

Design and Analysis of Algorithms (XIV)

Reductions on Various Problems

Guoqiang Li School of Computer Science



Poly-Time Reductions

◆□ → ◆□ → ◆ 三 → ◆ 三 → ○ へ ○ 2/63



Algorithm design patterns.

SHANGHAI JIAO TONG UNIVERSITY

Algorithm design patterns.

- Divide and conquer.
- Dynamic programming.
- Greedy.
- Duality.
- Reductions.
- Local search.
- Approximation.
- Randomization.

SHANGHAI JIAO TONG UNIVERSITY

Algorithm design patterns.

- Divide and conquer.
- Dynamic programming.
- Greedy.
- Duality.
- Reductions.
- Local search.
- Approximation.
- Randomization.

Algorithm design antipatterns.

• NP-completeness.



Algorithm design patterns.

- Divide and conquer.
- Dynamic programming.
- Greedy.
- Duality.
- Reductions.
- Local search.
- Approximation.
- Randomization.

Algorithm design antipatterns.

• NP-completeness.

 $O(n^k)$ algorithm unlikely.



Algorithm design patterns.

- Divide and conquer.
- Dynamic programming.
- Greedy.
- Duality.
- Reductions.
- Local search.
- Approximation.
- Randomization.

Algorithm design antipatterns.

• NP-completeness.

- $O(n^k)$ algorithm unlikely.
- PSPACE-completeness



Algorithm design patterns.

- Divide and conquer.
- Dynamic programming.
- Greedy.
- Duality.
- Reductions.
- Local search.
- Approximation.
- Randomization.

Algorithm design antipatterns.

- NP-completeness.
- PSPACE-completeness

 $O(n^k)$ algorithm unlikely. $O(n^k)$ certification algorithm unlikely.



Algorithm design patterns.

- Divide and conquer.
- Dynamic programming.
- Greedy.
- Duality.
- Reductions.
- Local search.
- Approximation.
- Randomization.

Algorithm design antipatterns.

- NP-completeness.
- PSPACE-completeness
- Undecidability

 $O(n^k)$ algorithm unlikely. $O(n^k)$ certification algorithm unlikely.



Algorithm design patterns.

- Divide and conquer.
- Dynamic programming.
- Greedy.
- Duality.
- Reductions.
- Local search.
- Approximation.
- Randomization.

Algorithm design antipatterns.

- NP-completeness.
- PSPACE-completeness
- Undecidability

 ${\cal O}(n^k)$ algorithm unlikely. ${\cal O}(n^k) \mbox{ certification algorithm unlikely.}$ No algorithm possible.



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



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)

SHANGHAI JIAO TONG UNIVERSITY

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

A working definition. Those with poly-time algorithms.





Nash

(1955)

Gödel (1956)





(1964)

Edmonds (1965)



Rabin (1966)

Theory. Definition is broad and robust.

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

Shanghai Jiao Tong University

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

A working definition. Those with poly-time algorithms.



von Neumann (1953)

Nash (1955)



Cobham



Edmonds (1965)



Rabin (1966)

Theory. Definition is broad and robust.

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

Practice. Poly-time algorithms scale to huge problems.



- Q. Which problems will we be able to solve in practice?
- A working definition. Those with poly-time algorithms.

yes	probably no
shortest path	longest path
min cut	max cut
2-satisfiability	3-satisfiability
planar 4-colorability	planar 3-colorability
bipartite vertex cover	vertex cover
2D-matching	3D-matching
primality testing	factoring
linear programming	integer linear programming

Classify problems



Requirement. Classify problems according to those that can be solved in polynomial time and those that cannot.

Classify problems



Requirement. Classify problems according to those that can be solved in polynomial time and those that cannot.

Provably requires exponential time.

- Given a constant-size program, does it halt in at most k steps?
- Given a board position in an *n*-by-*n* generalization of checkers, can black guarantee a win?





Classify problems



Requirement. Classify problems according to those that can be solved in polynomial time and those that cannot.

Provably requires exponential time.

- Given a constant-size program, does it halt in at most k steps?
- Given a board position in an *n*-by-*n* generalization of checkers, can black guarantee a win?





Frustrating news. Huge number of fundamental problems have defied classification for decades.



Suppose we could solve problem Y in polynomial time. What else could we solve in polynomial time?



Suppose we could solve problem Y in polynomial time. What else could we solve in polynomial time?

Reduction. Problem X polynomial-time (Cook) reduces to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem Y.





Suppose we could solve problem Y in polynomial time. What else could we solve in polynomial time?

Reduction. Problem X polynomial-time (Cook) reduces to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem Y.

Notation. $X \leq_{\mathbf{P}} Y$.

Note. We pay for time to write down instances of Y sent to oracle \Rightarrow instances of Y must be of polynomial size.



Suppose we could solve problem Y in polynomial time. What else could we solve in polynomial time?

Reduction. Problem X polynomial-time (Cook) reduces to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem Y.

Notation. $X \leq_{\mathrm{P}} Y$.

Note. We pay for time to write down instances of Y sent to oracle \Rightarrow instances of Y must be of polynomial size.

Novice mistake. Confusing $X \leq_{\mathbf{P}} Y$ with $Y \leq_{\mathbf{P}} X$.

Quiz



Suppose that $X \leq_{\mathbf{P}} Y$. Which of the following can we infer?

- **(A)** If X can be solved in polynomial time, then so can Y.
- **3** X can be solved in poly time iff Y can be solved in poly time.
- **(a)** If X cannot be solved in polynomial time, then neither can Y.
- **)** If Y cannot be solved in polynomial time, then neither can X.

Quiz



Which of the following poly-time reductions are known?

- FIND-MAX-FLOW \leq_P FIND-MIN-CUT.
- **③** FIND-MIN-CUT \leq_{P} FIND-MAX-FLOW.
- Generation Both A and B.
- Neither A nor B.



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



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

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

Establish equivalence. If both $X \leq_P Y$ and $Y \leq_P X$, we use notation $X \equiv_P Y$. In this case, X can be solved in polynomial time iff Y can be.



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

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

Establish equivalence. If both $X \leq_P Y$ and $Y \leq_P X$, we use notation $X \equiv_P Y$. In this case, X can be solved in polynomial time iff Y can be.

Evaluate (unknown) problems To evaluate the complexity of an unknown problem.



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

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

Establish equivalence. If both $X \leq_P Y$ and $Y \leq_P X$, we use notation $X \equiv_P Y$. In this case, X can be solved in polynomial time iff Y can be.

Evaluate (unknown) problems To evaluate the complexity of an unknown problem.

Bottom line. Reductions classify problems according to relative difficulty.

Packing and Covering Problems

Independent set



INDEPENDENT SET. Given a graph G = (V, E) and an integer k, is there a subset of k (or more) vertices such that no two are adjacent?

Independent set



INDEPENDENT SET. Given a graph G = (V, E) and an integer k, is there a subset of k (or more) vertices such that no two are adjacent?

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



Vertex cover



Vertex Cover. Given a graph G = (V, E) and an integer k, is there a subset of k (or fewer) vertices such that each edge is incident to at least one vertex in the subset?

Vertex cover



Vertex Cover. Given a graph G = (V, E) and an integer k, is there a subset of k (or fewer) vertices such that each edge is incident to at least one vertex in the subset?

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







Consider the following graph G. Which are true?

- ♦ The white vertices are a vertex cover of size 7.
- 3 The black vertices are an independent set of size 3.
- G Both A and B.
- Neither A nor B.



Vertex cover and independent set reduce to one another



Theorem

Independent Set \equiv_P Vertex Cover.



Vertex cover and independent set reduce to one another



Theorem

Independent Set \equiv_P Vertex Cover.

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




Theorem

Independent Set \equiv_P Vertex Cover.

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



Theorem

Independent Set \equiv_P Vertex Cover.

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

\Rightarrow

- Let S be any independent set of size k.
- V-S is of size n-k.
- Consider an arbitrary edge $(u, v) \in E$.



Theorem

Independent Set \equiv_P Vertex Cover.

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

\Rightarrow

- Let S be any independent set of size k.
- V-S is of size n-k.
- Consider an arbitrary edge $(u, v) \in E$.

•

$$\begin{split} S \text{ independent} &\Rightarrow \mathsf{either} \; u \notin S, \; \mathsf{or} \; v \notin S, \; \mathsf{or} \; \mathsf{both}. \\ &\Rightarrow \mathsf{either} \; u \in V-S, \; \mathsf{or} \; v \in V-S, \; \mathsf{or} \; \mathsf{both}. \end{split}$$



Theorem

Independent Set \equiv_P Vertex Cover.

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

\Rightarrow

- Let S be any independent set of size k.
- V-S is of size n-k.
- Consider an arbitrary edge $(u, v) \in E$.
- ٠

S independent \Rightarrow either $u \notin S$, or $v \notin S$, or both. \Rightarrow either $u \in V - S$, or $v \in V - S$, or both.

• Thus, V - S covers (u, v).



Theorem

Independent Set \equiv_P Vertex Cover.

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





Theorem

Independent Set \equiv_P Vertex Cover.

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

⇐

- Let V S be any independent set of size n k.
- S is of size k.
- Consider an arbitrary edge $(u, v) \in E$.



Theorem

Independent Set \equiv_P Vertex Cover.

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

⇐

- Let V S be any independent set of size n k.
- *S* is of size *k*.
- Consider an arbitrary edge $(u, v) \in E$.

•

V-S is a vertex cover \Rightarrow either $u \in V-S$, or $v \in V-S$, or both. \Rightarrow either $u \notin S$, or $v \notin S$, or both.



Theorem

Independent Set \equiv_P Vertex Cover.

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

¢

٠

- Let V S be any independent set of size n k.
- *S* is of size *k*.
- Consider an arbitrary edge $(u, v) \in E$.

V-S is a vertex cover \Rightarrow either $u \in V-S$, or $v \in V-S$, or both. \Rightarrow either $u \notin S$, or $v \notin S$, or both.

• Thus, S is an independent set.

Quiz



CLIQUE. Given a graph G = (V, E) and an integer k, is there a subset of k (or more) vertices such that each of two are adjacent?

Set cover



SET COVER. Given a set U of elements, a collection S of subsets of U, and an integer k, are there $\leq k$ of these subsets whose union is equal to U?



SET COVER. Given a set U of elements, a collection S of subsets of U, and an integer k, are there $\leq k$ of these subsets whose union is equal to U?

Sample application.

- *m* available pieces of software.
- Set U of n capabilities that we would like our system to have.
- The i^{th} piece of software provides the set $S_i \subseteq U$ of capabilities.
- Goal: achieve all *n* capabilities using fewest pieces of software.

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

$$S_a = \{3, 7\}$$

$$S_b = \{2, 4\}$$

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

$$S_d = \{5\}$$

$$S_e = \{1\}$$

$$k = 2$$

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

Quiz



Given the universe $U = \{1, 2, 3, 4, 5, 6, 7\}$ and the following sets, which is the minimum size of a set cover?

A.	1
B.	2
0	3
0	None of the above.

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

$$S_a = \{1, 4, 6\}$$

$$S_c = \{1, 2, 3, 6\}$$

$$S_d = \{1, 3, 5, 7\}$$

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

$$S_f = \{3, 4, 5\}$$

SHANGHAI JIAO TONG UNIVERSITY

Theorem

VERTEX COVER \leq_P SET COVER.



Theorem

VERTEX COVER \leq_P SET COVER.

Proof.

◆□ ▶ < □ ▶ < Ξ ▶ < Ξ ▶ Ξ < つ Q ○ 22/63</p>



Theorem

VERTEX COVER \leq_P SET COVER.

Proof. Given a VERTEX COVER 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.

Shanghai Jiao Tong University

Theorem

VERTEX COVER \leq_P SET COVER.

Proof. Given a VERTEX COVER 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.

Construction.

- Universe U = E.
- Include one subset for each node $v \in V : S_v = \{e \in E : e \text{ incident to } v\}$.



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

$$S_a = \{3, 7\}$$

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

$$S_d = \{5\}$$

$$S_e = \{1\}$$

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



Lemma

G = (V, E) contains a vertex cover of size k iff (U, S, k) contains a set cover of size k.





Lemma

G = (V, E) contains a vertex cover of size k iff (U, S, k) contains a set cover of size k.

Proof. \Rightarrow

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



$U = \{1, 2, 3, 4, 5, 6, 7\}$	
$S_a = \{3, 7\}$	$S_b = \{2, 4\}$
$S_c = \{3, 4, 5, 6\}$	$S_d = \{5\}$
$S_e = \{1\}$	$S_f = \{1, 2, 6, 7\}$



Lemma

G = (V, E) contains a vertex cover of size k iff (U, S, k) contains a set cover of size k.

Proof. 🗧 🗧

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



$U = \{1, 2, 3, 4, 5, 6, 7\}$	
$S_a = \{3, 7\}$	$S_b = \{2, 4\}$
$S_c = \{3, 4, 5, 6\}$ $S_e = \{1\}$	$S_d = \{5\}$ $S_f = \{1, 2, 6, 7\}$

Satisfaction Problems

▲□▶ ▲圖▶ ▲ 콜▶ ▲ 콜▶ · 콜 · • ○ Q · · · 25/63



Literal. A Boolean variable or its negation: x_i or \overline{x}_i .



Literal. A Boolean variable or its negation: x_i or \overline{x}_i .

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



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): $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$



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): $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

SAT. Given a CNF formula Φ , does it have a satisfying truth assignment?



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): $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

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).



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): $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

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).

$$\Phi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4)$$



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): $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

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).

$$\Phi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4)$$

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

Key application. Electronic design automation (EDA).

Satisfiability is hard



Scientific hypothesis. There does not exists a poly-time algorithm for 3-SAT.

Satisfiability is hard



Scientific hypothesis. There does not exists a poly-time algorithm for 3-SAT.

P vs. NP This hypothesis is equivalent to $\mathbf{P} \neq \mathbf{NP}$ conjecture.

Satisfiability is hard



Scientific hypothesis. There does not exists a poly-time algorithm for 3-SAT.

P vs. NP This hypothesis is equivalent to $\mathbf{P} \neq \mathbf{NP}$ conjecture.



Donald J. Trump 🤣 @realDonaldTrump



Computer Scientists have so much funding and time and can't even figure out the boolean satisfiability problem. SAT!

RETWEETS 16,936	LIKES 50,195	
6:31 AM - 1	17 Apr 2017	
🛧 20К	17 17K	♥ 50K

https://www.facebook.com/pg/npcompleteteens





This is an interesting and common kind of reduction, from a problem to a special case of itself.



This is an interesting and common kind of reduction, from a problem to a special case of itself.

Given an instance I of SAT, use exactly the same instance for 3SAT,



This is an interesting and common kind of reduction, from a problem to a special case of itself.

Given an instance I of SAT, use exactly the same instance for 3SAT, except that any clause with more than three literals,

$$(a_1 \lor a_2 \lor \ldots \lor a_k)$$

is replaced by a set of clauses,

```
(a_1 \vee a_2 \vee y_1)(\overline{y_1} \vee a_3 \vee y_2)(\overline{y_2} \vee a_4 \vee y_3) \dots (\overline{y_{k-3}} \vee a_{k-1} \vee a_k)
```

where the y_i 's are new variables.



This is an interesting and common kind of reduction, from a problem to a special case of itself.

Given an instance I of SAT, use exactly the same instance for 3SAT, except that any clause with more than three literals,

$$(a_1 \lor a_2 \lor \ldots \lor a_k)$$

is replaced by a set of clauses,

```
(a_1 \lor a_2 \lor y_1)(\overline{y_1} \lor a_3 \lor y_2)(\overline{y_2} \lor a_4 \lor y_3) \dots (\overline{y_{k-3}} \lor a_{k-1} \lor a_k)
```

where the y_i 's are new variables.

The reduction is in polynomial and I' is equivalent to I in terms of satisfiability.



$$\left\{\begin{array}{c} (a_1 \lor a_2 \lor \dots \lor a_k) \\ \text{is satisfied} \end{array}\right\} \Longleftrightarrow \left\{\begin{array}{c} \text{there is a setting of the } y_i\text{'s for which} \\ (a_1 \lor a_2 \lor y_1) (\overline{y}_1 \lor a_3 \lor y_2) \cdots (\overline{y}_{k-3} \lor a_{k-1} \lor a_k) \\ \text{are all satisfied} \end{array}\right\}$$



$$\left\{\begin{array}{c} (a_1 \vee a_2 \vee \dots \vee a_k) \\ \text{is satisfied} \end{array}\right\} \Longleftrightarrow \left\{\begin{array}{c} \text{there is a setting of the } y_i\text{'s for which} \\ (a_1 \vee a_2 \vee y_1) (\overline{y}_1 \vee a_3 \vee y_2) \cdots (\overline{y}_{k-3} \vee a_{k-1} \vee a_k) \\ \text{are all satisfied} \end{array}\right\}$$

Suppose that the clauses on the right are all satisfied. Then at least one of the literals a_1, \ldots, a_k must be true.

Otherwise y_1 would have to be true, which would in turn force y_2 to be true, and so on.

Conversely, if $(a_1 \lor a_2 \lor \ldots \lor a_k)$ is satisfied, then some a_i must be true. Set y_1, \ldots, y_{i-2} to true and the rest to false.


Theorem

 $3SAT \leq_P INDEPENDENT SET.$





Theorem

 $3SAT \leq_P INDEPENDENT SET.$

Proof.

▲□▶ ▲□▶ ▲ 王▶ ▲ 王▶ 王 · • ○ へ ○ 30/63



Theorem

 $3SAT \leq_P INDEPENDENT SET.$

Proof. Given an instance Φ of 3-SAT, we construct an instance (G, k) of INDEPENDENT SET that has an independent set of size $k = |\Phi|$ iff Φ is satisfiable.



Theorem

 $3SAT \leq_P INDEPENDENT SET.$

Proof. Given an instance Φ of 3-SAT, we construct an instance (G, k) of INDEPENDENT SET that has an independent set of size $k = |\Phi|$ iff Φ is satisfiable.

Construction.

- G contains 3 nodes for each clause, one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.





Theorem

 $3SAT \leq_P INDEPENDENT SET.$

Proof.

Shanghai Jiao Tong University

Theorem

 $3SAT \leq_P INDEPENDENT SET.$

Proof. \Rightarrow

Consider any satisfying assignment for Φ .

- Select one true literal from each clause/triangle.
- This is an independent set of size $k = |\Phi|$.



 $\Phi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4)$



Theorem

 $3SAT \leq_P INDEPENDENT SET.$

Proof.

▲□▶ ▲□▶ ▲ 三▶ ▲ 三▶ 三 ○ ○ ○ 32/63

Shanghai Jiao Tong University

Theorem

 $3SAT \leq_P INDEPENDENT SET.$

Proof. 🗧 ⇐

Let S be independent set of size k.

- S must contain exactly one node in each triangle.
- Set these literals to true and remaining literals consistently.
- All clauses in Φ are satisfied.



 $\Phi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4) \land (\overline{x_1} \lor x_2 \lor x_4) \land (\overline{x_1} \lor x_2 \lor x_3) \land (\overline{x_1} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_3) \land (\overline{x_1} \lor x_1) \lor (\overline{x_1} \lor x_2) \lor (\overline{x_1} \lor x_3) \land (\overline{x_1} \lor x_3) \lor (\overline$



Basic reduction strategies.

- Simple equivalence: Independent Set \equiv_P Vertex Cover
- Special case to general case: Vertex Cover \leq_P Set Cover.
- Encoding with gadgets: 3-SAT \leq_P Independent Set.



Basic reduction strategies.

- Simple equivalence: Independent Set \equiv_P Vertex Cover
- Special case to general case: Vertex Cover \leq_P Set Cover.
- Encoding with gadgets: 3-SAT \leq_P Independent Set.

Transitivity. If $X \leq_P Y$ and $Y \leq_P Z$, then $X \leq_P Z$.



Basic reduction strategies.

- Simple equivalence: Independent Set \equiv_P Vertex Cover
- Special case to general case: Vertex Cover \leq_P Set Cover.
- Encoding with gadgets: 3-SAT \leq_P Independent Set.

Transitivity. If $X \leq_P Y$ and $Y \leq_P Z$, then $X \leq_P Z$.

Proof idea. Compose the two algorithms.



Basic reduction strategies.

- Simple equivalence: Independent Set \equiv_P Vertex Cover
- Special case to general case: Vertex Cover \leq_P Set Cover.
- Encoding with gadgets: 3-SAT \leq_P Independent Set.

Transitivity. If $X \leq_P Y$ and $Y \leq_P Z$, then $X \leq_P Z$.

Proof idea. Compose the two algorithms.

Example. SAT \leq_P 3SAT \leq_P Independent Set \leq_P Vertex Cover \leq_P Set Cover.

Decision, Search and Optimization

▲□▶ ▲□▶ ▲ 三▶ ▲ 三 ▶ 三 ⑦ Q ℃ 34/63



Decision problem. Does there exist a vertex cover of size $\leq k$?





Decision problem. Does there exist a vertex cover of size $\leq k$?

Search problem. Find a vertex cover of size $\leq k$.



Decision problem. Does there exist a vertex cover of size $\leq k$?

Search problem. Find a vertex cover of size $\leq k$.

Optimization problem. Find a vertex cover of minimum size.



Decision problem. Does there exist a vertex cover of size $\leq k$?

Search problem. Find a vertex cover of size $\leq k$.

Optimization problem. Find a vertex cover of minimum size.

Goal. Show that all three problems poly-time reduce to one another.



VERTEX COVER. Does there exist a vertex cover of size $\leq k$?

FIND VERTEX COVER. Find a vertex cover of size $\leq k$.



VERTEX COVER. Does there exist a vertex cover of size $\leq k$?

FIND VERTEX COVER. Find a vertex cover of size $\leq k$.

Theorem. Vertex cover \equiv_P Find vertex cover.



VERTEX COVER. Does there exist a vertex cover of size $\leq k$?

FIND VERTEX COVER. Find a vertex cover of size $\leq k$.

Theorem. Vertex cover \equiv_P Find vertex cover.

Proof.



VERTEX COVER. Does there exist a vertex cover of size $\leq k$?

FIND VERTEX COVER. Find a vertex cover of size $\leq k$.

Theorem. Vertex cover \equiv_P Find vertex cover.

Proof.

 \leq_{P} . Decision problem is a special case of search problem.



VERTEX COVER. Does there exist a vertex cover of size $\leq k$?

FIND VERTEX COVER. Find a vertex cover of size $\leq k$.

Theorem. Vertex cover \equiv_P Find vertex cover.

Proof.

 \leq_{P} . Decision problem is a special case of search problem.

 \geq_{P} . To find a vertex cover of size $\leq k$:



VERTEX COVER. Does there exist a vertex cover of size $\leq k$?

FIND VERTEX COVER. Find a vertex cover of size $\leq k$.

Theorem. Vertex cover \equiv_P Find vertex cover.

Proof.

 \leq_P . Decision problem is a special case of search problem.

- \geq_P . To find a vertex cover of size $\leq k$:
 - Determine if there exists a vertex cover of size $\leq k$.
 - Find a vertex v such that G {v} has a vertex cover of size ≤ k 1. (any vertex in any vertex cover of size ≤ k will have this property)
 - Include *v* in the vertex cover.
 - Recursively find a vertex cover of size $\leq k 1$ in $G \{v\}$.



FIND VERTEX COVER. Find a vertex cover of size $\leq k$.

FIND MIN VERTEX COVER. Find a vertex cover of minimum size.



FIND VERTEX COVER. Find a vertex cover of size $\leq k$.

FIND MIN VERTEX COVER. Find a vertex cover of minimum size.

Theorem. Find vertex cover \equiv_P Find min vertex cover.



FIND VERTEX COVER. Find a vertex cover of size $\leq k$.

FIND MIN VERTEX COVER. Find a vertex cover of minimum size.

Theorem. Find vertex cover \equiv_P Find min vertex cover.

Proof.



FIND VERTEX COVER. Find a vertex cover of size $\leq k$.

FIND MIN VERTEX COVER. Find a vertex cover of minimum size.

Theorem. Find vertex cover \equiv_P Find min vertex cover.

Proof.

 \leq_P . Search problem is a special case of optimization problem.



FIND VERTEX COVER. Find a vertex cover of size $\leq k$.

FIND MIN VERTEX COVER. Find a vertex cover of minimum size.

Theorem. Find vertex cover \equiv_P Find min vertex cover.

Proof.

- \leq_{P} . Search problem is a special case of optimization problem.
- \geq_P . To find vertex cover of minimum size:



FIND VERTEX COVER. Find a vertex cover of size $\leq k$.

FIND MIN VERTEX COVER. Find a vertex cover of minimum size.

Theorem. Find vertex cover \equiv_P Find min vertex cover.

Proof.

- \leq_P . Search problem is a special case of optimization problem.
- \geq_{P} . To find vertex cover of minimum size:
 - Binary search (or linear search) for size k^* of min vertex cover.
 - Solve search problem for given k^* .

Sequencing Problems

▲□▶ ▲□▶ ▲ ■ ▶ ▲ ■ ▶ ● ■ ⑦ Q ○ 38/63

Hamilton cycle



HAMILTON CYCLE. Given an undirected graph G = (V, E), does there exist a cycle Γ that visits every node exactly once?







no



DIRECTED HAMILTON CYCLE. Given a directed graph G = (V, E), does there exist a directed cycle Γ that visits every node exactly once?

Theorem

DIRECTED HAMILTON CYCLE \leq_P HAMILTON CYCLE.



DIRECTED HAMILTON CYCLE. Given a directed graph G = (V, E), does there exist a directed cycle Γ that visits every node exactly once?

Theorem

DIRECTED HAMILTON CYCLE \leq_P HAMILTON CYCLE.

Proof. Given a directed graph G = (V, E), construct a graph G' with 3n nodes.





Lemma

G has a directed Hamilton cycle iff G' has a Hamilton cycle.



Lemma

G has a directed Hamilton cycle iff G' has a Hamilton cycle.

Proof.





Lemma

G has a directed Hamilton cycle iff G' has a Hamilton cycle.

Proof.

\Rightarrow

- Suppose G has a directed Hamilton cycle Γ .
- Then G' has an undirected Hamilton cycle (same order).
Directed Hamilton cycle reduces to Hamilton cycle



Lemma

G has a directed Hamilton cycle iff G' has a Hamilton cycle.

Proof.

\Rightarrow

- Suppose G has a directed Hamilton cycle Γ .
- Then G' has an undirected Hamilton cycle (same order).

\Leftarrow

- Suppose G' has an undirected Hamilton cycle Γ' .
- Γ' must visit nodes in G' using one of following two orders:
 ..., black, white, blue, black, white, blue, black, white, blue, ...
 ..., black, blue, white, black, blue, white, black, blue, white, ...
- Black nodes in Γ' comprise either a directed Hamilton cycle Γ in G, or reverse of one.



Theorem

3-SAT \leq_P Directed Hamilton cycle.



Theorem

3-SAT \leq_P Directed Hamilton cycle.

Proof.

▲□▶ ▲□▶ ▲ ミ▶ ▲ ミ ▶ ミ • つ < ○ 42/63</p>



Theorem

3-SAT \leq_P Directed Hamilton cycle.

Proof.

Given an instance Φ of 3-SAT, we construct an instance G of Directed Hamilton cycle that has a Hamilton cycle iff Φ is satisfiable.



Theorem

3-SAT \leq_P Directed Hamilton cycle.

Proof.

Given an instance Φ of 3-SAT, we construct an instance G of Directed Hamilton cycle that has a Hamilton cycle iff Φ is satisfiable.

Construction overview. Let n denote the number of variables in Φ . We will construct a graph G that has 2^n Hamilton cycles, with each cycle corresponding to one of the 2^n possible truth assignments.



Construction. Given 3-SAT instance Φ with n variables x_i and k clauses.

- Construct G to have 2^n Hamilton cycles.
- Intuition: traverse path *i* from left to right \Leftrightarrow set variables $x_i =$ true





Which is truth assignment corresponding to Hamilton cycle below?

(a) $x_1 = true, x_2 = true, x_3 = true$ (b) $x_1 = false, x_2 = false, x_3 = true$

$$x_1 = true, x_2 = true, x_3 = false$$

 \bigcirc $x_1 = false, x_2 = false, x_3 = false$





Construction. Given 3-SAT instance Φ with *n* variables x_i and *k* clauses.

• For each clause: add a node and 2 edges per literal.





Construction. Given 3-SAT instance Φ with n variables x_i and k clauses.

• For each clause: add a node and 2 edges per literal.





Lemma

 Φ is satisfiable iff G has a Hamilton cycle.



Lemma

 Φ is satisfiable iff G has a Hamilton cycle.

Proof.

▲□▶ ▲□▶ ▲ 三▶ ▲ 三▶ 三 の Q ○ 47/63



Lemma

 Φ is satisfiable iff G has a Hamilton cycle.

Proof. \Rightarrow

- Suppose 3-SAT instance Φ has satisfying assignment x^* .
- Then, define Hamilton cycle Γ in G as follows:
 - if $x_i^* = true$, traverse row *i* from left to right.
 - if $x_i^* = false$, traverse row *i* from right to left.
 - for each clause C_j , there will be at least one row *i* in which we are going in "correct" direction to splice clause node C_j into cycle (and we splice in C_j exactly once)



Lemma

 Φ is satisfiable iff G has a Hamilton cycle.

Proof.





Lemma

 Φ is satisfiable iff G has a Hamilton cycle.

Proof. 🗧 ⇐

- Suppose G has a Hamilton cycle Γ .
- If Γ enters clause node C_j, it must depart on mate edge.
 - nodes immediately before and after C_i are connected by an edge $e \in E$.
 - removing C_j from cycle, and replacing it with edge e yields Hamilton cycle on $G \{C_j\}$.
- Continuing in this way, we are left with a Hamilton cycle Γ' in $G \{C_1, C_2, \ldots, C_k\}$.
- Set $x_i^* = true$ if Γ' traverses row *i* left-to-right; otherwise, set $x_i^* = false$.
- traversed in "correct" direction, and each clause is satisfied.

Graph Coloring

▲□▶ ▲□▶ ▲ 三▶ ▲ 三 ▶ 三 ⑦ Q ℃ 49/63

Home reading!

Numerical Problems



SUBSET SUM. Given n natural numbers w_1, \ldots, w_n and an integer W, is there a subset that adds up to exactly W?



SUBSET SUM. Given n natural numbers w_1, \ldots, w_n and an integer W, is there a subset that adds up to exactly W?

Example. { 215, 215, 275, 275, 355, 355, 420, 420, 580, 580, 655, 655 }, W = 1505.



SUBSET SUM. Given n natural numbers w_1, \ldots, w_n and an integer W, is there a subset that adds up to exactly W?

Example. { 215, 215, 275, 275, 355, 355, 420, 420, 580, 580, 655, 655 }, W = 1505. Yes. 215 + 355 + 355 + 580 = 1505.



SUBSET SUM. Given n natural numbers w_1, \ldots, w_n and an integer W, is there a subset that adds up to exactly W?

Example. { 215, 215, 275, 275, 355, 355, 420, 420, 580, 580, 655, 655 }, W = 1505. Yes. 215 + 355 + 355 + 580 = 1505.

Remark. With arithmetic problems, input integers are encoded in binary. Poly-time reduction must be polynomial in binary encoding.



Theorem

3-SAT \leq_P SUBSET SUM.

◆□ ▶ < □ ▶ < Ξ ▶ < Ξ ▶ Ξ · 𝔅 𝔅 53/63</p>



Theorem

3-SAT \leq_P Subset sum.

Proof. Given an instance Φ of 3-SAT, we construct an instance of Subset sum that has solution iff Φ is satisfiable.



Construction. Given 3-SAT instance Φ with n variables and k clauses, form 2n + 2k decimal integers, each having n + k digits:

- Include one digit for each variable x_i and one digit for each clause C_j .
- Include two numbers for each variable x_i .
- Include two numbers for each clause C_j .



- Sum of each x_i digit is 1;
- Sum of each C_j digit is 4.

Key property. No carries possible \Rightarrow each digit yields one equation.



dummies to get clause columns to sum to 4

	x_1	<i>x</i> ₂	<i>x</i> 3	<i>C</i> 1	<i>C</i> ₂	<i>C</i> ₃	
x_1	1	0	0	0	1	0	100,010
$\neg x_1$	1	0	0	1	0	1	100,101
x_2	0	1	0	1	0	0	10,100
$\neg x_2$	0	1	0	0	1	1	10,011
<i>x</i> 3	0	0	1	1	1	0	1,110
¬ <i>x</i> 3	0	0	1	0	0	1	1,001
(0	0	0	1	0	0	100
	0	0	0	2	0	0	200
J	0	0	0	0	1	0	10
Ì	0	0	0	0	2	0	20
	0	0	0	0	0	1	1
L	0	0	0	0	0	2	2
W	1	1	1	4	4	4	111,444



Lemma

 Φ is satisfiable iff there exists a subset that sums to W.



Lemma

 Φ is satisfiable iff there exists a subset that sums to W.

Proof.



Lemma

 Φ is satisfiable iff there exists a subset that sums to W.

Proof. \Rightarrow Suppose 3-SAT instance Φ has satisfying assignment x^* . If $x_i^* = true$, select integer in row x_i , otherwise, select integer in row $\neg x_i$.

- Each x_i digit sums to 1.
- Since Φ is satisfiable, each C_j digit sums to at least 1 from x_i and ¬x_i rows.
- Select dummy integers to make C_j digits sum to 4.



dummies to get clause columns to sum to 4

	<i>x</i> 1	<i>x</i> 2	ж3	<i>C</i> 1	<i>C</i> ₂	<i>C</i> ₃	
<i>x</i> 1	1	0	0	0	1	0	100,010
$\neg x_1$	1	0	0	1	0	1	100,101
<i>x</i> ₂	0	1	0	1	0	0	10,100
$\neg x_2$	0	1	0	0	1	1	10,011
х3	0	0	1	1	1	0	1,110
$\neg x_3$	0	0	1	0	0	1	1,001
Ċ	0	0	0	1	0	0	100
	0	0	0	2	0	0	200
J	0	0	0	0	1	0	10
Ĵ	0	0	0	0	2	0	20
	0	0	0	0	0	1	1
Ĺ	0	0	0	0	0	2	2
W	1	1	1	4	4	4	111,444



Lemma

 Φ is satisfiable iff there exists a subset that sums to W.

Proof.

▲□▶ ▲□▶ ▲豆▶ ▲豆▶ 豆 のへで 57/63



Lemma

 Φ is satisfiable iff there exists a subset that sums to W.

Proof. \Leftarrow Suppose there exists a subset S^* that sums to W. Digit x_i forces subset S^* to select either row x_i or row $\neg x_i$ (but not both). If row x_i selected, assign $x_i^* = true$; otherwise, assign $x_i^* = false$.

Digit C_j forces subset S^* to select at least one literal in clause.



dummies to get clause columns to sum to 4

<i>x</i> 1	0					
		0	0	1	0	100,010
¬.x1 1	0	0	1	0	1	100,101
<i>x</i> ₂ 0	1	0	1	0	0	10,100
$\neg x_2 = 0$	1	0	0	1	1	10,011
.x3 0	0	1	1	1	0	1,110
¬.x ₃ 0	0	1	0	0	1	1,001
0	0	0	1	0	0	100
0	0	0	2	0	0	200
0	0	0	0	1	0	10
ົງ 0	0	0	0	2	0	20
0	0	0	0	0	1	1
0	0	0	0	0	2	2
W = 1	1	1	4	4	4	111,444

Subset sum reduces to knapsack



Subset sum. Given n natural numbers w_1, \ldots, w_n and an integer W, is there a subset that adds up to exactly W?

Knapsack. Given a set of items X, weights $u_i \ge 0$, values $v_i \ge 0$, a weight limit U, and a target value V, is there a subset $S \subseteq X$ such that:

$$\sum_{i \in S} u_i \le U, \quad \sum_{i \in S} v_i \ge V$$

Subset sum reduces to knapsack



Subset sum. Given n natural numbers w_1, \ldots, w_n and an integer W, is there a subset that adds up to exactly W?

Knapsack. Given a set of items X, weights $u_i \ge 0$, values $v_i \ge 0$, a weight limit U, and a target value V, is there a subset $S \subseteq X$ such that:

$$\sum_{i \in S} u_i \le U, \quad \sum_{i \in S} v_i \ge V$$

Recall. O(nU) dynamic programming algorithm for knapsack.

Subset sum reduces to knapsack



Subset sum. Given n natural numbers w_1, \ldots, w_n and an integer W, is there a subset that adds up to exactly W?

Knapsack. Given a set of items X, weights $u_i \ge 0$, values $v_i \ge 0$, a weight limit U, and a target value V, is there a subset $S \subseteq X$ such that:

$$\sum_{i \in S} u_i \le U, \quad \sum_{i \in S} v_i \ge V$$

Recall. O(nU) dynamic programming algorithm for knapsack.

```
Challenge. Prove subset sum \leq_P Knapsack.
```

Quiz



Can you prove ILP is a difficult problem?

Poly-time reductions



constraint satisfaction



◆□▶ ◆□▶ ◆ ≧▶ ◆ ≧▶ ○ 差 の � 00/63

Karp's 20 poly-time reductions from satisfiability




Referred Materials

◆□ ▶ < □ ▶ < Ξ ▶ < Ξ ▶ Ξ < つへ ? 62/63</p>

Referred Materials



• Content of this lecture comes from Chapter 8 in [KT05].

