Aalto University
School of Science

CS-E4530 Computational Complexity Theory

Lecture 7: NP-Complete Problems

Aalto University
School of Science
Department of Computer Science

Spring 2019

Agenda

- Proving NP-completeness
- Roadmap
- Compendium of fundamental problems
  - 3-SAT
  - 0/1 Integer Programming
  - Maximum Independent Set
  - k-colouring and Chromatic Number
  - Maximum Clique
  - Minimum Vertex Cover
  - Minimum Dominating Set
- Other NP-complete problems
- Decision versus search

NP-Complete Problems

- *Last lecture:*
  - We established that CNF-SAT is NP-complete
- *This lecture:*
  - Start proving that other natural problems are NP-complete
  - Build a *tree of reductions* step-by-step, starting from CNF-SAT

NP-Completeness via Reductions

Definition

Let $L_1, L_2 \subseteq \{0,1\}^*$ be languages. A polynomial-time *reduction* from L_1 to L_2 is a polynomial-time computable function $R\colon \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*$

$$x \in L_1 \text{ if and only if } R(x) \in L_2\,.$$

If there is a polynomial-time reduction from L_1 to L_2, we write $L_1 \leq_p L_2$.

NP-Completeness via Reductions

Theorem

Let $L_1, L_2 \in \{0,1\}^*$ *be languages. If* L_1 *is* NP*-hard and* $L_1 \leq_p L_2$*, then* L_2 *is* NP*-hard.*

- **General template for proving NP-completeness:**
  - Let L_2 be our problem of interest
  - Prove that L_2 is in NP
  - Pick a known NP-complete problem L_1
  - Construct a polynomial-time reduction from L_1 to L_2

NP-Completeness via Reductions

- **Building a reduction:**
  - *Step 1:* Define the transformation R from instances of L_1 to instances of L_2
  - *Step 2:* Prove the correctness of the reduction:
    - Let u be a certificate for $x \in L_1$. Show that we can use u to build a certificate for $R(x)$, showing that $R(x) \in L_2$.
    - Let u be a certificate for $R(x) \in L_2$. Show that we can use u to build a certificate for x, showing that $x \in L_1$.
  - *Step 3:* Prove that the reduction can be computed in polynomial time (*usually easy*)

NP-Completeness via Reductions

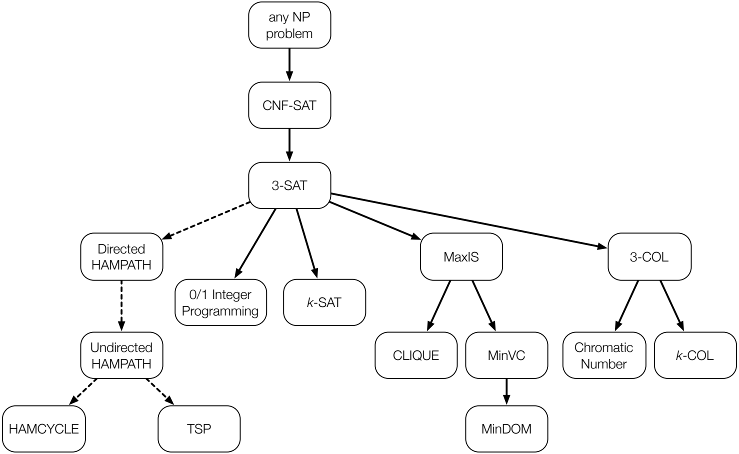
- **How to select the starting problem L_1 for reduction?**
  - Ideally, pick something as close as possible to L_2
  - Useful to know many NP-complete problems (*or find a list*)!

- **Common strategies**
  - *Restriction:* Show that L_2 contains a known NP-complete problem as a special case
  - *Local transformation:* Locally modify the instance structure to get from L_1 to L_2
  - *Gadget design and composition:* Build more complicated 'gadgets' to encode L_1 into L_2 instances
    - *Constraint satisfaction problems:* In many NP-complete problems one is given a finite set of variables and constraints between them, and the question is whether all the constraints can be satisfied simultaneously. (Consider e.g. SAT, COL.)
    - In such cases it is often helpful to first think how to map variables in L_1 to variables in L_2 (representation change), and then how to build gadgets in L_2 to enforce the constraints similarly as in L_1.

NP-Complete Problems

- **We will next consider some prototypical NP-complete problems**
  - Among *Karp's 21 NP-complete problems*
  - Richard Karp: *Reducibility Among Combinatorial Problems*, 1972
- **Thousands of more NP-complete problems known**
  - See e.g. Michael R. Garey and David S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1979

Roadmap



any NP problem
CNF-SAT
3-SAT
Directed HAMPATH
0/1 Integer Programming
k-SAT
MaxIS
3-COL
Undirected HAMPATH
CLIQUE
MinVC
Chromatic Number
k-COL
HAMCYCLE
TSP
MinDOM

CNF-SAT and k-SAT

Definition (CNF-SAT)

- **Instance:** A CNF formula φ.
- **Question:** Is φ satisfiable?

Definition (k-SAT)

- **Instance:** A CNF formula φ such that clause in φ has at most k literals.
- **Question:** Is φ satisfiable?

- **2-SAT instance:** $(x_1 \vee x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$
- **3-SAT instance:** $(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$
- **4-SAT instance:** $(x_1 \vee x_2 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4)$

3-SAT: NP-hardness

Theorem

3-SAT is NP*-hard.*

- **Proof: by reduction from CNF-SAT:**
  - Replace each clause with more than than three literals with equivalent set of three-literal clauses
  - For each clause $C = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_k$ in CNF-SAT instance φ, add $k-1$ new variables $y_1, \ldots, y_{k-1}$
  - Replace C with CNF

$$(\ell_1 \vee y_1) \wedge (\neg y_1 \vee \ell_2 \vee y_2) \wedge \cdots \wedge (\neg y_{k-1} \vee \ell_k)$$

3-SAT: NP-hardness

- **Proof: correctness of the reduction:**
  - Denote by $R(\varphi)$ the 3-CNF obtained from a CNF φ by above construction
  - If φ has a satisfying assignment, then $R(\varphi)$ has a satisfying assignment
  - If $R(\varphi)$ has a satisfying assignment, then φ has a satisfying assignment
  - $R(\varphi)$ can be constructed in polynomial time (in $|\llcorner\varphi\lrcorner|$)

k-SAT: NP-hardness

Theorem

k-SAT is NP*-hard for any $k \geq 3$.*

- **Proof:** 3-SAT is a special case of k-SAT

0/1 Integer Programming

0/1 Integer Programming

- **Instance:** A set of integral inequalities over variables $x_1, \dots, x_n$:

$$
\begin{aligned}
A_{1,1}x_1 + A_{1,2}x_2 + \cdots + A_{1,n}x_n &\geq C_1 \\
A_{2,1}x_1 + A_{2,2}x_2 + \cdots + A_{2,n}x_n &\geq C_2 \\
&\dots \\
A_{m,1}x_1 + A_{m,2}x_2 + \cdots + A_{m,n}x_n &\geq C_2
\end{aligned}
$$

- **Question:** Is there an assignment of values 0 and 1 to variables $x_1 \dots, x_n$ so that all inequalities are satisfied?

$$
\begin{array}{cccccccc}
x_1 & & +2x_3 & -x_4 & & \geq & 3 \\
& 3x_2 & -x_3 & & & \geq & 2 \\
x_1 & +x_2 & +x_3 & & +x_5 & \geq & 3 \\
x_1 & -2x_2 & -x_3 & -2x_4 & -x_5 & \geq & -2
\end{array}
$$

0/1 Integer Programming: NP-hardness

Theorem

0/1 Integer Programming is NP*-hard.*

- **Proof: by reduction from 3-SAT:**
  - Let φ be a 3-CNF formula with m clauses
  - Construct a system of inequalities $R(\varphi)$ over the 'same' variables
  - For each clause $C = \ell_1 \vee \ell_2 \vee \ell_3$, add an inequality

$$z_1 + z_2 + z_3 \geq 1\,,$$

    where $z_i = x_j$ if $\ell_i = x_j$, and $z_i = (1 - x_j)$ if $\ell_j = \neg x_j$
  - Transform inequalities to normal form

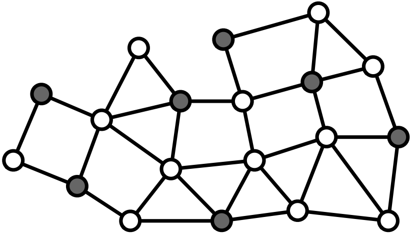
0/1 Integer Programming: NP-hardness

- **Proof: correctness of the reduction:**
  - If φ has a satisfying assignment, then the inequalities in $R(\varphi)$ are satisfied by the 'same' assignment
  - If all inequalities in $R(\varphi)$ can be satisfied, then the 'same' assignment satisfies φ
  - $R(\varphi)$ can be constructed in polynomial time (in $|\llcorner\varphi\lrcorner|$)

Maximum Independent Set

Maximum Independent Set (MaxIS)

- **Instance:** Graph $G = (V, E)$, an integer $k \geq 1$.
- **Question:** Is there a set of vertices I such that $|I| \geq k$ and for all $u, v \in I$, we have that $\{u, v\} \notin E$?



Maximum Independent Set: NP-hardness

Theorem

Maximum Independent Set is NP*-hard.*

- **Proof: by reduction from 3-SAT:**
  - Let φ be a 3-CNF formula with m clauses
  - For each clause C, there are 7 satisfying *partial assignments* to variables in C
  - Construct a graph $R(\varphi)$ by adding a clique on 7 vertices for each clause C
  - Identify each of the 7 vertices with satisfying partial assignments to variables in C
  - Add edges between inconsistent partial assignments
  - Set $k = m$

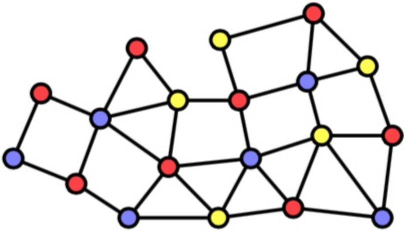
Maximum Independent Set: NP-hardness

- **Proof: correctness of the reduction:**
  - If φ has a satisfying assignment z, then $R(\varphi)$ has an independent set of size $k = m$
    - From each clique, pick the vertex representing the partial assignment consistent with z
    - z satisfying $\rightarrow$ can pick a vertex from each clique
  - If $R(\varphi)$ has an independent set I of size $k = m$, then φ has a satisfying assignment
    - Partial assignments corresponding to I are consistent with each other
  - $R(\varphi)$ can be constructed in polynomial time (in $|\llcorner\varphi\lrcorner|$)

3-Colouring

3-Colouring (3-COL)

- **Instance:** Graph $G = (V, E)$.
- **Question:** Is there a function $c\colon V \to \{1, 2, 3\}$ such that $c(u) \neq c(v)$ for all $\{u, v\} \in E$?



3-Colouring: NP-hardness

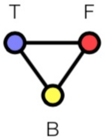
Theorem

3-Colouring is NP*-hard.*

- **Proof: by reduction from 3-SAT:**
  - Let φ be a 3-CNF formula with n variables and m clauses
  - Construct a graph $R(\varphi)$ that is 3-colourable if and only if φ if satisfiable
  - We need to build some *gadgets* for this
  - We will go over the ideas of the gadget constructions first, and then put everything together in the end

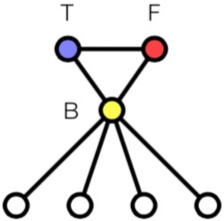
3-Colouring: NP-hardness

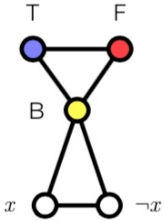
- **The Palette:**
  - A 3-colouring will use three colours
  - Assign one colour to stand for *true*
  - Assign one colour to stand for *false*
  - Third colour does not have a semantic meaning (*blank*)
- **Use a triangle to assign the semantics to colours**



3-Colouring: NP-hardness

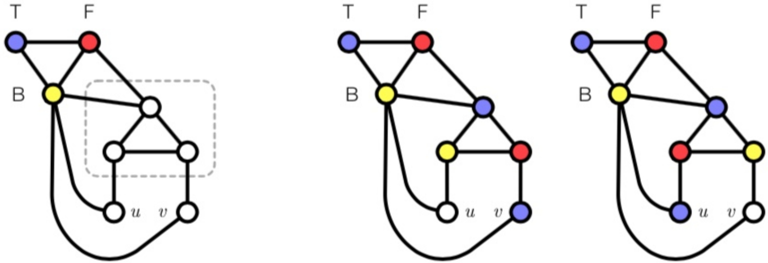
- **Representing 'variables':**
  - Vertex connected to the *blank* vertex on the palette will get colour 'true' or 'false'
  - Used to represent 'variables' that are true/false
  - Variable and its negation can be represented by two connected vertices





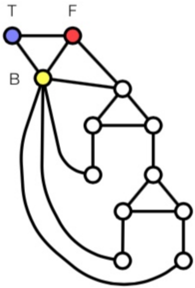
3-Colouring: NP-hardness

- **OR gadget for two variables:**
  - Let v and u be two 'variables' connected to *blank*
  - The following construction forces at least one of u and v to get the colour *true*



3-Colouring: NP-hardness

- **OR gadget for three variables:**
  - OR-gadgets can be composed to get bigger ORs
  - Gadget 3-colourable if and only if at least one of the *variable* vertices has colour 'true'



3-Colouring: NP-hardness

- **Reduction from 3-CNF φ to a graph $R(\varphi)$:**
  - $R(\varphi)$ has one *palette*
  - For each variable x_i, add two connected *variable* vertices corresponding to x_i and $\neg x_i$
  - For each clause $\ell_1 \vee \ell_2 \vee \ell_3$, add an OR gadget connecting vertices corresponding to ℓ_1, ℓ_2 and ℓ_3

3-Colouring: NP-hardness

- **φ satisfiable implies $R(\varphi)$ 3-colourable**
  - Colour palette arbitrarily
  - Colour *variables* according to a satisfying assignment z
  - z satisfying implies at least one *variable* in each or gadget is coloured with 'true', so OR gadgets can also be coloured

- **$R(\varphi)$ 3-colourable implies φ satisfiable**
  - Assign values to variable x_i depending on which *variable* vertex in $R(\varphi)$ is coloured 'true'
  - OR gadgets three-coloured implies at least one literal in each clause is true

3-Colouring: NP-hardness

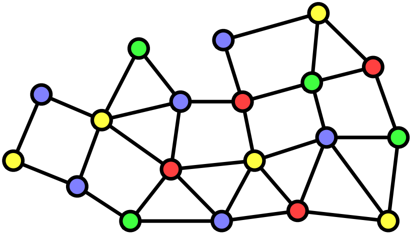
- **Reduction gives instances of polynomial size**
  - 3 vertices for the palette
  - $2n$ vertices for the variables
  - $6m$ vertices for the OR gadgets

- **Clearly computable in polynomial time**

k-Colouring

k-Colouring (k-COL)

- **Instance:** Graph $G = (V, E)$.
- **Question:** Is there a function $c\colon V \to \{1, 2, \ldots, k\}$ such that $c(u) \neq c(v)$ for all $\{u, v\} \in E$?



k-Colouring: NP-hardness

Theorem

k-Colouring is NP*-hard for any $k \geq 4$.*

- **Proof: Lecture 4**

Chromatic Number

Chromatic Number

- **Instance:** Graph $G = (V, E)$, an integer $k \geq 1$.
- **Question:** Is there a function $c\colon V \to \{1, 2, \ldots, k\}$ such that $c(u) \neq c(v)$ for all $\{u, v\} \in E$?

Theorem

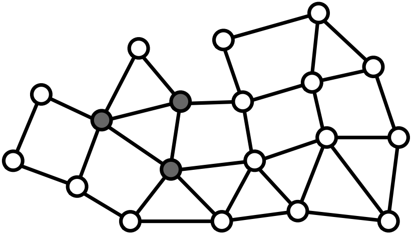
Chromatic Number is NP*-hard.*

- **Proof: contains 3-Colouring as a special case**

Maximum Clique

Maximum Clique (CLIQUE)

- **Instance:** Graph $G = (V,E)$, an integer $k \geq 1$.
- **Question:** Is there a set of vertices C such that $|C| \geq k$ and for all $u,v \in C$, we have that $\{u,v\} \in E$?



Maximum Clique: NP-hardness

Theorem

Maximum Clique is NP*-hard.*

- **Proof: by reduction from Maximum Independent Set**
- Set $U \subseteq V$ is an independent set in $G = (V, E)$ if and only if U is a clique in the complement graph $\overline{G}$
  - **Complement graph:** $\overline{G}$ has vertex set V, edge set

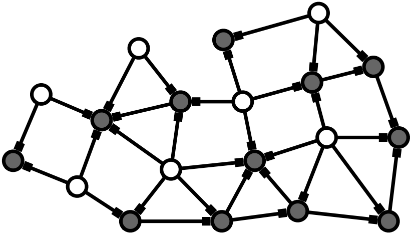
$$\overline{E} = \{\{u, v\} \subseteq V \colon \{u, v\} \notin E\}$$

  - **Reduction:** $R(G, k) = (\overline{G}, k)$

Minimum Vertex Cover

Minimum Vertex Cover (MinVC)

- **Instance:** Graph $G = (V, E)$, an integer $k \geq 1$.
- **Question:** Is there a set of vertices C such that $|C| \leq k$ and for all $\{u, v\} \in E$, either $v \in C$ or $u \in C$ (or both)?



Minimum Vertex Cover: NP-hardness

Theorem

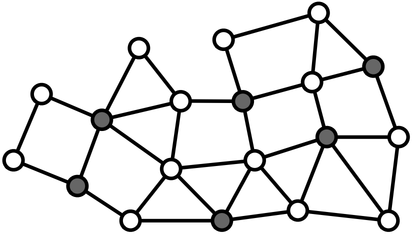
Minimum Vertex Cover is NP*-hard.*

- **Proof: by reduction from Maximum Independent Set**
- Set $U \subseteq V$ is an independent set if and only if $V \setminus U$ is a vertex cover
  - **Reduction:** $R(G,k) = (G, n-k)$

Minimum Dominating Set

Minimum Dominating Set (MinDS)

- **Instance:** Graph $G = (V, E)$, an integer $k \geq 1$.
- **Question:** Is there a set of vertices D such that $|D| \leq k$ and for all $v \in V$, either $v \in D$ or at least one of the neighbours of v is in D?



Minimum Dominating Set: NP-hardness

Theorem

Minimum Dominating Set is NP*-hard.*

- **Proof: by reduction from Minimum Vertex Cover:**
  - Let $G = (V, E)$ be the minimum vertex cover instance graph
  - Construct a new graph G' by adding a new vertex v_e for each edge $\{u, v\} \in E$
  - Add edges $\{u, v_e\}$ and $\{v, v_e\}$
  - $R(G, k) = (G', k)$

Minimum Dominating Set: NP-hardness

- **Proof: correctness of the reduction:**
  - If G has a vertex cover of size k, then G' has a dominating set of size k
  - If G' has a dominating set of size k, then G has a vertex cover of size k
    - We may assume that vertices v_e are not used in the dominating set
  - G' can be constructed in polynomial time (in $|\llcorner G \lrcorner|$)

Hamiltonian Path Problems

Directed Hamiltonian Path

- **Instance:** A directed graph $G = (V, E)$, vertices $s, t \in V$.
- **Question:** Does there exist a path from s to t that visits each vertex exactly once?

- **NP-hardness:** reduction from 3-SAT (complicated)

Undirected Hamiltonian Path

- **Instance:** An undirected graph $G = (V, E)$, vertices $s, t \in V$.
- **Question:** Does there exist a path from s to t that visits each vertex exactly once?

- **NP-hardness:** reduction from Directed Hamiltonian Path

Hamiltonian Cycle Problems

Hamiltonian Cycle

- **Instance:** An undirected/directed graph $G = (V, E)$.
- **Question:** Is there a cycle that visits each vertex exactly once?

- **NP-hardness:** reduction from Hamiltonian Path

Travelling Salesman Problem

- **Instance:** An undirected/directed graph $G = (V, E)$ with edge weights, integer W.
- **Question:** Is there a cycle that visits vertices exactly once with weight at most W?

- **NP-hardness:** contains Hamiltonian Cycle as a special case

Set Cover

Set Cover

- **Instance:** A finite set U, a family $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of subsets of U, an integer k.
- **Question:** Is there a subfamily $\mathcal{T} \subseteq \mathcal{S}$ such that $\mathcal{T}$ contains at most k sets from $\mathcal{S}$, and any element $u \in U$ is contained in at least one set $T \in \mathcal{T}$?

- **NP-hardness:** reduction from Vertex Cover

Exact Cover

Exact Cover

- **Instance:** A finite set U, a family $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of subsets of U, an integer k.
- **Question:** Is there a subfamily $\mathcal{T} \subseteq \mathcal{S}$ such that $\mathcal{T}$ contains at most k sets from $\mathcal{S}$, and any element $u \in U$ is contained in exactly one set $T \in \mathcal{T}$?

- **NP-hardness:** reduction from 3-SAT/3-Colouring

Subset Sum

Subset sum

- **Instance:** A list of integers $a_1, a_2, \ldots, a_n$ and an integer T.
- **Question:** Is there a subset of the input list that sums up to T?

- **NP-hardness:** reduction from 3-SAT (slightly technical)

Decision versus Search

- **We've defined** P **and** NP **in terms of decision problems**
  - In practice, we usually want to *find* a solution, not just know if it exists
- **Each** NP **problem has a natural search version:**
  - On input x, produce a certificate for $x \in L$ or decide that one does not exist
  - E.g. output a satisfying assignment, 3-colouring or an independent set of size k
  - Defined in terms of some fixed verifier

Decision versus Search

Theorem

Suppose $\mathsf{P} = \mathsf{NP}$*. Then for every language* $L \in \mathsf{NP}$ *and a verifier* M *for* L*, there is a polynomial-time Turing machine* M' *that computes a function* $f\colon \{0,1\}^* \to \{0,1\}^*$ *such that* $M(x, f(x)) = 1$ *for all* $x \in L$*.*

- **Proof:** suffices to prove the claim for CNF-SAT by Cook–Levin theorem
  - Try fixing $x_1 = 1$ and $x_1 = 0$, use decision algorithm to see if the formula remains satisfiable
  - Pick the alternative that retains satisfiability, repeat for $x_2, x_3, \ldots, x_n$

Lecture 7: Summary

- Existence of natural NP-complete problems
  - CNF-SAT, 3-SAT and Integer Programming
  - Chromatic Number and 3-Colouring
  - Maximum Independent Set and Maximum Clique
  - Minimum Vertex Cover
  - Minimum Dominating Set
  - Hamiltonian Paths/Cycles and TSP
  - Set Cover and Subset Sum