Algorithm II

8. Intractability I

WU Xiaokun 吴晓堃

xkun.wu [at] gmail

Polynomial-Time Reductions

Design patterns and anti-patterns

Algorithm design patterns.

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

Algorithm design anti-patterns.

- NP-completeness. $O(n^k)$ algorithm unlikely.
- PSPACE-completeness. $O(n^k)$ certification algorithm unlikely.
- Undecidability. No algorithm possible.

Recap: Efficiency

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

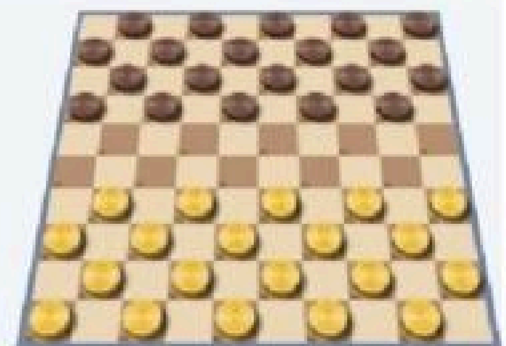
Theory. Definition is broad and robust.
Practice. Poly-time algorithms scale to huge problems.

Conceptual: Classify problems

Idea. Classify problems: can be solved in poly-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 game of n-by-n checkers, can black guarantee a win?



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

Practical: Poly-time reductions

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

Reduction. Problem X **polynomial-time 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.

Practical: Poly-time reductions

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

Reduction. Problem X **polynomial-time 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. $X \leq_P Y$.

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

Common mistake. Confusing $X \leq_P Y$ with $Y \leq_P X$.

Quiz: $X \leq_P Y$

Suppose that $X \leq_P Y$. Which of the following can we infer?

A. If X can be solved in polynomial time, then so can Y.
B. X can be solved in poly time iff Y can be solved in poly time.
C. If X cannot be solved in polynomial time, then neither can Y.
D. If Y cannot be solved in polynomial time, then neither can X.

Quiz: $X \leq_P Y$

Suppose that $X \leq_P Y$. Which of the following can we infer?

A. If X can be solved in polynomial time, then so can Y.
B. X can be solved in poly time iff Y can be solved in poly time.
C. If X cannot be solved in polynomial time, then neither can Y.
D. If Y cannot be solved in polynomial time, then neither can X.

C. contrapositive

Poly-time reductions

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.

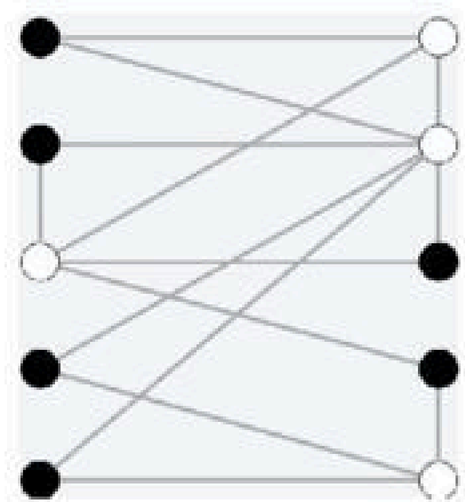
Bottom line. Reductions classify problems according to relative difficulty.

Packing and covering

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?

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

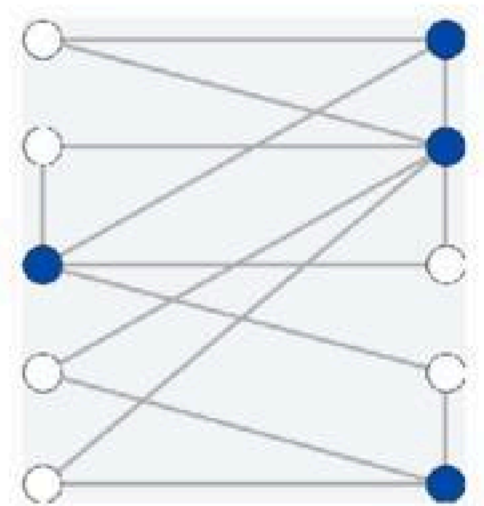


Optimization: [Packing] What is the maximum size independent set?

Vertex cover

VERTEX-COVER. Given a graph $G = (V, E)$ and an integer k, is there a subset of $\leq k$ vertices that each edge is incident to at least one vertex in the subset?

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

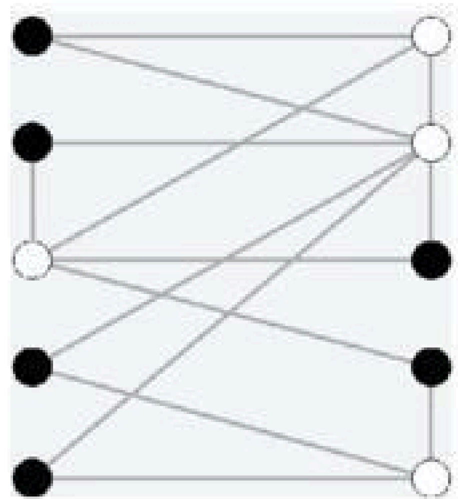


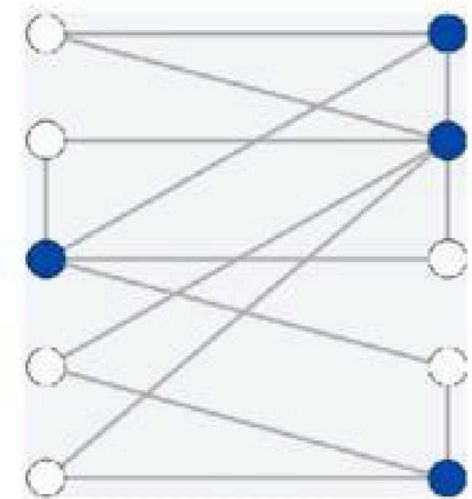
Optimization: [Covering] What is the minimum size vertex cover?

Packing $\equiv_P$ Covering

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

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





Packing $\equiv_P$ Covering: $\Rightarrow$

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

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

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

Packing $\equiv_P$ Covering: $\Leftarrow$

Theorem. INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.
Pf. 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 vertex cover 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.
- $\Rightarrow$ INDEPENDENT-SET $\leq_P$ VERTEX-COVER.

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?

Sample application. software-services.

	1	2	3	4	5	6	7
A			+				
B		+		+			
C			+	+	+	+	
D					+		
E	+						
F	+	+				+	+

Vertex Cover $\leq_P$ Set Cover

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

Pf. 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.

Vertex Cover $\leq_P$ Set Cover

Theorem. VERTEX-COVER $\leq_P$ SET-COVER.
Pf. 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 $v \in V : S_v = \{e \in E : e \text{ incident to } v\}$.

Vertex Cover $\leq_P$ Set Cover: Lemma

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

Pf. $\Rightarrow$ 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.

Vertex Cover $\leq_P$ Set Cover: Lemma (cont.)

Lemma. $G = (V, E)$ contains a vertex cover of size k iff (U, S, k) contains a set cover of size k.
Pf. $\Leftarrow$ 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.

Constraint satisfaction

Recap: Conjunctive normal form (CNF)

Literal. A Boolean variable or its negation: $x_i, \bar{x_i}$.

Clause. A *disjunction* of literals: eg., $C_j = x_1 \vee \bar{x_2} \vee x_3$.

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

Satisfiability

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).

Ex. $\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4)$

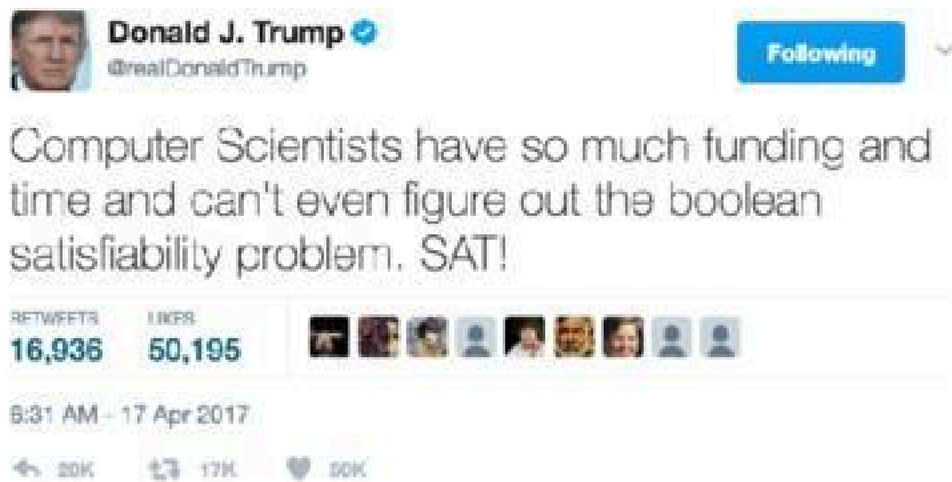
- yes instance: $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$

Key application. Electronic design automation (EDA).

Satisfiability is hard

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

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



3-SAT $\leq_P$ INDEPENDENT-SET

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

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

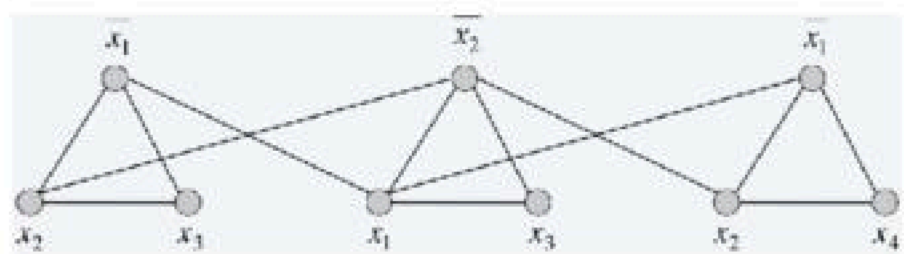
3-SAT $\leq_P$ INDEPENDENT-SET

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

Pf. Given a 3-SAT instance Φ, we construct a INDEPENDENT-SET instance (G, k) that has a size $k = |\Phi|$ independent set 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.



$$\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4)$$

3-SAT $\leq_P$ INDEPENDENT-SET: Lemma

Lemma. Φ is satisfiable iff G contains an independent set of size $k = |\Phi|$.

Pf. $\Rightarrow$ Consider any satisfying assignment for Φ.

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

3-SAT $\leq_P$ INDEPENDENT-SET: Lemma

Lemma. Φ is satisfiable iff G contains an independent set of size $k = |\Phi|$.

Pf. $\Leftarrow$ 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.

Review

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.

Review

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 $X \leq_P Z$.
Pf idea. Compose those two algorithms.

Ex. 3-SAT $\leq_P$ INDEPENDENT-SET $\equiv_P$ VERTEX-COVER $\leq_P$ SET-COVER.

Decision, Search, Optimization

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.

Decision vs. Search

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.

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

Pf. $\geq_P$ To find a vertex cover of size $\leq k$:

- Determine if there exists a vertex cover of size $\leq k$.
- Enumerate V and find a vertex v that $G - \{v\}$ has a vertex cover of size $\leq k - 1$. (any vertex in any vertex cover of size $\leq k$ will suffice)
- Include v in the vertex cover.
- Recursively find a vertex cover of size $\leq k - 1$ in $G - \{v\}$.

Search vs. Optimization

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.

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

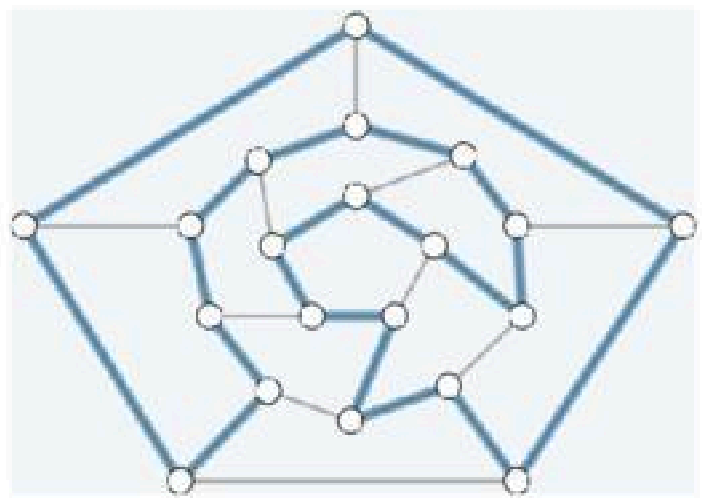
Pf. $\geq_P$ To find vertex cover of minimum size:

- Binary search (or linear search) for size k^* of min vertex cover.
- Solve search problem for given k^*.

Sequencing problems

Hamilton cycle

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



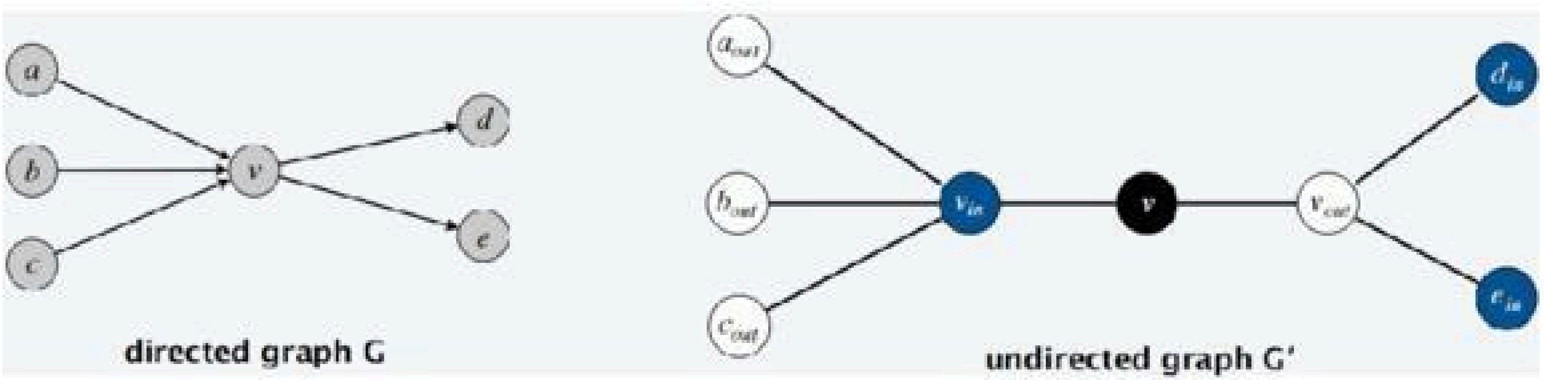
Sequencing Problems. Search over all *permutations* of a collection of objects.

- Ex. Traveling Salesman Problem.
  - missing ordering?

Directed Hamilton cycle

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

Theorem. DIR-HAM-CYCLE $\leq_P$ HAM-CYCLE.
Pf. Given a directed graph $G = (V, E)$, construct a graph G' with $3n$ nodes.



DIR-HAM-CYCLE $\leq_P$ HAM-CYCLE

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

Pf. $\Rightarrow$

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

Pf. $\Leftarrow$

- Suppose G' has an undirected Hamilton cycle Γ'.
- Γ' must visit nodes in G' using one of following two reverse orders:
  - $\ldots, B, K, W, B, K, W, B, K, W, \ldots$
  - $\ldots, W, K, B, W, K, B, W, K, B, \ldots$
- Black nodes in Γ' comprise either a directed Hamilton cycle Γ in G, or reverse of one.

3-SAT $\leq_P$ DIR-HAM-CYCLE

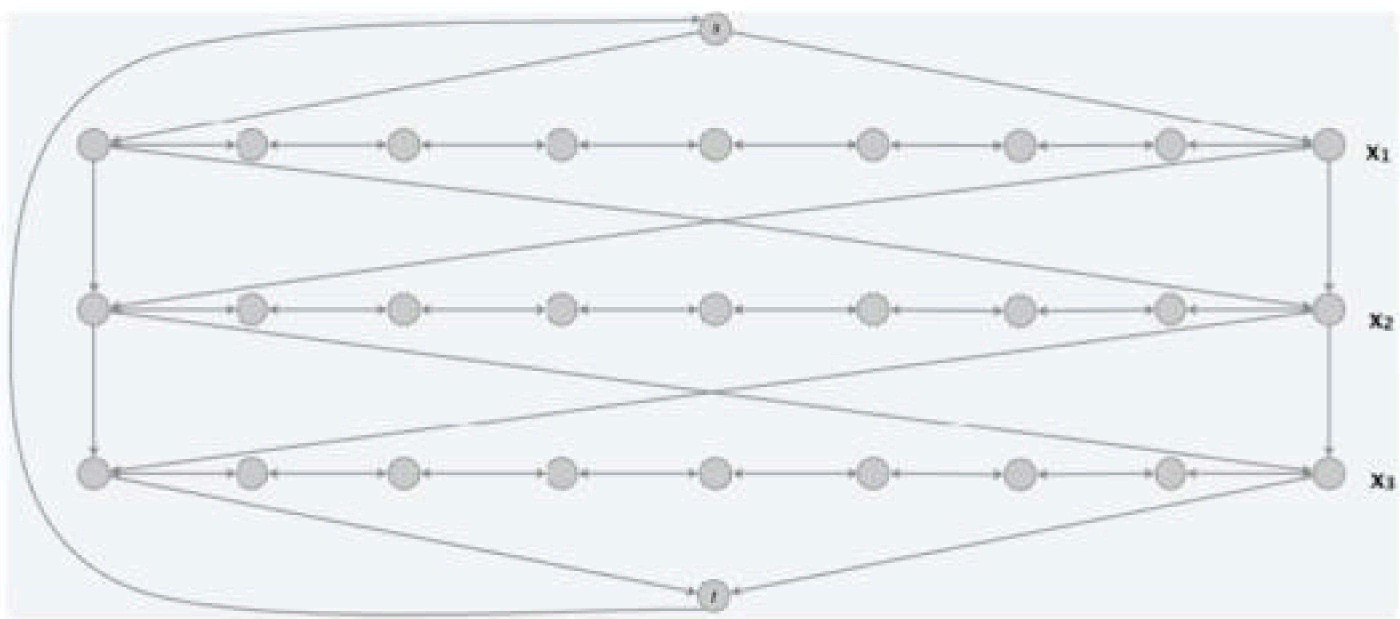
Theorem. 3-SAT $\leq_P$ DIR-HAM-CYCLE.
Pf. Given an instance Φ of 3-SAT, we construct an instance G of DIR-HAM-CYCLE that has a Hamilton cycle iff Φ is satisfiable.

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

Construction: variable

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

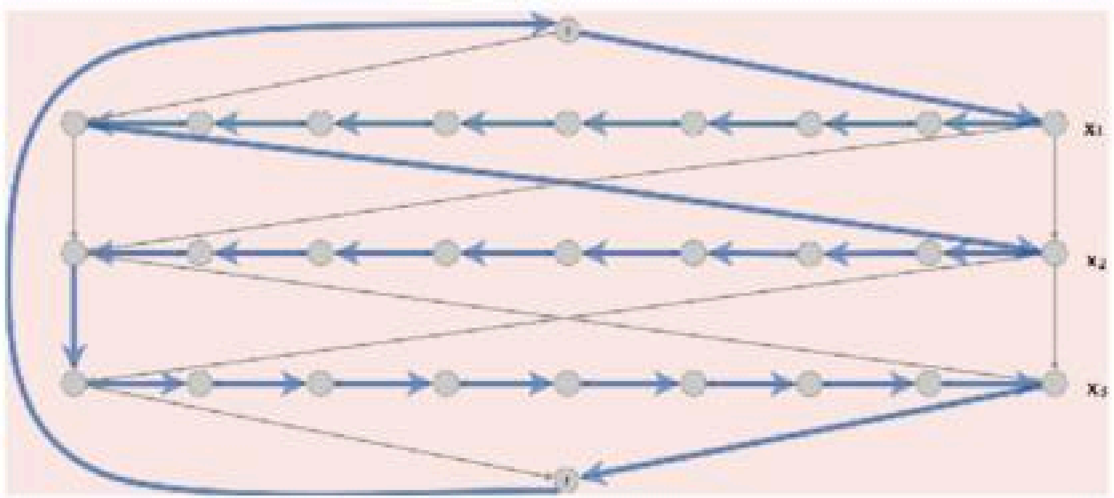
- Construct G to have 2^n Hamilton cycles.
- Intuition: traverse path i from left to right $\Leftrightarrow$ set variable x_i = `true`.



Quiz: DIR-HAM-CYCLE

Which is truth assignment corresponding to Hamilton cycle below?

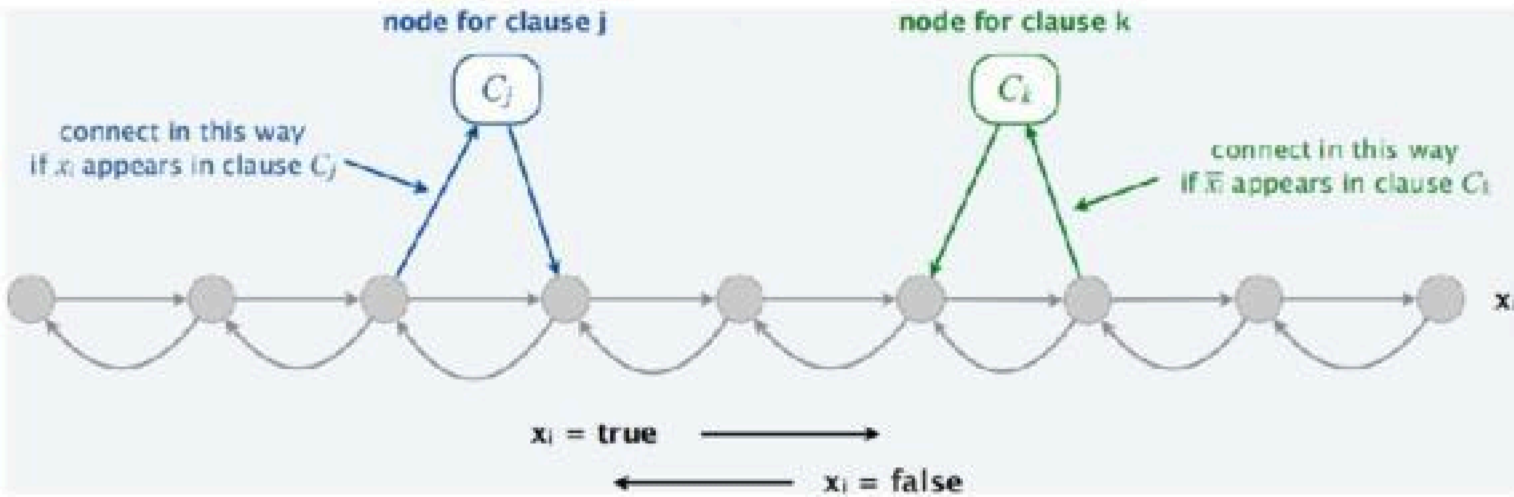
- x_1 = `true`, x_2 = `true`, x_3 = `true`
- x_1 = `true`, x_2 = `true`, x_3 = `false`
- x_1 = `false`, x_2 = `false`, x_3 = `true`
- x_1 = `false`, x_2 = `false`, x_3 = `false`



Construction: clause

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

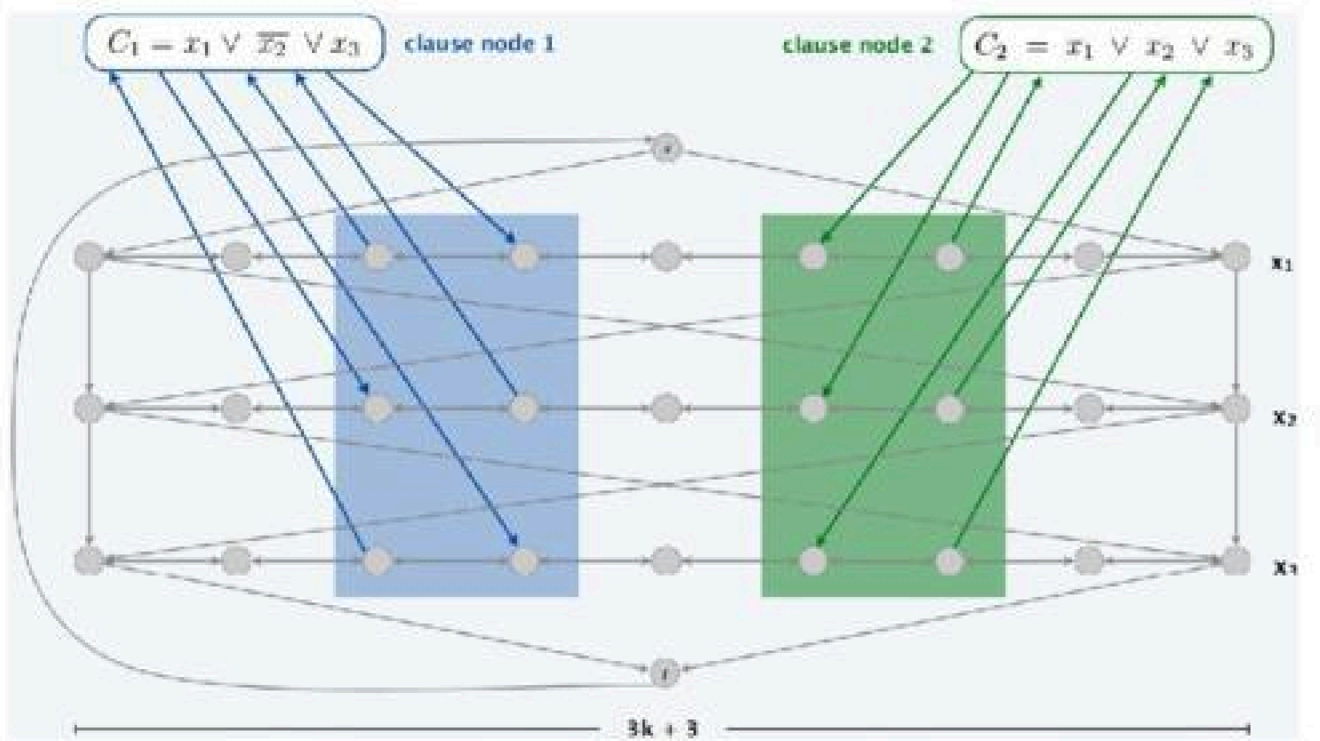
- For each clause: add a node and 2 edges per literal.



Construction: example

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.

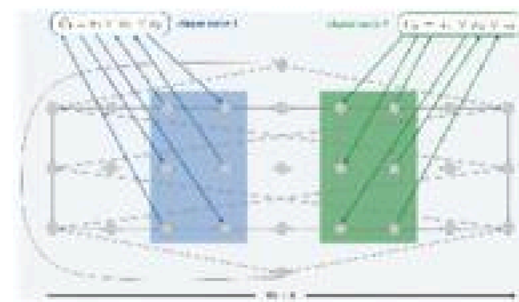


3-SAT $\leq_P$ DIR-HAM-CYCLE: Lemma

Lemma. Φ is satisfiable iff G has a Hamilton cycle.

Pf. $\Rightarrow$

- Suppose 3-SAT instance Φ has satisfying assignment x^*.
- Then, define Hamilton cycle Γ in G as follows:

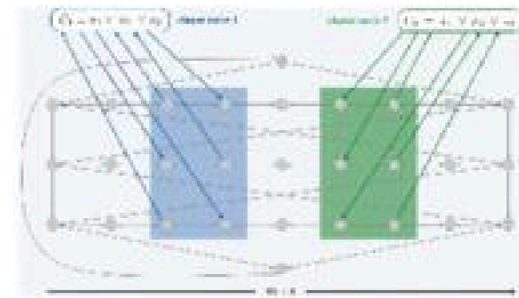


- for each variable x_i^*,
  - if x_i^* = `true`, traverse row i from left to right
  - if x_i^* = `false`, traverse row i from right to left
- for each clause C_j,
  - at least one row i in which we are going in "correct" direction
  - splice clause node C_j into cycle (and splice C_j exactly once)

3-SAT $\leq_P$ DIR-HAM-CYCLE: Lemma

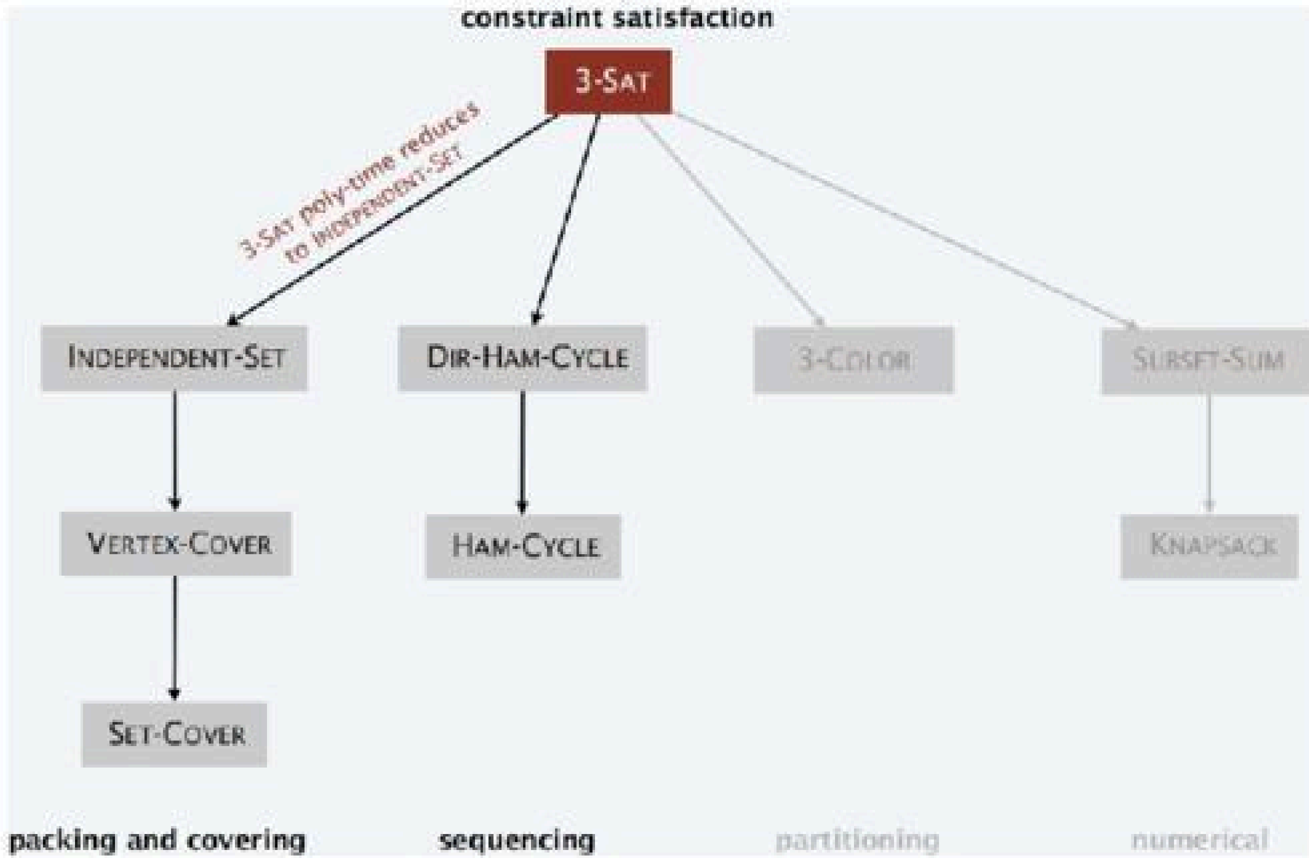
Lemma. Φ is satisfiable iff G has a Hamilton cycle.

Pf. $\Leftarrow$



- Suppose G has a Hamilton cycle Γ.
- If Γ enters clause node C_j, it must depart on a parallel (variable) edge.
  - nodes neighbor to C_j are connected by an edge $e \in E$
- remove C_j from cycle, and replace 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.

Poly-time reductions: review I



Partitioning problems

3-dimensional matching

3D-MATCHING. Given n instructors, n courses, and n times, and a list of the possible courses and times each instructor is willing to teach, is it possible to make an assignment so that all courses are taught at different times?

Ex. Three courses, Mon.-Wed. afternoon.

I	**M**	**T**	**W**
A			AI, **AD**
B	AI	**AI**, SE	
C	AI, **SE**		SE

{A, AD, W}, {B, AI, T}, {C, SE, M}

3D-MATCHING

3D-MATCHING. Given 3 disjoint sets X, Y, Z, each of size n and a set $T \subseteq X \times Y \times Z$ of triples, does there exist a set of n triples in T such that each element of $X \cup Y \cup Z$ is in *exactly one* of these triples?

Remark. Generalization of bipartite matching.

- each element of $X \cup Y$ is in *exactly one* of $X \times Y$

3D-MATCHING

3D-MATCHING. Given 3 disjoint sets X, Y, Z, each of size n and a set $T \subseteq X \times Y \times Z$ of triples, does there exist a set of n triples in T such that each element of $X \cup Y \cup Z$ is in *exactly one* of these triples?

Remark. Generalization of bipartite matching.

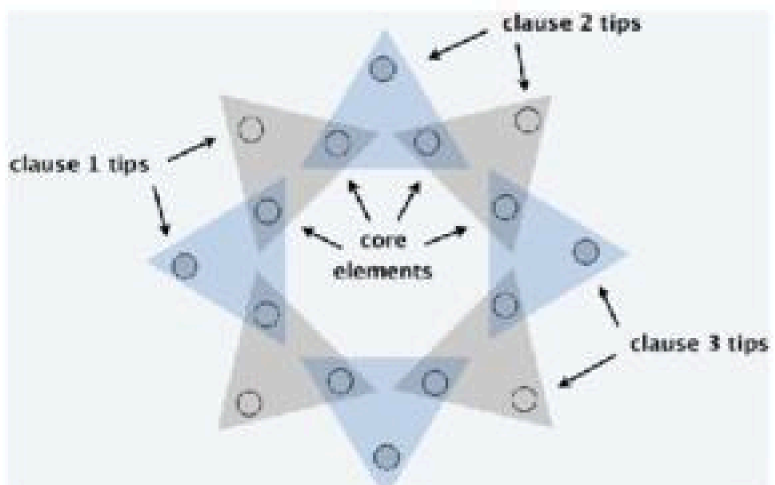
- each element of $X \cup Y$ is in *exactly one* of $X \times Y$

Theorem. 3-SAT $\leq_P$ 3D-MATCHING.
Pf. Given an instance Φ of 3-SAT, we construct an instance of 3D-MATCHING that has a perfect matching iff Φ is satisfiable.

Constructing gadget: variable

Construction. (variable)

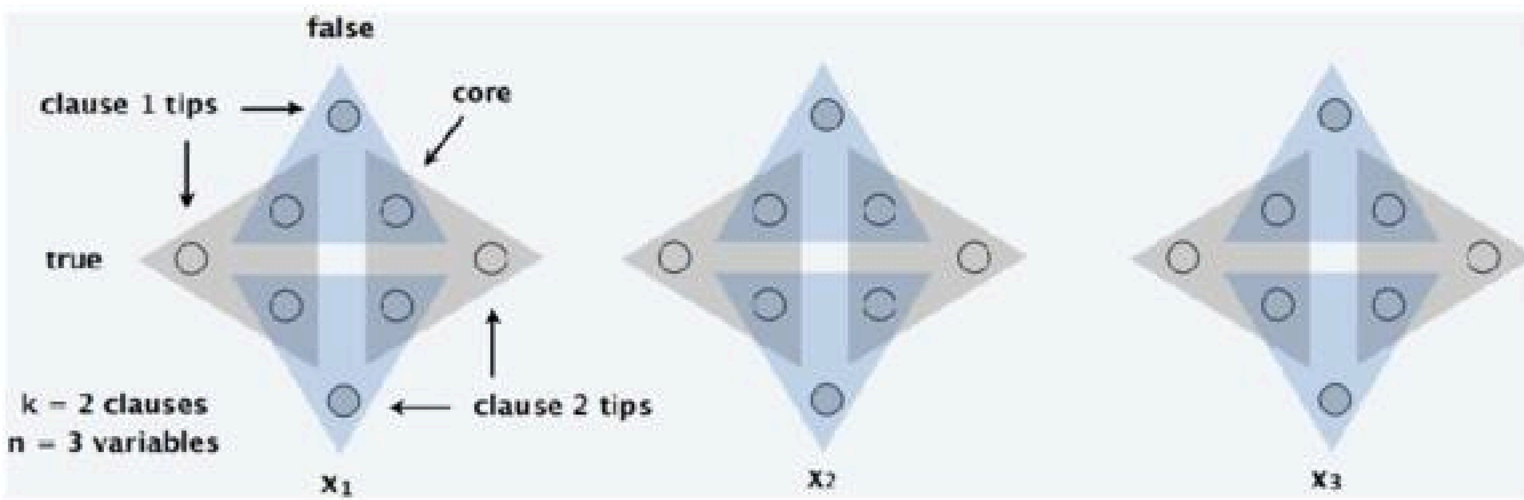
- Create gadget for each variable x_i with $2k$ *core* elements and $2k$ *tip* ones.
  - k: number of clauses, or triplets.
  - tip: assignment of one variable.
  - core: one pair in some triplet.



Constructing gadget: variable (cont.)

Construction. (variable)

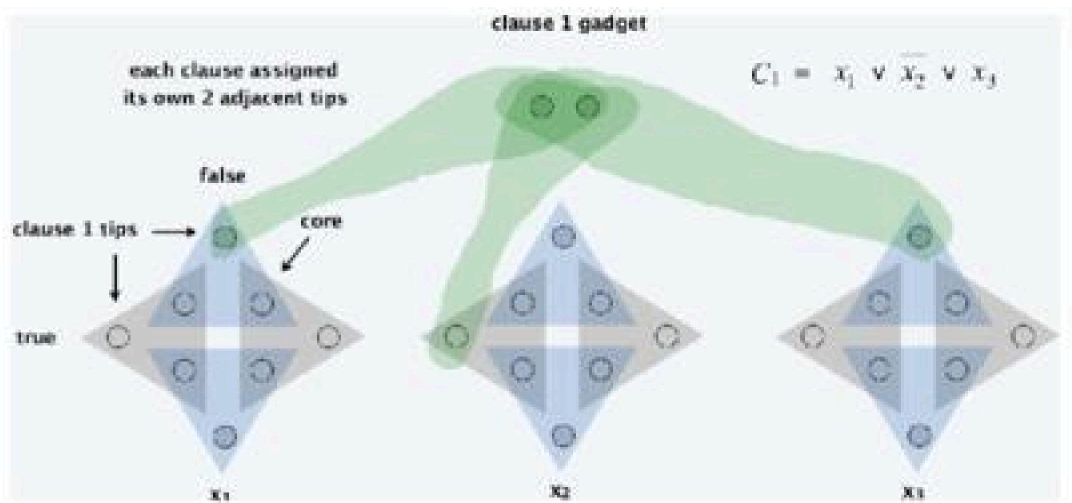
- Create gadget for each variable x_i with $2k$ *core* elements and $2k$ *tip* ones.
  - A perfect matching will not use overlapping core elements.
    - In gadget for x_i, must use either all gray triples (*even*: x_i = `true`) or all blue ones (*odd*: x_i = `false`).
  - Or view from tips: two possible choices.



Constructing gadget: clause

Construction. (clause)

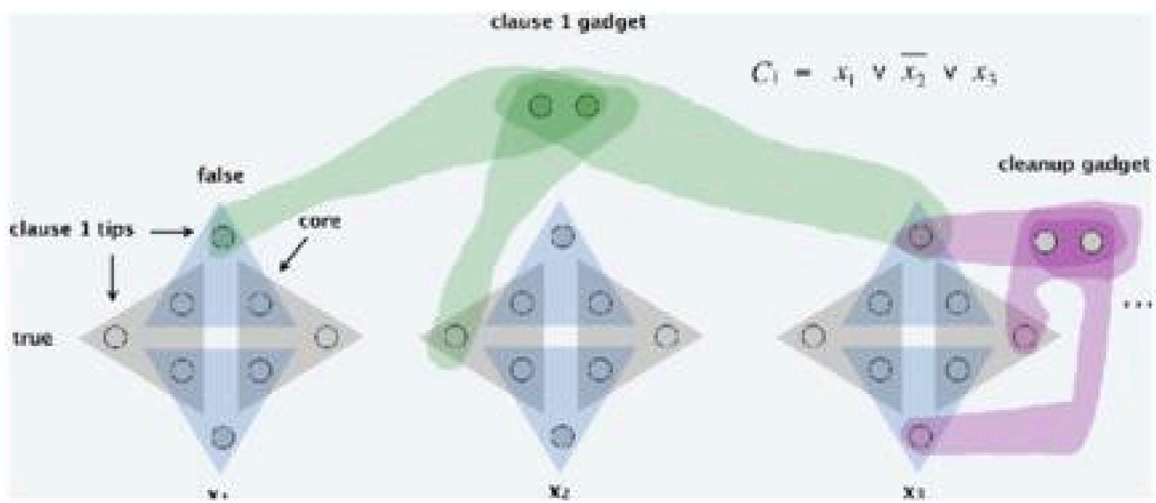
- Create gadget for each clause C_j with two *core* elements and three triples.
  - *Exactly one* of these triples will be used in any 3d-matching.
    - Ensures example perfect matching uses either: (i) grey core of x_1 or (ii) blue core of x_2 or (iii) grey core of x_3.
  - *Opposite* to truth assignment of variables.



Constructing gadget: cleanup

Construction. (cleanup)

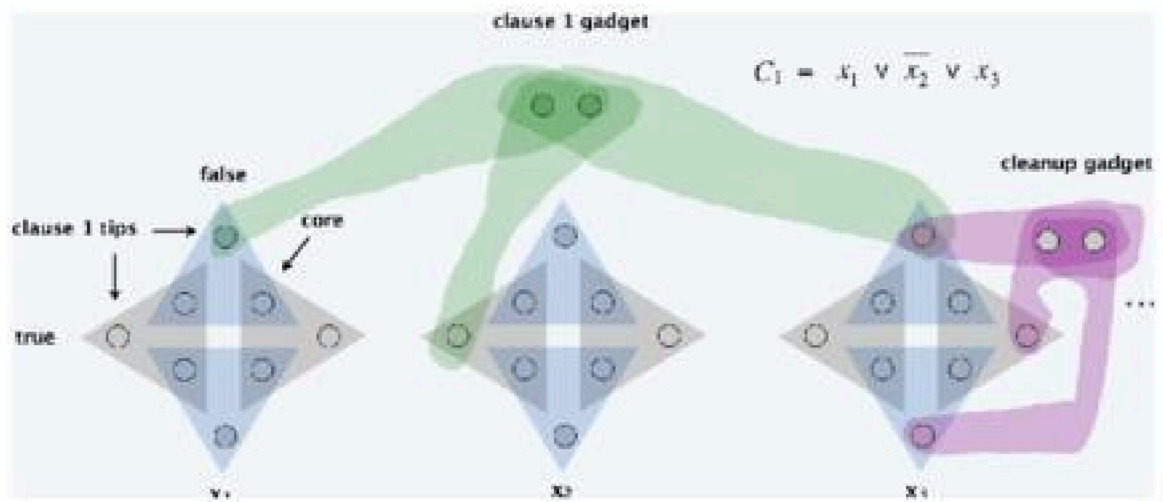
- There are $2nk$ tips: nk covered by blue/gray triples; k by clause triples.
- To cover remaining $(n-1)k$ tips, create $(n-1)k$ “cleanup” gadgets: same as clause gadget but with $2nk$ triples, connected to *every* tip.



3-SAT $\leq_P$ 3D-MATCHING

Lemma. Instance (X, Y, Z) has a perfect matching iff Φ is satisfiable.

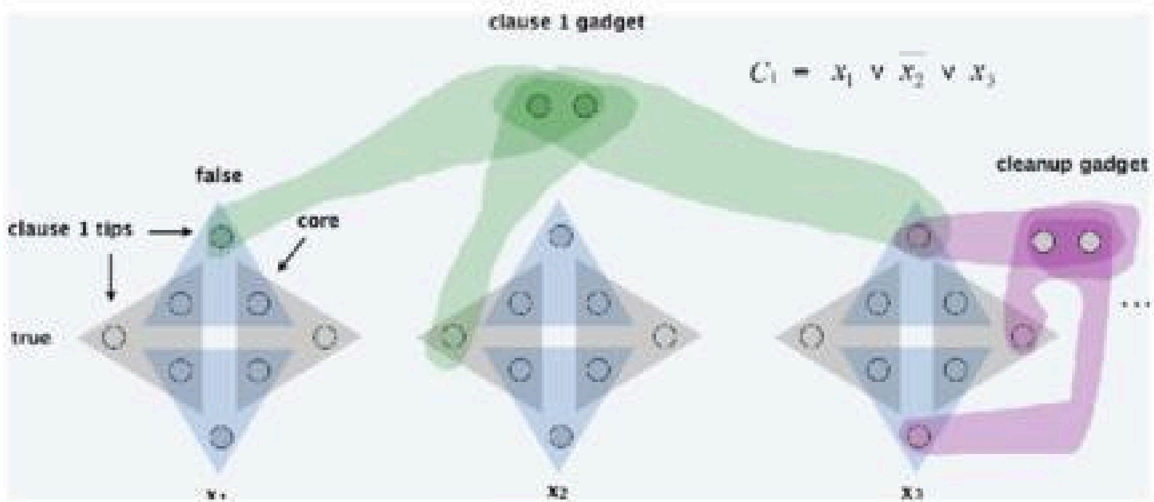
Q. What are X, Y, and Z?



3-SAT $\leq_P$ 3D-MATCHING (cont.)

Lemma. Instance (X, Y, Z) has a perfect matching iff Φ is satisfiable.

Q. What are X, Y, and Z?
A. X = `black`, Y = `white`, and Z = `blue`.

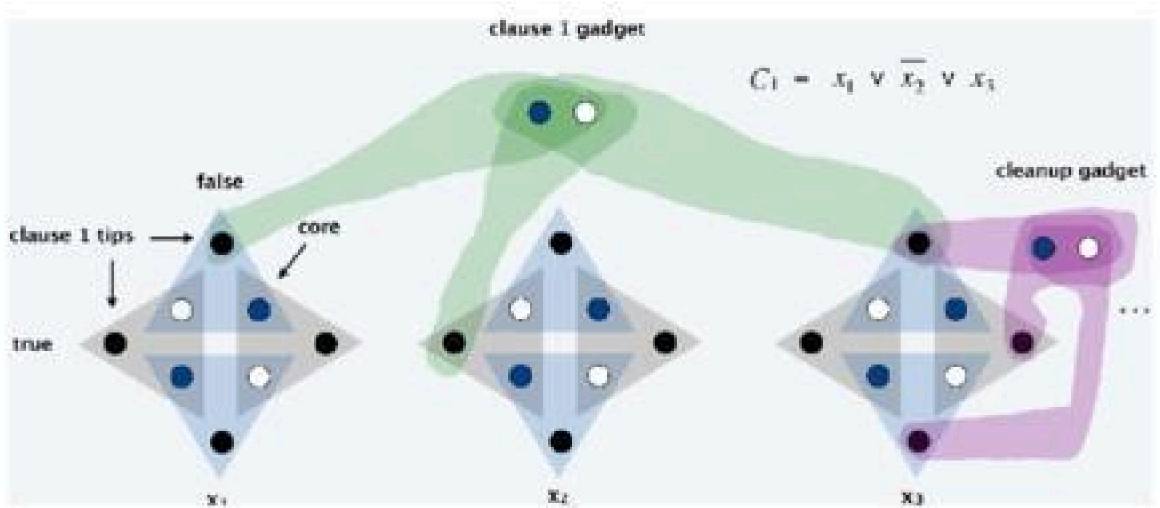


3-SAT $\leq_P$ 3D-MATCHING: proof

Lemma. Instance (X, Y, Z) has a perfect matching iff Φ is satisfiable.

Pf. $\Rightarrow$ If 3d-matching, then assign x_i according to gadget x_i.

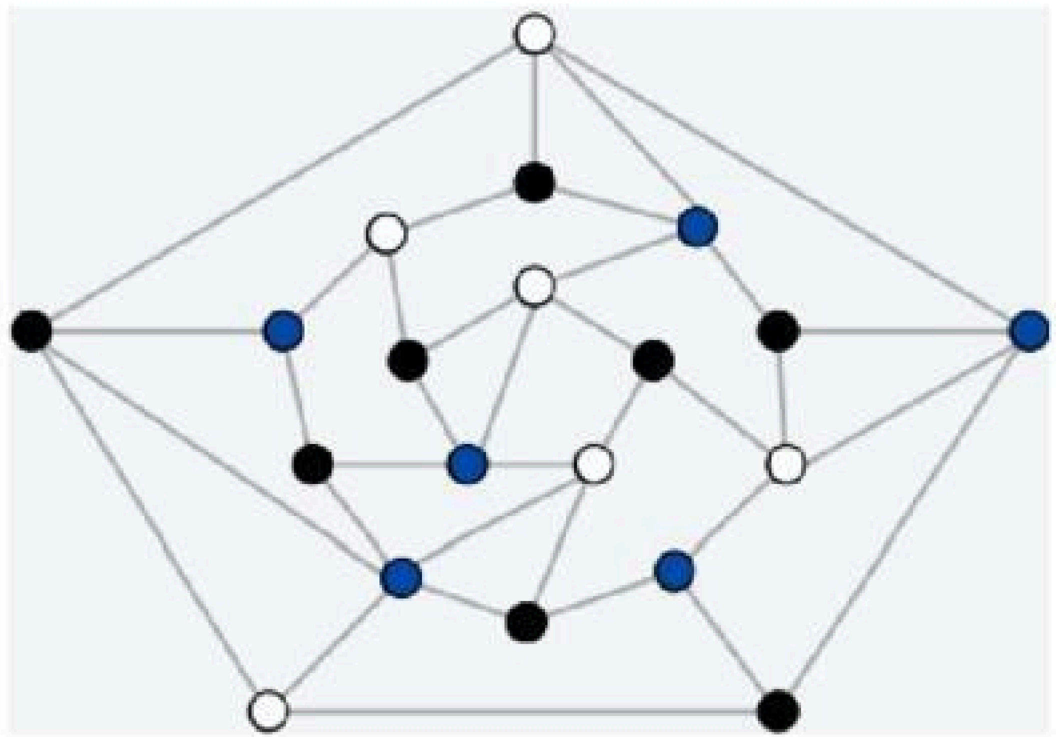
Pf. $\Leftarrow$ If Φ is satisfiable, use any true literal in C_j to select gadget C_j triple.



Graph coloring

3-colorability

3-COLOR. Given an undirected graph G, can the nodes be colored `black`, `white`, and `blue` so that no adjacent nodes have the same color?



Quiz: 2-COLOR

How difficult to solve 2-COLOR?

A. $O(m+n)$ using BFS or DFS.
B. $O(mn)$ using maximum flow.
C. $\Omega(2^n)$ using brute force.
D. Not even Tarjan knows.

Quiz: 2-COLOR

How difficult to solve 2-COLOR?

A. $O(m+n)$ using BFS or DFS.
B. $O(mn)$ using maximum flow.
C. $\Omega(2^n)$ using brute force.
D. Not even Tarjan knows.

A graph G is 2-colorable if and only if it is bipartite.

- so, $O(m+n)$
- see Section 3.4

Application: register allocation

Register allocation. Assign program variables to machine registers so that: (i) no more than k registers are used, (ii) and no two program variables that are needed at the same time are assigned to the same register.

Application: register allocation

Register allocation. Assign program variables to machine registers so that: (i) no more than k registers are used, (ii) and no two program variables that are needed at the same time are assigned to the same register.

Interference graph. Nodes are program variables; edge between u and v if there exists an operation where both u and v are "live" at the same time.

Observation. [Chaitin 1982] Can solve register allocation problem iff interference graph is k-colorable.

Application: register allocation

Register allocation. Assign program variables to machine registers so that: (i) no more than k registers are used, (ii) and no two program variables that are needed at the same time are assigned to the same register.

Interference graph. Nodes are program variables; edge between u and v if there exists an operation where both u and v are "live" at the same time.

Observation. [Chaitin 1982] Can solve register allocation problem iff interference graph is k-colorable.

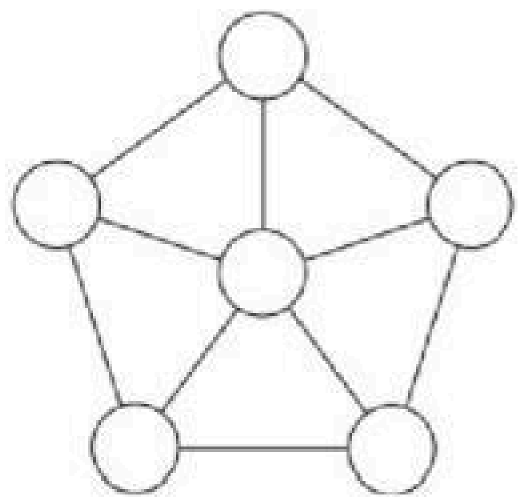
Fact. 3-COLOR $\leq_P$ K-REGISTER-ALLOCATION for any constant $k \geq 3$.

3-SAT $\leq_P$ 3-COLOR

Theorem. 3-SAT $\leq_P$ 3-COLOR.

Pf. Given 3-SAT instance Φ, we construct an instance of 3-COLOR that is 3-colorable iff Φ is satisfiable.

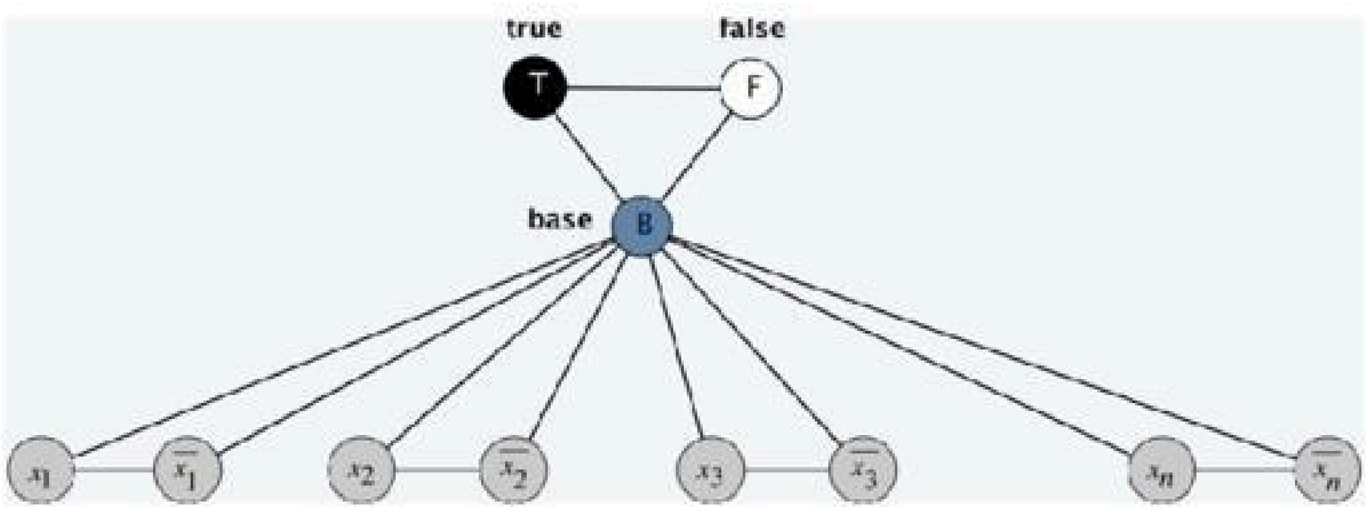
- Intuition: see the following graph which is not 3-colorable.



3-SAT $\leq_P$ 3-COLOR: Construction

Construction.

1. Create a graph G with a node for each literal.
2. Connect each literal to its negation.
3. Create 3 new nodes T, F, and B; connect them in a triangle.
4. Connect each literal to B.
5. For each clause C_j, add a gadget of 6 nodes and 13 edges.

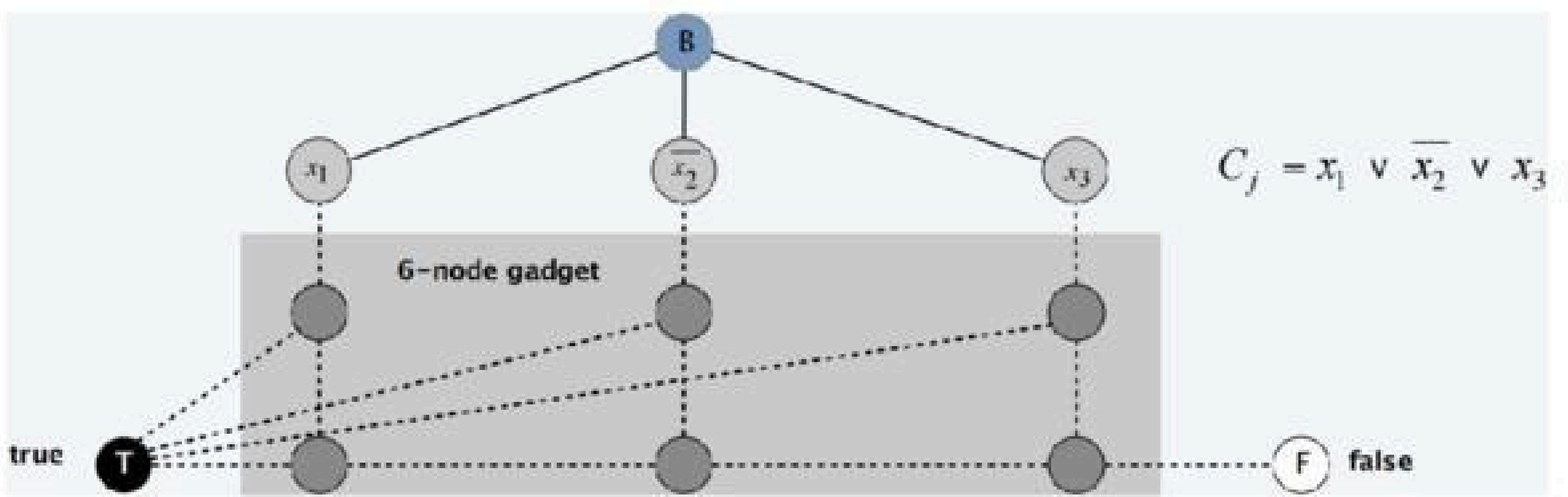


3-SAT $\leq_P$ 3-COLOR: $\Rightarrow$

Lemma. Graph G is 3-colorable iff Φ is satisfiable.

Pf. $\Rightarrow$ Suppose graph G is 3-colorable.

- WLOG, assume node T is colored `black`, F is `white`, and B is `blue`.
- Consider assignment sets all `black` literals to `true` (and `white` to `false`).
- #4 ensures each literal is colored either `black` or `white`.
- #2 ensures each literal is `white` if its negation is `black` (and vice versa).

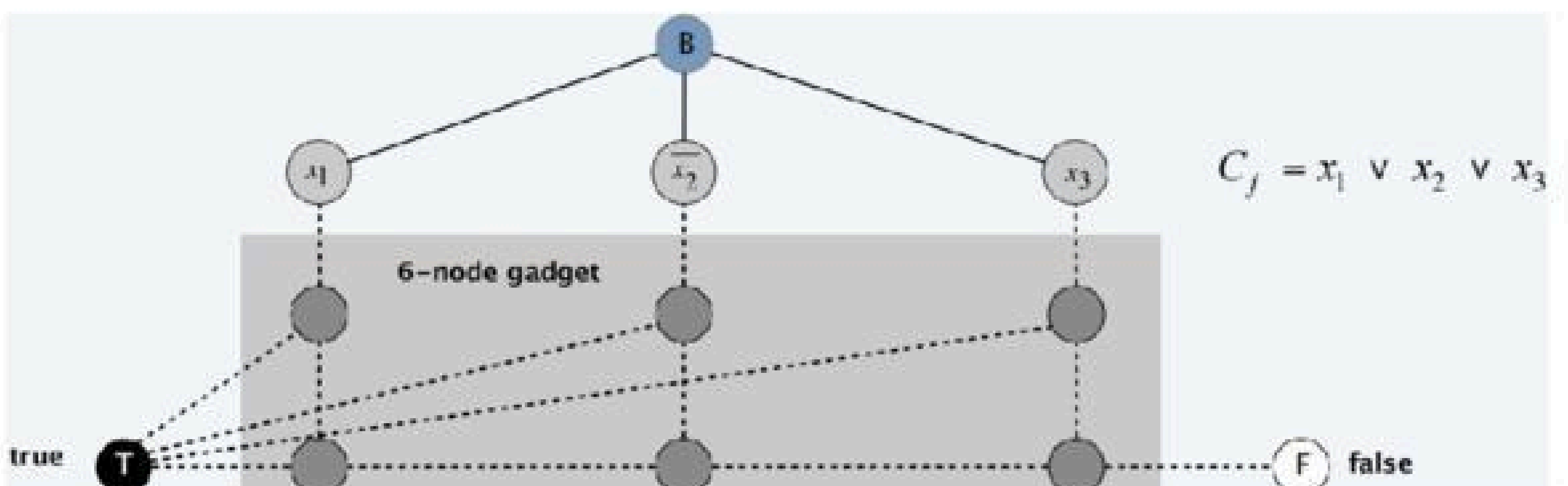


3-SAT $\leq_P$ 3-COLOR: $\Rightarrow$ (cont.)

Lemma. Graph G is 3-colorable iff Φ is satisfiable.

Pf. $\Rightarrow$ Suppose graph G is 3-colorable.

- #5 ensures at least one literal in each clause is `black`.
  - suppose (for contradiction) all 3 literals are `white` in some 3-coloring
  - then first row must be 3 `blue`,
  - then second row must alternate between `white` & `black`,
    - no possible coloring.

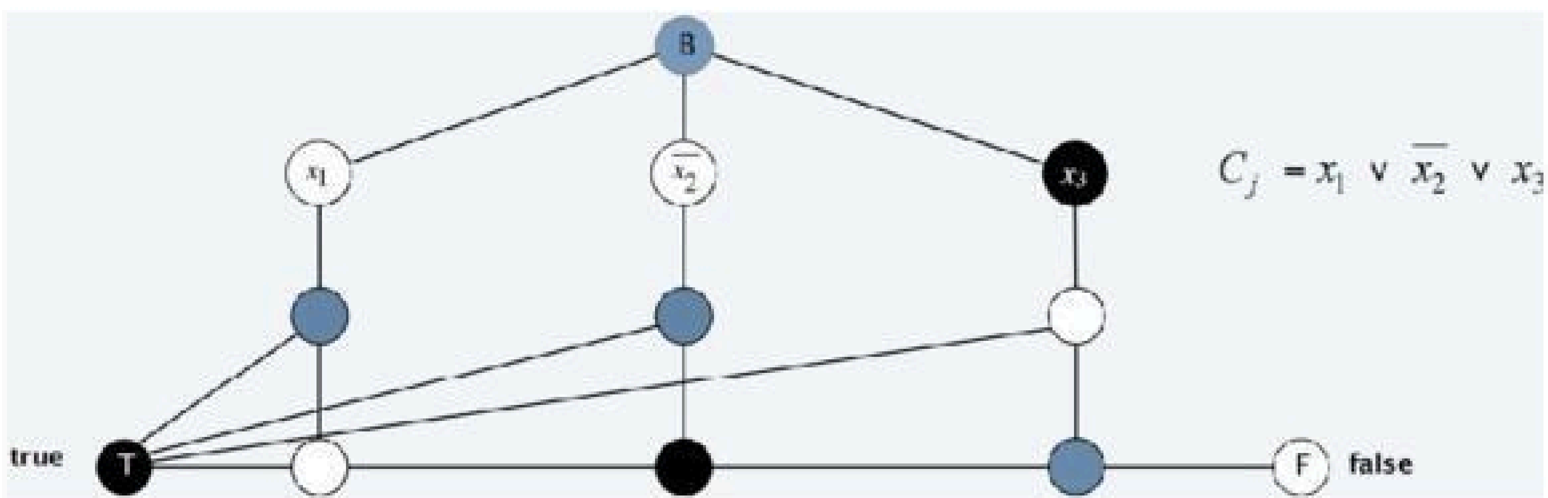


3-SAT $\leq_P$ 3-COLOR: $\Leftarrow$

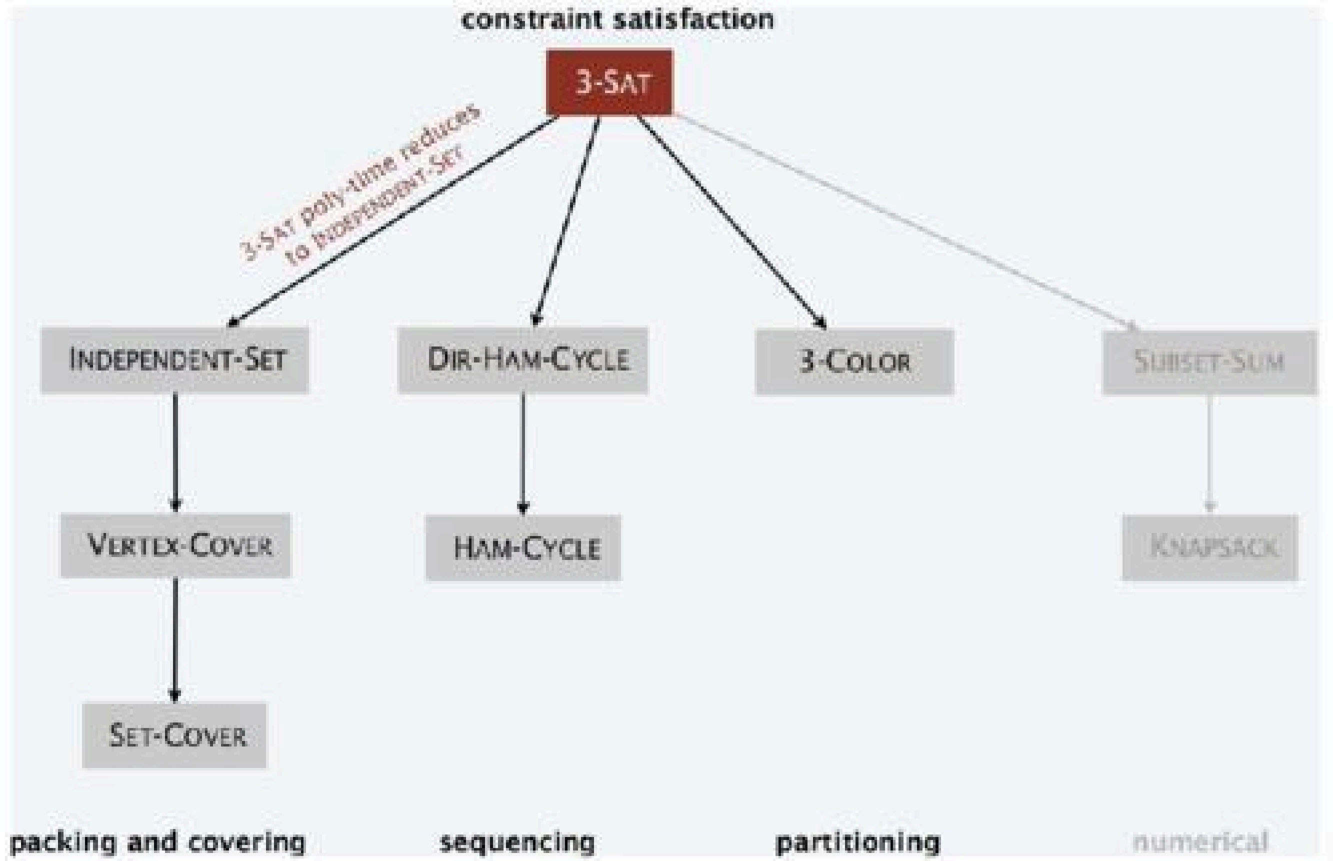
Lemma. Graph G is 3-colorable iff Φ is satisfiable.

Pf. $\Leftarrow$ Suppose 3-SAT instance Φ is satisfiable.

- Color all `true` literals `black` and all `false` literals `white`.
- Pick one `true` literal; color node below that node `white`, and node below that `blue`.
- Color remaining middle row nodes `blue`.
- Color remaining bottom nodes `black` or `white`, as forced.



Poly-time reductions: review II



Numerical problems

Subset sum

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?

Ex. $\{215, 215, 275, 275, 355, 355, 420, 420, 580, 580, 655, 655\}, W = 1505.$
Yes. $215 + 355 + 355 + 580 = 1505.$

Why is it a problem?

We solved it using dynamic programming with time $O(nW)$.

Why is it a problem?

We solved it using dynamic programming with time $O(nW)$.

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

- problem comes when W is large.
- ex. 100 numbers, each number is 100 bits long:
  - input: $100 \times 100 = 10000$ digits,
  - W: roughly 2^{100}, *exponential* to size of input.

Why is it a problem?

We solved it using dynamic programming with time $O(nW)$.

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

- problem comes when W is large.
- ex. 100 numbers, each number is 100 bits long:
  - input: $100 \times 100 = 10000$ digits,
  - W: roughly 2^{100}, *exponential* to size of input.

We referred to such problem as **Pseudo-polynomial**.

- ran in time polynomial in the magnitude of the input numbers,
- but not polynomial in the size of their representation.

3-SAT $\leq_P$ SUBSET-SUM

Theorem. 3-SAT $\leq_P$ SUBSET-SUM.

Pf. Given an instance Φ of 3-SAT, we construct an instance of SUBSET-SUM that has solution iff Φ is satisfiable.

3-SAT $\leq_P$ SUBSET-SUM: construction

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.
  - two numbers for each variable x_i.
  - two numbers for each clause C_j.
- Sum of x_i column is 1; sum of C_j column is 4.

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

C_1	=	$\neg x_1$	$\vee$ x_2	$\vee$	x_3
C_2	=	x_1	$\vee$ $\neg x_2$	$\vee$	x_3
C_3	=	$\neg x_1$	$\vee$ $\neg x_2$	$\vee$	$\neg x_3$

	x_1	x_2	x_3	C_1	C_2	C_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
x_3	0	0	1	1	1	0	1,110
$\neg x_3$	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
	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

3-SAT $\leq_P$ SUBSET-SUM: $\Rightarrow$

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

Pf. $\Rightarrow$ 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 $\neg x_i$ rows.
- Select dummy integers to make C_j digits sum to 4.

C_1	=	$\neg x_1$	$\vee$ x_2	$\vee$	x_3
C_2	=	x_1	$\vee$ $\neg x_2$	$\vee$	x_3
C_3	=	$\neg x_1$	$\vee$ $\neg x_2$	$\vee$	$\neg x_3$

	x_1	x_2	x_3	C_1	C_2	C_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
x_3	0	0	1	1	1	0	1,110
$\neg x_3$	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
	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

3-SAT $\leq_P$ SUBSET-SUM: $\Leftarrow$

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

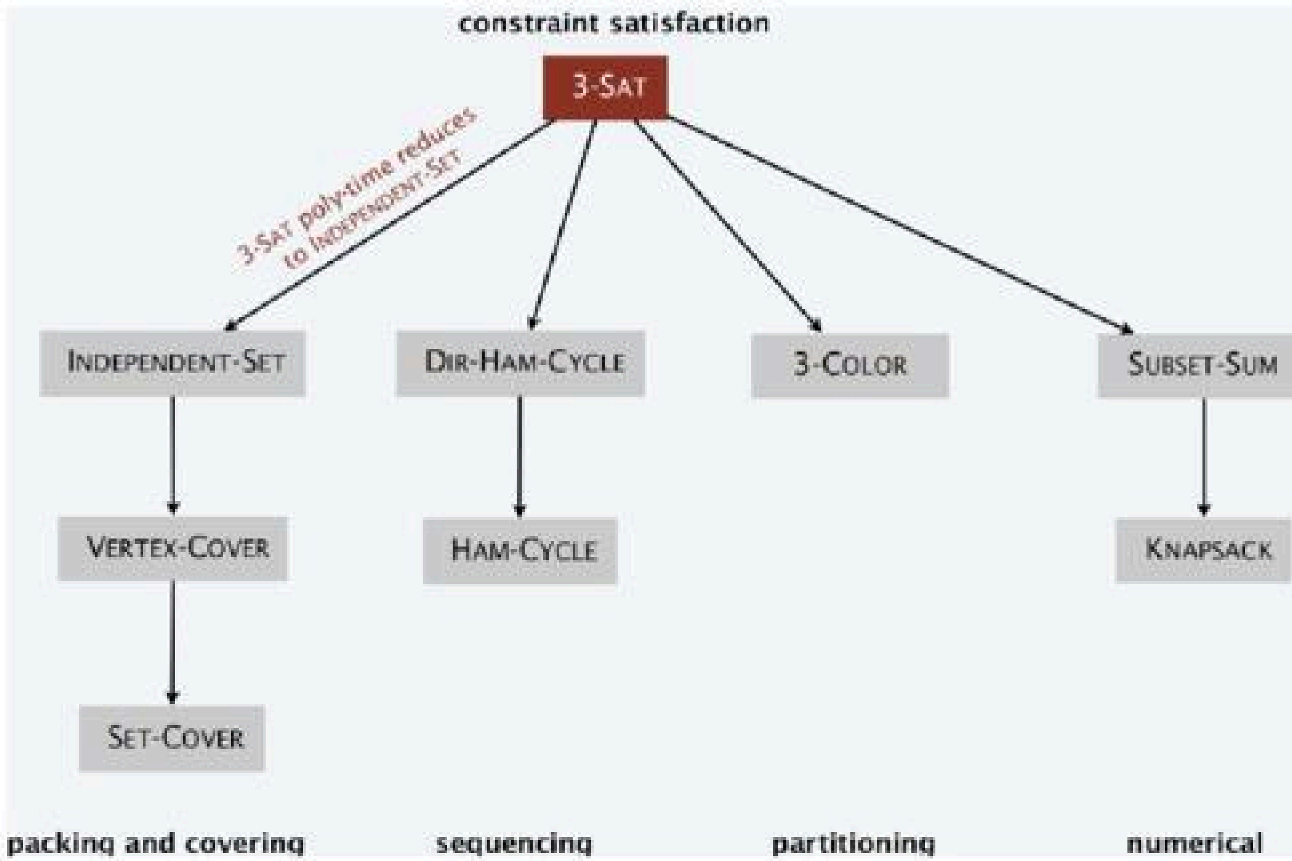
Pf. $\Leftarrow$ Suppose there exists a subset S^* that sums to W.

- 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 Cj forces subset S^* to select at least one literal in clause.

C_1 =	$\neg x_1$	∨	x_2	∨	x_3
C_2 =	x_1	∨	$\neg x_2$	∨	x_3
C_3 =	$\neg x_1$	∨	$\neg x_2$	∨	$\neg x_3$

	x_1	x_2	x_3	C_1	C_2	C_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
x_3	0	0	1	1	1	0	1,110
$\neg x_3$	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
	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

Poly-time reductions: review III



Karp's 20 reductions from satisfiability

Karp [1972], 1985 Turing Award.