Algorithms for Data Processing
Lecture VII: Intractable Problems–Polynomial Reduction

Alessandro Artale

Free University of Bozen-Bolzano
Faculty of Computer Science
http://www.inf.unibz.it/~artale
artale@inf.unibz.it

2019/20 – First Semester
MSc in Computational Data Science — UNIBZ

Some material (text, figures) displayed in these slides is courtesy of:
Alberto Montresor, Werner Nutt, Kevin Wayne, Jon Kleinberg, Eva Tardos.

Classify problems according to computational requirements

Q. Which problems will we be able to solve in practice?

A working definition. Those with poly-time algorithms.


von Neumann (1953)


Nash (1955)


Gödel (1956)


Cobham (1964)


Edmonds (1965)


Rabin (1966)

Turing machine, word RAM, uniform circuits, ...

Theory. Definition is broad and robust.

constants tend to be small, e.g., $3n^2$

Practice. Poly-time algorithms scale to huge problems.

Hard Problems

- Having defined *efficient algorithms* those solvable in polynomial-time gives us the opportunity to prove mathematically that certain problems cannot be solved by polynomial-time, and hence by any "efficient" algorithm.
- There are hard problems for which we do not know of polynomial-time algorithms BUT we cannot prove that no polynomial-time algorithm exists.

Hard Problems

- Having defined *efficient algorithms* those solvable in polynomial-time gives us the opportunity to prove mathematically that certain problems cannot be solved by polynomial-time, and hence by any "efficient" algorithm.
- There are hard problems for which we do not know of polynomial-time algorithms BUT we cannot prove that no polynomial-time algorithm exists.
- A large class of problems in this gray area has been characterized, and it has been proved that they are equivalent in the following sense:
  - A polynomial-time algorithm for any one of them would imply the existence of a polynomial-time algorithm for all of them.

Hard Problems

- Having defined *efficient algorithms* those solvable in polynomial-time gives us the opportunity to prove mathematically that certain problems cannot be solved by polynomial-time, and hence by any "efficient" algorithm.
- There are hard problems for which we do not know of polynomial-time algorithms BUT we cannot prove that no polynomial-time algorithm exists.
- A large class of problems in this gray area has been characterized, and it has been proved that they are equivalent in the following sense:
  - A polynomial-time algorithm for any one of them would imply the existence of a polynomial-time algorithm for all of them.
- One such class is the class of NP-complete problems.

NP Problems

- There are thousands of NP-complete problems arising in numerous areas of computer science.
- The formulation of NP-completeness and the proof that such problems are equivalent is a powerful thing:
  - It says that all these open questions are really a single open question, a single type of complexity that we don't yet fully understand.
- From a pragmatic point of view NP-complete means computationally hard.
- Discovering that a problem is NP-complete provides a reason to stop searching for an efficient algorithm!

Polynomial-Time Reduction

To explore the space of computationally hard problems we need a mathematical characterization of when two problems are computationally equivalent.

- When problem Y is at least as hard as problem X?

Polynomial-Time Reduction

To explore the space of computationally hard problems we need a mathematical characterization of when two problems are computationally equivalent.

- When problem Y is at least as hard as problem X?

Reduction. Problem X polynomial-time reduces to problem Y (in symbols, $X \leq_P Y$) if arbitrary inputs of problem X can be solved using:

- Polynomial time to transform the input to X into an input for Y, plus
- Polynomial number of calls to the algorithm that solves problem Y.

Polynomial-Time Reduction

To explore the space of computationally hard problems we need a mathematical characterization of when two problems are computationally equivalent.

- When problem Y is at least as hard as problem X?

Reduction. Problem X polynomial-time reduces to problem Y (in symbols, $X \leq_P Y$) if arbitrary inputs of problem X can be solved using:

- Polynomial time to transform the input to X into an input for Y, plus
- Polynomial number of calls to the algorithm that solves problem Y.

At Least as Hard Problems. When problem Y is at least as hard as problem X?

- Whenever $X \leq_P Y$

Polynomial-Time Reduction/2

Equivalent Hard Problems. If both $X \leq_P Y$ and $Y \leq_P X$, we use the notation $X \equiv_P Y$.

Transitivity. If $X \leq_P Y$ and $Y \leq_P Z$, then $X \leq_P Z$.
Proof idea. Compose the two algorithms.

Polynomial-Time Reduction/3

Theorem 1. If $X \leq_P Y$ and Y can be solved in polynomial time, then X can be solved in polynomial time.

- Design polynomial-time algorithms for new problems: by reduction to a problem we already know how to solve in polynomial time.

Polynomial-Time Reduction/3

Theorem 1. If $X \leq_P Y$ and Y can be solved in polynomial time, then X can be solved in polynomial time.

- Design polynomial-time algorithms for new problems: by reduction to a problem we already know how to solve in polynomial time.

Corollary 2. If $X \leq_P Y$ and X cannot be solved in polynomial time, then Y cannot be solved in polynomial time.

- Establish intractability for new problems: by reduction of a problem we already know cannot be solved in polynomial time:
  - If we have a problem X that is known to be *hard* and we show that $X \leq_P Y$ then also Y is *hard*.

Independent Set

Independent Set Problem: Is a prototypical example of a hard problem: We don't know a polynomial-time algorithm for it, but we also don't know how to prove that none exists.

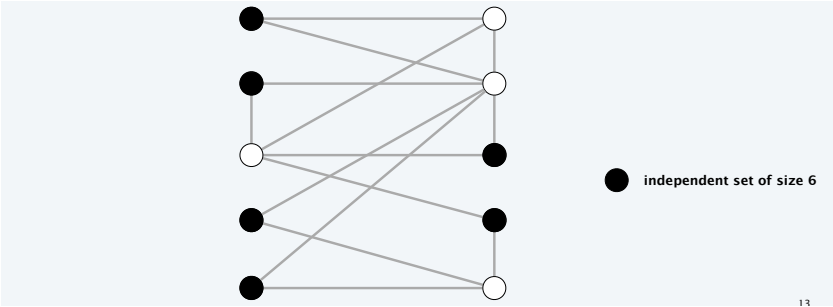
Definition. Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and no two vertices in S are adjacent?

Note: The above problem is formulated as a *yes*/*no* problem, we call such problems decision problems.

Independent Set/2

Definition. Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and no two vertices in S are adjacent?

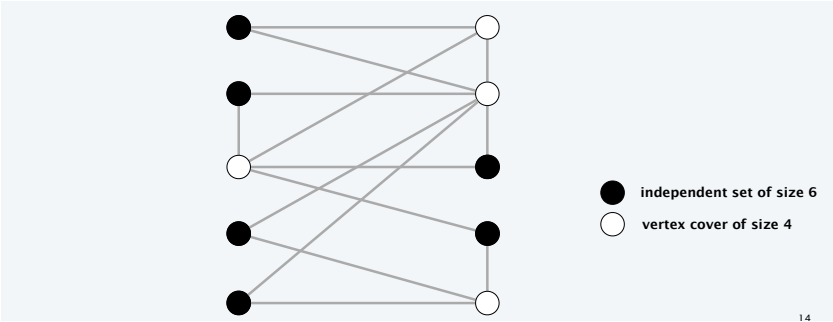
Ex. Is there an independent set of size ≥ 6?
Ex. Is there an independent set of size ≥ 7?



Vertex Cover

Definition. Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and each edge is incident to at least one vertex in S?

Ex. Is there a vertex cover of size ≤ 4?
Ex. Is there a vertex cover of size ≤ 3?



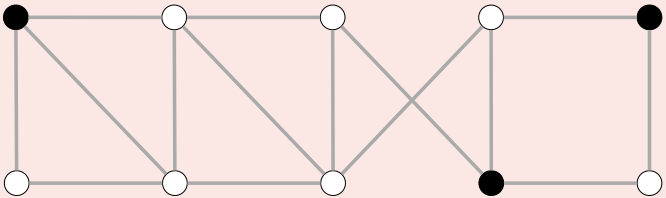
Consider the following graph G. Which are true?

A. The white vertices are a vertex cover of size 7.

B. The black vertices are an independent set of size 3.

C. Both A and B.

D. Neither A nor B.



Vertex Cover and Independent Set are Equivalent

Theorem. INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.

Proof. We show that S is an independent set of size k iff $V \setminus S$ is a vertex cover of size $n - k$.

($\Rightarrow$) Let S be any independent set of size k, then:

- $V \setminus S$ is of size $n - k$;
- Consider an arbitrary edge $(u, v) \in E$, then
- either $u \notin S$, or $v \notin S$, or both, then
- either $u \in V \setminus S$, or $v \in V \setminus S$, or $u, v \in V \setminus S$, then
- $V \setminus S$ is a vertex cover.
- Note: This also proves that VERTEX-COVER $\leq_P$ INDEPENDENT-SET.

Vertex Cover and Independent Set are Equivalent

Theorem. INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.

Proof. We show that S is an independent set of size k iff $V \setminus S$ is a vertex cover of size $n - k$.

($\Leftarrow$) Let $V \setminus S$ be a vertex cover of size $n - k$, then:

- S is of size k;
- Consider an arbitrary edge $(u, v) \in E$, then
- either $u \in V \setminus S$, or $v \in V \setminus S$, or $u, v \in V \setminus S$, then
- either $u \notin S$, or $v \notin S$, or both, then
- S is an independent set.
- Note: This also proves that INDEPENDENT-SET $\leq_P$ VERTEX-COVER.

Set Cover

- Vertex Cover can be viewed as a covering problem: The goal is to parsimoniously "cover" all the edges in the graph using as few vertices as possible.
- Vertex Cover is a coveting problem phrased specifically in the language of graph.
- There is a more general covering problem, Set Cover, to cover an arbitrary set of objects using a collection of sets.

Set Cover/2

Definition. Given a set U of n elements, a collection $S = \{S_1, \ldots, S_m\}$ of subsets of U, and a number k, does there exist a collection of at most k of these sets whose union is equal to U?

$U = \{ 1, 2, 3, 4, 5, 6, 7 \}$

$S_a = \{ 3, 7 \}$ $S_b = \{ 2, 4 \}$

$S_c = \{ 3, 4, 5, 6 \}$ $S_d = \{ 5 \}$

$S_e = \{ 1 \}$ $S_f = \{ 1, 2, 6, 7 \}$

$k = 2$

a set cover instance

Vertex Cover Reduces to Set Cover

Intuitively, Vertex Cover seems a special case of Set Cover:

- In Set Cover, we are trying to cover a set using arbitrary subsets;
- In Vertex Cover, we are trying to cover edges of a graph using sets of edges incident to a particular subset of vertices.

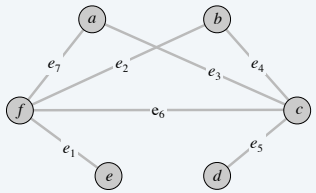
Vertex Cover Reduces to Set Cover/2

Theorem. VERTEX-COVER $\leq_P$ SET-COVER (SET-COVER is at least as hard as VERTEX-COVER.)

Proof. Given a VERTEX-COVER input instance $G = (V, E)$ and k, we construct a SET-COVER instance (U, S, k) that has a set cover of size k iff G has a vertex cover of size k.

Instance Construction. We need to cover the edges in E:

- $U = E$;
- For each vertex in the Vertex Cover Problem, we cover all the edges incident to it: For each vertex $v \in V$, we add a set $S_v = \{e \in E \mid e \textit{ incident to } v\}$.



vertex cover instance
(k = 2)

$U = \{1, 2, 3, 4, 5, 6, 7\}$

$S_a = \{3, 7\}$ $S_b = \{2, 4\}$

$S_c = \{3, 4, 5, 6\}$ $S_d = \{5\}$

$S_e = \{1\}$ $S_f = \{1, 2, 6, 7\}$

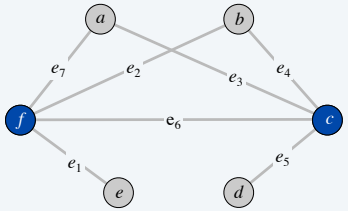
set cover instance
(k = 2)

Vertex Cover Reduces to Set Cover/3

Theorem. VERTEX-COVER $\leq_P$ SET-COVER.

($\Rightarrow$, Completeness) Let $G = (V, E)$ contains a Vertex Cover of size k, then, (U, S, k) that has a Set Cover of size k.

- Let $X \subseteq V$ be a vertex cover of size k in G, then, $Y = \{S_v \mid v \in X\}$ is a set cover of size k.



vertex cover instance
(k = 2)

$U = \{1, 2, 3, 4, 5, 6, 7\}$

$S_a = \{3, 7\}$ $S_b = \{2, 4\}$

$S_c = \{3, 4, 5, 6\}$ $S_d = \{5\}$

$S_e = \{1\}$ $S_f = \{1, 2, 6, 7\}$

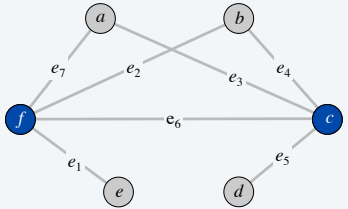
set cover instance
(k = 2)

Vertex Cover Reduces to Set Cover/4

Theorem. VERTEX-COVER $\leq_P$ SET-COVER.

($\Leftarrow$, Soundness) Let (U, S, k) contains a Set Cover of size k, then, $G = (V, E)$ contains a Vertex Cover of size k.

- Let $Y \subseteq S$ be a set cover of size k in (U, S, k), then, $X = \{v \mid S_v \in Y\}$ is a vertex cover of size k in G.



vertex cover instance
(k = 2)

$U = \{1, 2, 3, 4, 5, 6, 7\}$

$S_a = \{3, 7\}$ $S_b = \{2, 4\}$

$S_c = \{3, 4, 5, 6\}$ $S_d = \{5\}$

$S_e = \{1\}$ $S_f = \{1, 2, 6, 7\}$

set cover instance
(k = 2)

Satisfiability

Literal. A Boolean variable or its negation. x_i or $\overline{x_i}$

Clause. A disjunction of literals. $C_j = x_1 \vee \overline{x_2} \vee x_3$

Conjunctive normal form (CNF). A propositional formula Φ that is a conjunction of clauses. $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

SAT. Given a CNF formula Φ, does it have a satisfying truth assignment?
3-SAT. SAT where each clause contains exactly 3 literals (and each literal corresponds to a different variable).

$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4\right)$$

yes instance: x_1 = true, x_2 = true, x_3 = false, x_4 = false

Key application. Electronic design automation (EDA).

Syntax of Propositional Logic

Countable alphabet Σ of **atomic propositions**: $a, b, c, \ldots$.

Propositional formulas:

$$
\begin{array}{rcll}
\phi, \psi & \longrightarrow & a & \textit{atomic formula} \\
& | & \bot & \textit{false} \\
& | & \top & \textit{true} \\
& | & \neg\phi & \textit{negation} \\
& | & \phi \wedge \psi & \textit{conjunction} \\
& | & \phi \vee \psi & \textit{disjunction} \\
& | & \phi \rightarrow \psi & \textit{implication} \\
& | & \phi \leftrightarrow \psi & \textit{equivalence}
\end{array}
$$

- **Atom:** atomic formula
- **Literal:** (negated) atomic formula
- **Clause:** disjunction of literals

Semantics: Formally

A **truth value assignment** (or **interpretation**) of the atoms in Σ is a function μ:

$$v \colon \Sigma \to \{0, 1\}.$$

Note: Instead of $v(a)$ we also write a^v.

Definition: A formula ϕ is *satisfied* by an interpretation v ($v \models \phi$), or is *true under* v, if and only if:

$$\begin{array}{rcl}
 & v \models \top, \quad v \not\models \bot & \\
v \models a & \text{iff} & a^v = 1 \\
v \models \neg\phi & \text{iff} & v \not\models \phi \\
v \models \phi \wedge \psi & \text{iff} & v \models \phi \text{ and } v \models \psi \\
v \models \phi \vee \psi & \text{iff} & v \models \phi \text{ or } v \models \psi \\
v \models \phi \to \psi & \text{iff} & \text{if } v \models \phi, \text{ then } v \models \psi \\
v \models \phi \leftrightarrow \psi & \text{iff} & v \models \phi, \text{ if and only if } v \models \psi
\end{array}$$

Exercises

Let:

$$\nu: \begin{cases} a & \mapsto & 1 \\ b & \mapsto & 0 \\ c & \mapsto & 0 \\ d & \mapsto & 1 \end{cases}$$

Check the truth value under $\mathcal{I}$ of the following formulas:

- $b \rightarrow c \vee d$
- $c \vee d \rightarrow b$
- $b \leftrightarrow c \vee d$
- $((a \vee b) \leftrightarrow (c \vee d)) \wedge (\neg(a \wedge b) \vee (c \wedge \neg d))$

3-SAT Reduces to Independent Set

3-SAT. Let Φ be a set of clauses $C_1, \ldots, C_k$, each of length 3, over a set of variables $X = \{x_1, \ldots, x_n\}$, does there exist a satisfying truth assignment for Φ?

Theorem. `3-SAT` $\leq_P$ `INDEPENDENT-SET`.

Instance Construction: Intuitions.

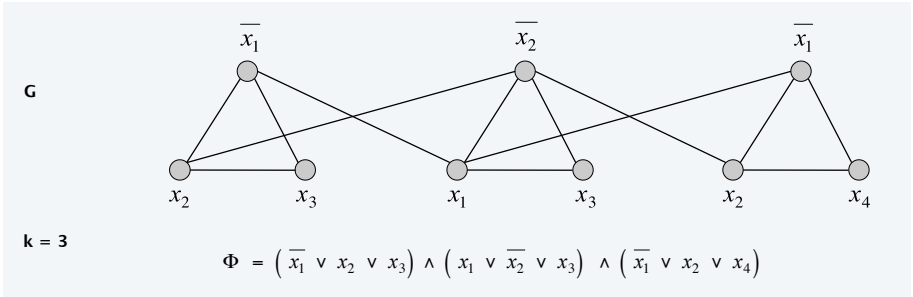
- Choose one literal from each clause, and then find a truth assignment that causes all these literals to evaluate to 1, thereby satisfying all clauses.
- Conflicting Literals. Two literals conflict if one is equal to a variable x_i and the other is equal to its negation $\overline{x_i}$.
- If we avoid choosing conflicting literals we can find a truth assignment that makes the selected literals from each clause evaluate to 1.

3-SAT Reduces to Independent Set/2

Theorem. 3-SAT $\leq_P$ INDEPENDENT-SET (INDEPENDENT-SET is at least as hard as 3-SAT.)

Instance Construction.

- Construct G containing 3 nodes for each clause in Φ, one for each literal;
- Connect 3 literals in a clause in a triangle;
- Connect literal to each of its negations (conflicting literals).

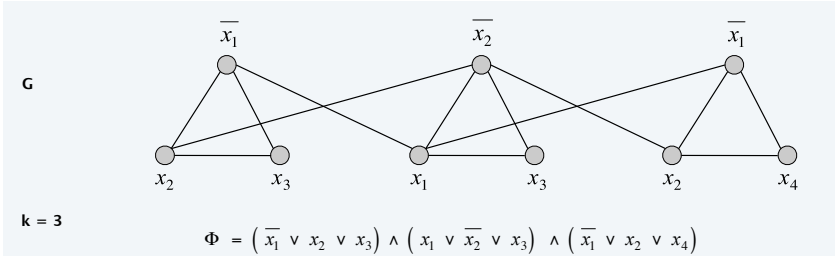


3-SAT Reduces to Independent Set/3

Theorem. 3-SAT $\leq_P$ INDEPENDENT-SET.

($\Rightarrow$, Completeness) Let Φ be satisfiable, then G contains an independent set of size $k = |\Phi|$.

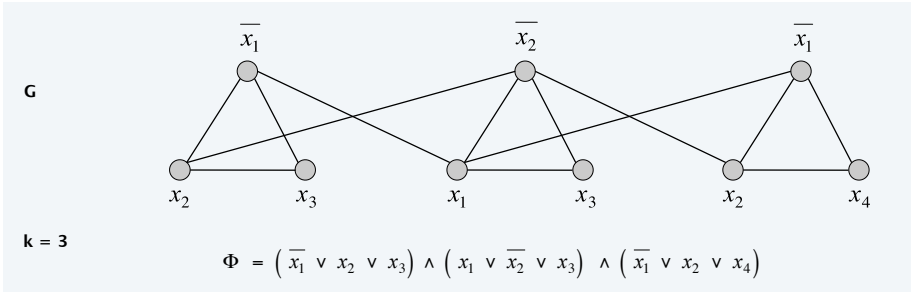
- Consider any satisfying assignment for Φ;
- For each triangle, select one node labeled by a true literal;
- This is an independent set, S, of size $k = |\Phi|$, since if there were an edge between two nodes $u, v \in S$, then the labels of u and v would have to conflict.



3-SAT Reduces to Independent Set/4

($\Leftarrow$, Soundness) Let G contains an independent set of size $k = \mid \Phi \mid$, then, Φ is satisfiable.

- Let S be an independent set of size k;
- S must contain exactly one node in each triangle;
- Set these literals to true (and remaining literals consistently): This is an assignment satisfying Φ, indeed, exactly one of x_i or $\overline{x_i}$ may appear as a label of a node in S, for otherwise there would be an edge between these two nodes.



Karp's 20 poly-time reductions from satisfiability

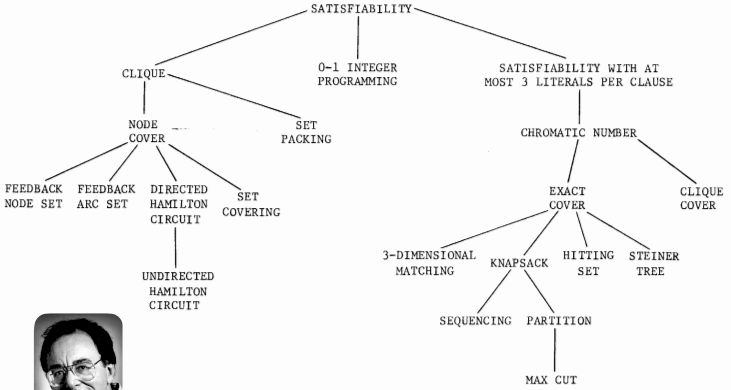


FIGURE 1 - Complete Problems

RICHARD M. KARP

Dick Karp (1972)
1985 Turing Award

Thank You!