard**
    - By reduction of bounded tiling to SAT
    - Consider variables  $x_{nmd}$  standing for “tile  $d$  is at position  $(n, m)$  in the tiling”
    - Construct clauses such that  $x_{nmd} = \top$  iff  $f(m, n) = d$
    - We first specify that we have a function:
      - each position has at least one tile:  $\forall 0 \leq m, n \leq s : x_{mnd_1} \vee x_{mnd_2} \vee \dots$
      - no more than one tile in a given position:  $\forall 0 \leq m, n \leq s, d \neq d' : \overline{x_{mnd}} \vee \overline{x_{mnd'}}$
    - Then we specify the restrictions  $H$  and  $V$ :
      - $(d, d') \in D^2 \setminus H \Rightarrow \overline{x_{mnd}} \vee \overline{x_{m+1nd'}}$        $(d, d') \in D^2 \setminus V \Rightarrow \overline{x_{mnd}} \vee \overline{x_{mn+1d'}}$
- In fact 3-SAT is also  $\mathcal{NP}$ -complete



- We need to find reductions from **all** problems in  $\mathcal{NP}$  to bounded tiling
  - The only thing in common to all the  $\mathcal{NP}$  problems is that each of them is decided by a nondeterministic, polynomially bounded Turing machine
  - We therefore find a reduction from an arbitrary such a machine to bounded tiling
- We find a tiling system such that each row in the tiling corresponds to one configuration of the given Turing machine



- To show that a problem is  $\mathcal{NP}$ -complete we need to show that
- The problem is in  $\mathcal{NP}$ 
  - Construct a Turing machine, or find succinct certificates
  - Usually quite straightforward
- The problem is  $\mathcal{NP}$ -hard
  - Exhibit a polynomial reduction from a known  $\mathcal{NP}$ -complete problem
  - Reduction can happen from any problem discussed in class and also from any problem discussed in Sections 7.2 and 7.3 (take those problems as solved exercises)
  - Make sure that you are comfortable with this way of thinking! There are numerous solved exercises to make you comfortable



- 3-SAT is  $\mathcal{NP}$ -complete
  - Hint: any clause  $x_1 \vee x_2 \vee \dots \vee x_n$  is logically equivalent with  $(x_1 \vee x_2 \vee x'_2) \wedge (\overline{x'_2} \vee x_3 \vee x'_3) \wedge (\overline{x'_3} \vee x_4 \vee x'_4) \wedge \dots \wedge (\overline{x'_{n-2}} \vee x_{n-1} \vee x_n)$
- **CLIQUE** =  $\{(G = (V, E), k) : k \geq 2 : \text{there exists a clique } C \text{ of } V, |C| = k\}$   
 Membership in  $\mathcal{NP}$  and 3-SAT being reducible to CLIQUE implies **CLIQUE is  $\mathcal{NP}$ -complete**
  - Start from  $\phi = C_1 \wedge C_2 \wedge \dots \wedge C_k$ , construct  $G = (V, E)$
  - Start with  $V = \emptyset$  and  $E = \emptyset$
  - For each clause  $C_r = l'_1 \vee l'_2 \vee l'_3$  add vertices  $v'_1, v'_2$ , and  $v'_3$  to  $V$
  - Add  $(v'_i, v'_j)$  to  $E$  whenever  $r \neq s$  and  $l'_i$  is not the negation of  $l'_j$  ( $l'_i$  is and  $l'_j$  are **consistent**)
  - Suppose that  $\phi$  is satisfiable; then:
    - The interpretation that makes  $\phi$  true makes at least one literal  $l'_i$  per clause true
    - The vertex  $v'_i$  is connected to **all** the other vertices  $v'_j$  that make the other clauses true (these are all consistent with each other)
    - So the vertices  $v'_i$  form a clique (of size  $k$ )
  - Suppose that  $G$  has a clique  $C$  of size  $k$ ; then:
    - $C$  contains exactly one vertex per clause
    - Assigning  $\top$  to every literal  $l'_i$  for which  $v'_i \in C$  is possible (all are consistent with each other)
    - The assignment makes  $\phi$  true so  $\phi$  is satisfiable



- **co- $\mathcal{NP}$**  is the complement of  $\mathcal{NP}$  ( $P \in \text{co-}\mathcal{NP}$  iff  $\overline{P} \in \mathcal{NP}$ )
  - Thought to be different from  $\mathcal{NP}$
  - $\mathcal{P} \subseteq \text{co-}\mathcal{NP}, \mathcal{P} \subseteq \mathcal{NP}$
  - If  $P \in \text{co-}\mathcal{NP}, P \in \mathcal{NP}$ , and  $P \in \mathcal{P}$  then  $P$  is suspected **not** to be  $\mathcal{NP}$ -complete
  - Example: the language of composite numbers (aka the integer factorization problem)
    - in  $\mathcal{NP}$  and also in  $\text{co-}\mathcal{NP}$
    - suspected outside  $\mathcal{P}$
    - suspected outside  $\mathcal{NP}$ -complete
- **co- $\mathcal{NP}$ -complete** problems also definable
  - integer factorization also suspected outside  $\text{co-}\mathcal{NP}$ -complete



- A vertex cover of  $G = (V, E)$  is a set  $V' \subseteq V$  such that  $(u, v) \in E \Rightarrow u \in V' \vee v \in V'$
- **VERTEX-COVER** =  $\{(G = (V, E), k) : G \text{ has a vertex cover of size } k\}$   
 Membership in  $\mathcal{NP}$  and CLIQUE being reducible to VERTEX-COVER implies **VERTEX-COVER is  $\mathcal{NP}$ -complete**
  - Start from  $(G = (V, E), k) \in \text{CLIQUE}$
  - Compute  $\overline{G} = (V, \overline{E})$  where  $\overline{E} = (V \times V) \setminus E$  (the **complement** of  $G$ )
  - Then  $(G, k) \in \text{CLIQUE}$  iff  $(\overline{G}, |V| - k) \in \text{VERTEX-COVER}$
  - Suppose that  $G$  has a clique  $C, |C| = k$ ; then:
    - $(u, v) \notin E$  means that  $u$  and  $v$  cannot be both in  $C$
    - That is,  $V \setminus C$  covers every edge  $(u, v) \in E$  that is, every vertex  $(u, v) \in \overline{E}$
    - Therefore  $V \setminus C$  is a vertex cover for  $\overline{G}$  (of size  $|V| - k$ )
  - Suppose that  $\overline{G}$  has a vertex cover  $V'$  with  $|V'| = |V| - k$ ; then:
    - $(u, v) \in \overline{E} \Rightarrow u \in V' \vee v \in V'$
    - Contrapositive:  $u \notin V' \wedge v \notin V' \Rightarrow (u, v) \notin \overline{E}$
    - That is,  $u \in V \setminus V' \wedge v \in V \setminus V' \Rightarrow (u, v) \in E$
    - So  $V \setminus V'$  is a clique of  $G$  (or size  $k$ )



- Several complexity classes:
  - $\mathcal{L} \subseteq \mathcal{NL} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PSPACE} = \mathcal{NPSPACE}$
  - $\mathcal{L}$  stands for logarithmic space and  $\mathcal{NL}$  for nondeterministic logarithmic space
- The only thing known:  **$\mathcal{NL} \neq \mathcal{PSPACE}$** 
  - So at least one of the inclusions in between must be strict
  - But we do not know which ones are strict or not
- Each inclusion has its own completeness theory, so we have  $\mathcal{P}$ -complete and  $\mathcal{PSPACE}$ -complete problems
  - **The reduction for each completeness theory comes from the inner class**
    - Indeed, if we go higher then **all** problems in the given class become complete!
  - That is,  $\mathcal{P}$ -complete problems are defined in terms of  $\mathcal{NL}$  reductions, whereas  $\mathcal{PSPACE}$ -complete problems are defined in terms of  $\mathcal{NP}$  reductions