COMP 6651 / Winter 2022
Dr. B. Jaumard
Lecture 7 - Part I: NP-Completeness

Professor B. Jaumard

CSE Department, Concordia University, Canada

February 18, 2022

Outline

Tractable vs. Intractable Problems

Tractable

Problems which can be solved by algorithms which run in time $T(n) = n^k$ for some constant $k > 0$ where n is the input size.

- A problem is **intractable** if it is not tractable
- Are all problems tractable? Far from it
- No known polynomial time algorithm for
  - Halting Problem: Given a program P written in Java and an input x, does $P(x)$ terminates? Un-decidable!
  - Traveling Salesman Problem: On a weighted graph G, find shortest closed path that visits each vertex exactly once of minimal cost
  - Factoring: On input N, find its prime factorization.

Classify Problems

Three general categories

- Problems for which polynomial-time algorithms have been found
- Problems that have been proven to be intractable
- Problems that have not been proven to be intractable but for which polynomial-time algorithms have never been found

Problems for which polynomial-time algorithms have been found (1/3)

Examples

- Sorting $\theta(n \log n)$
- Matrix multiplication $\theta(n^{2.38})$
- Minimum spanning tree
- Shortest path
- ...

Problems that have been proven to be intractable (2/3)

Two types of problems

- Problems that require a non polynomial amount of output.
- Problems that are not asking for a non polynomial amount of output, but such that we can prove that the problem cannot be solved in polynomial time
  - Undecidable Problems: it can be proven that algorithms that solve them cannot exist
    - Example: Halting problem (Alan Turing, 1936)
  - Decidable Problems that are intractable
    - Example: Presburger Arithmetic (Fischer and Rabin, 1974. Also in the book of Hopcroft and Ullman, 1979)

Problems that have not been proven to be intractable but for which polynomial-time algorithms have never been found (3/3)

- 0-1 Knapsack problem
- Traveling salesman problem
- k-Coloring problem for $k \geq 3$
- ...

Decision, search, and optimization problems

- Feasible linear program: Given a matrix A, vectors b and c, and a value k, is there a real value vector x, such that $Ax \leq b$ and $cx \geq k$? **Decision Problem**
- Linear Program: Given a matrix A, and vectors b and c, and a value k, find a real-valued vector x such that $Ax < b$ and $cx > k$? **Search Problem**
- Optimal Linear Program: Given a matrix A, vectors b and c, and a value k, find a real value vector x, such that $Ax \leq b$ that maximizes the value of cx? **Optimization Problem**

Clearly: **Optimization** harder than
Search harder than
Decision

Decision, search, and optimization problems

- Spanning Tree 1: Given a weighted graph $G = (V, E)$, a weight function w, and a bound k, **is there** a spanning tree of G of total weight less than k? **Decision Problem**
- Spanning Tree 2: Given a weighted graph $G = (V, E)$, a weight function w, and a bound k, **find** a spanning tree of G of total weight less than k. **Search Problem**
- Spanning Tree 3: Given a weighted graph $G = (V, E)$, a weight function w, and a bound k, **find** a minimum weight spanning tree of G? **Optimization Problem**

Clearly: **Optimization** harder than
Search harder than
Decision

P, NP, co-NP Classes (1/4)

- Class P. Set of yes/no problems that can be solved in polynomial time, on a deterministic Turing machine.
- In other words, a problem is in the class P if it is a decision problem and there exists an algorithm that solves any instance of size n in $O(n^k)$ time, for some integer k. (Strictly, n must be the number of bits needed for a *reasonable* encoding of the input.)
- So P is just the set of tractable decision problems: the decision problems for which we have polynomial-time algorithms.

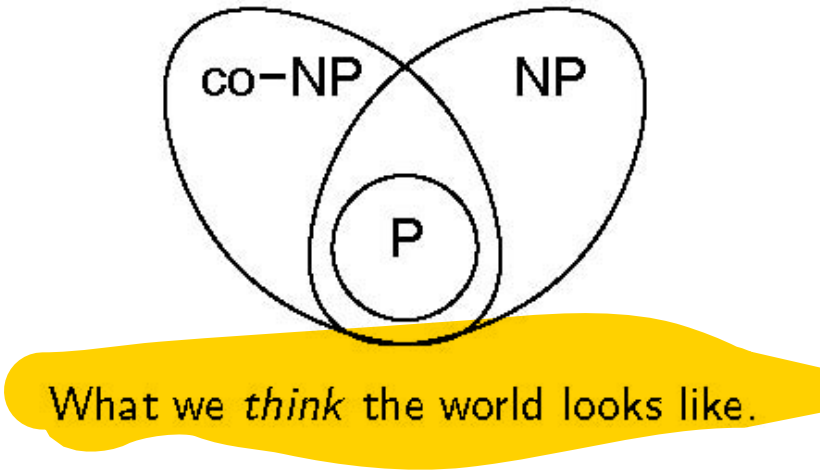
P, NP, co-NP Classes (2/4)

- A nondeterministic algorithm is an algorithm that, even for the same input, can exhibit different behaviors on different runs
- A polynomial-time nondeterministic algorithm is a nondeterministic algorithm whose verification stage is a polynomial-time algorithm.
- Class NP (Nondeterministic Problem). Set of yes/no problems that can be solved by polynomial-time nondeterministic algorithms, i.e., with the following property: If the answer is yes, then there is a proof of this fact that can be checked in polynomial time, on a deterministic Turing machine. Intuitively, NP is the class of problems where we can verify a YES answer quickly if we have the solution in front of us.
  - Example: Traveling Salesman Decision Problem.
- Two phases
  - the first consists of a guess about the solution, which is generated in a non-deterministic way
  - the second phase consists of a deterministic algorithm that verifies or rejects the guess as a valid solution to the problem

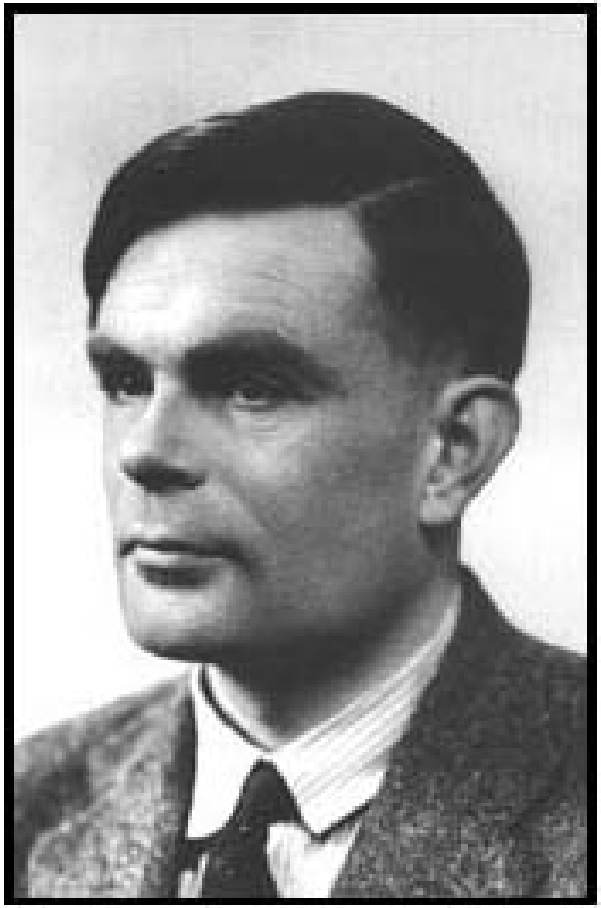
P, NP, co-NP Classes (3/4)

- Class co-NP. Exact opposite of NP. If the answer to a problem in co-NP is NO, then there is a proof of this fact that can be checked in polynomial time, on a deterministic Turing machine.
  - Example: Tautology Problem: Determine whether a given Boolean formula is a tautology; that is, whether every possible assignment of true/false values to variables yields a true statement.

P, NP, co-NP Classes (4/4)



Turing Machine (1/3)



Alan Turing (1912-1954)

Turing Machine (2/3)

Turing sought the most primitive model of a computing device

- Device should have some basic capabilities as human computer!

NO NEED TO STUDY THIS AND THE TOPICS AFTER

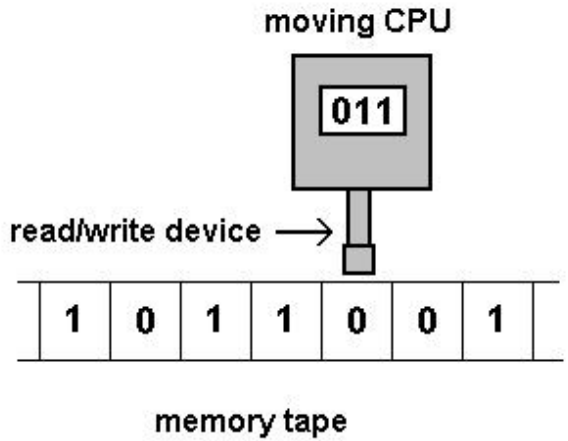
Tape

- Stores input, output, and intermediate results
- 1 arbitrarily long strip, divided into cells
- Finite alphabet of symbols

...	A	#	C	&	Y	B	A	#	...

Tape head

- Points to one cell of tape
- Reads a symbol from active cell
- Overwrites a symbol to active cell
- Moves left or right one cell at a time



Turing Machine (3/3)

- In a **deterministic Turing machine**, the set of rules prescribes at most one action to be performed for any given situation.
- A **non-deterministic Turing machine (NTM)**, by contrast, may have a set of rules that prescribes more than one action for a given situation.
- **Example**: a non-deterministic Turing machine may have both
  - If you are in state 2 and you see an A, change it to a B and move left
  - If you are in state 2 and you see an A, change it to a C and move right.

  in its rule set.
- At each step in the computation performed by an NTM, instead of having a single transition rule, there are multiple rules that can be invoked. To determine if the NTM accepts or rejects, you look at all possible branches of the computation. So if there are, say, exactly 2 transitions to choose from at each step, and each computation branch has a total of N steps, then there will be 2N total branches to consider.

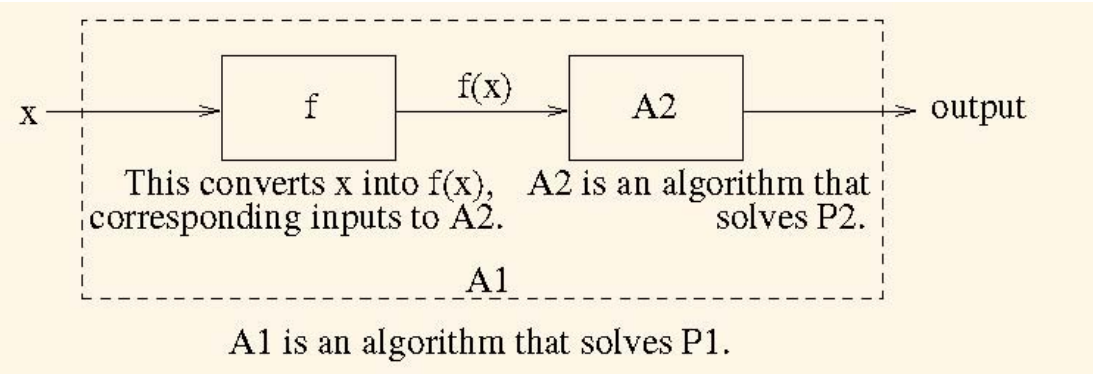
Problem Reduction

Suppose you have a problem P2 which you know how to solve, e.g., by using algorithm A2.

- Suppose you are given another problem P1 that seems similar to P2. How might you solve P1?
  - You could try to solve P1 from scratch.
  - You could try to borrow elements of A2.
  - You could try to find a reduction from P1 to P2.
- A reduction of P1 to P2:
  - Transforms inputs to P1 into inputs to P2;
  - Runs A2 (which solves P2) as a *black-box*; and
  - Interprets the outputs from A2 as answers to P1.

Reduction (2/3)

More formally,
A problem **P1 is reducible to a problem P2** if there is a function f that takes any input x to P1 and transforms it to an input $f(x)$ of P2, such that the solution to P2 on $f(x)$ is the solution to P1 on x.



Reduction: An example (3/3)

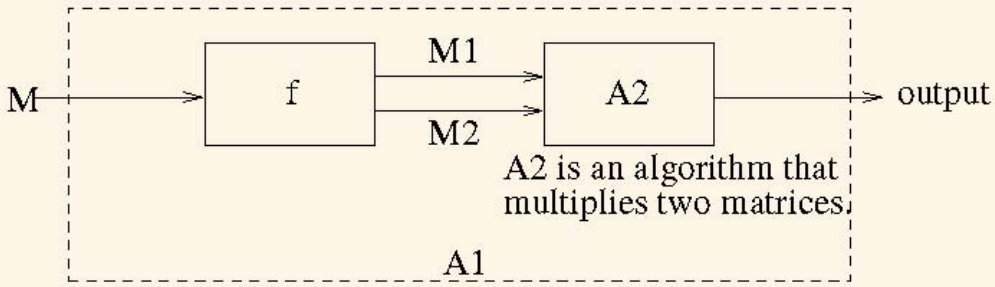
Matrix Multiplication

Parameters: Two matrices, M1 and M2
Returns: The result of multiplying M1 and M2 together.

Squaring a Matrix

Parameters: A matrix, M.
Returns: The result of squaring M.

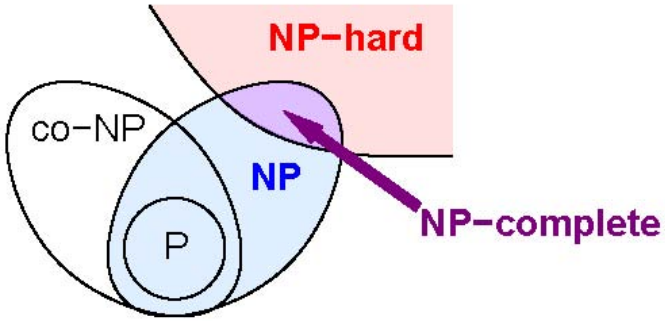
How to do it using a reduction:



A1 is an algorithm that squares a matrix.

NP-hard, NP-complete

- NP-hard. Problem P_i is NP-hard if *every* problem in NP is polynomial-time reducible to P_i: for every $P_j \in$ NP, $P_j \overset{\text{reduc}}{\rightarrow} P_i$
- NP-complete. Problem P_i is NP-complete if
  - P_i is NP-hard and P_i is an element of NP.
- NP-hard stands for at least as hard as NP (but not necessarily in NP)
- NP-complete problems are the hardest problems in NP.
- Every NP-complete problem can be translated in deterministic polynomial time to every other NP-complete problem.
- So, if there is a P-time to one NP-complete problem, there is a P-time solution to every NP-complete problem.



More of what we *think* the world looks like.

Basic Proof Strategy - NP-complete

NP-completeness is a good news/bad news situation. If it is NP-hard, that means that it is pretty hard. But, it could be worse!

So, a typical NP-completeness proof consists of two parts:

1. Prove that the problem is in NP. The usual strategy is to show that a solution can be verified in polynomial time with a deterministic Turing machine (DTM).
2. Prove that the problem is at least as hard as other problems in NP.

NP-completeness by reduction

Typical method: Reduce a known NP-complete problem P1 to the new problem P2. A reduction is a polynomial-time translation of the problem, call it r. If w is an instance of problem P1 (e.g., a string), then $r(w)$ is an instance of problem P2.

r must have two properties:

- it preserves the answer. So $w \in P1$ iff $r(w) \in P2$.
- $r(w)$ can be computed in time polynomial in $|w|$.

In practice, there are now thousands of known NP-complete problems. A good technique is to look for one similar to the one you are trying to solve.

Warning

Make sure you are reducing the known problem to the unknown problem!

SAT

Reduction proofs require that there is a known NP-complete problem. Where does the first one come from?

The primordial NP-Complete problem is satisfiability of a propositional logic formula (SAT)

Cook's Theorem: SAT is NP-complete.

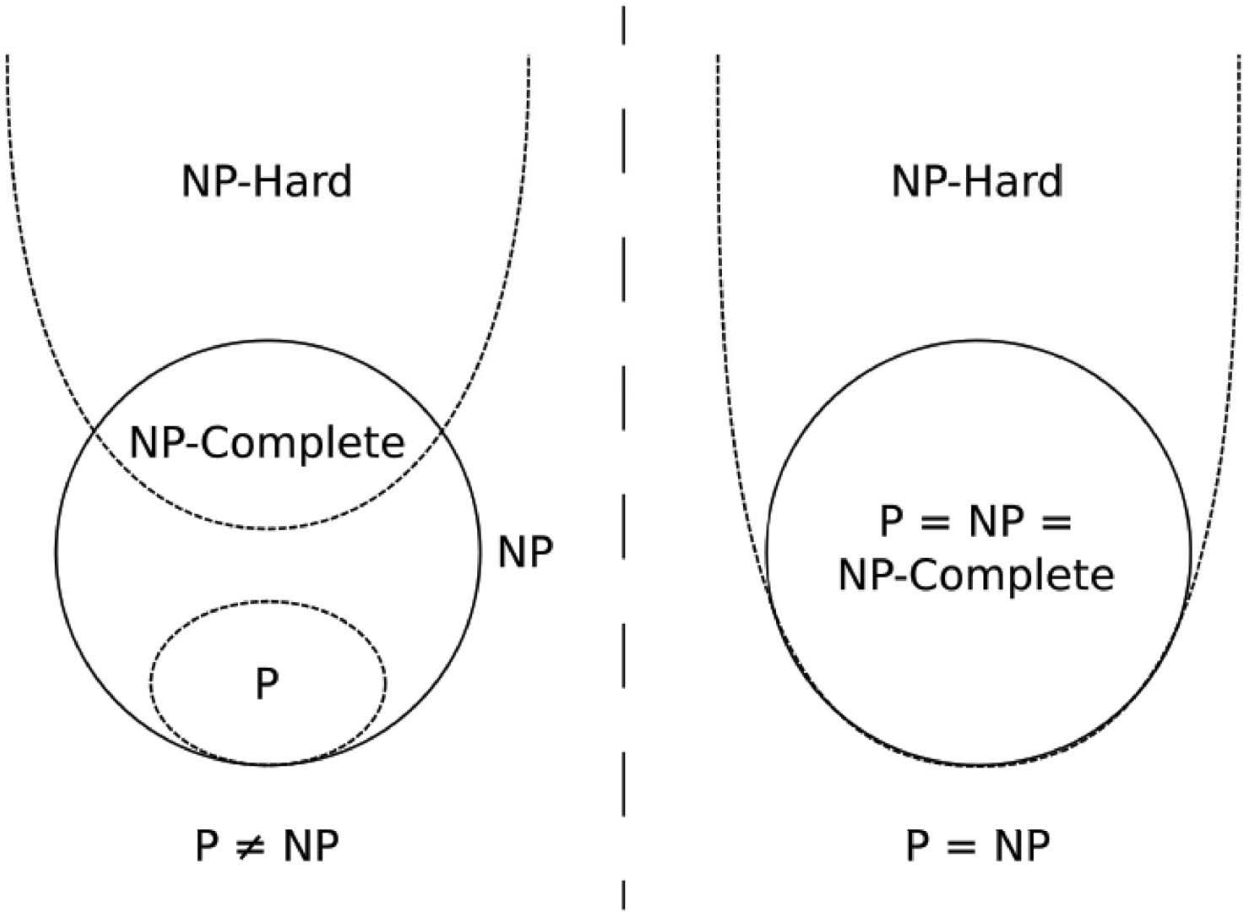
Propositional logic:

- Propositional variables which can take Boolean values.
- Propositional connectives $\neg\alpha, \alpha \vee \beta, \alpha \wedge \beta$

Example: $p \vee \neg q \wedge r$

Truth assignment: A function that assigns a truth value, $\{T, F\}$ to each propositional variable. A truth assignment that makes a propositional formula true is said to satisfy it.
A propositional formula is satisfiable if $\exists$ a satisfying assignment for it.

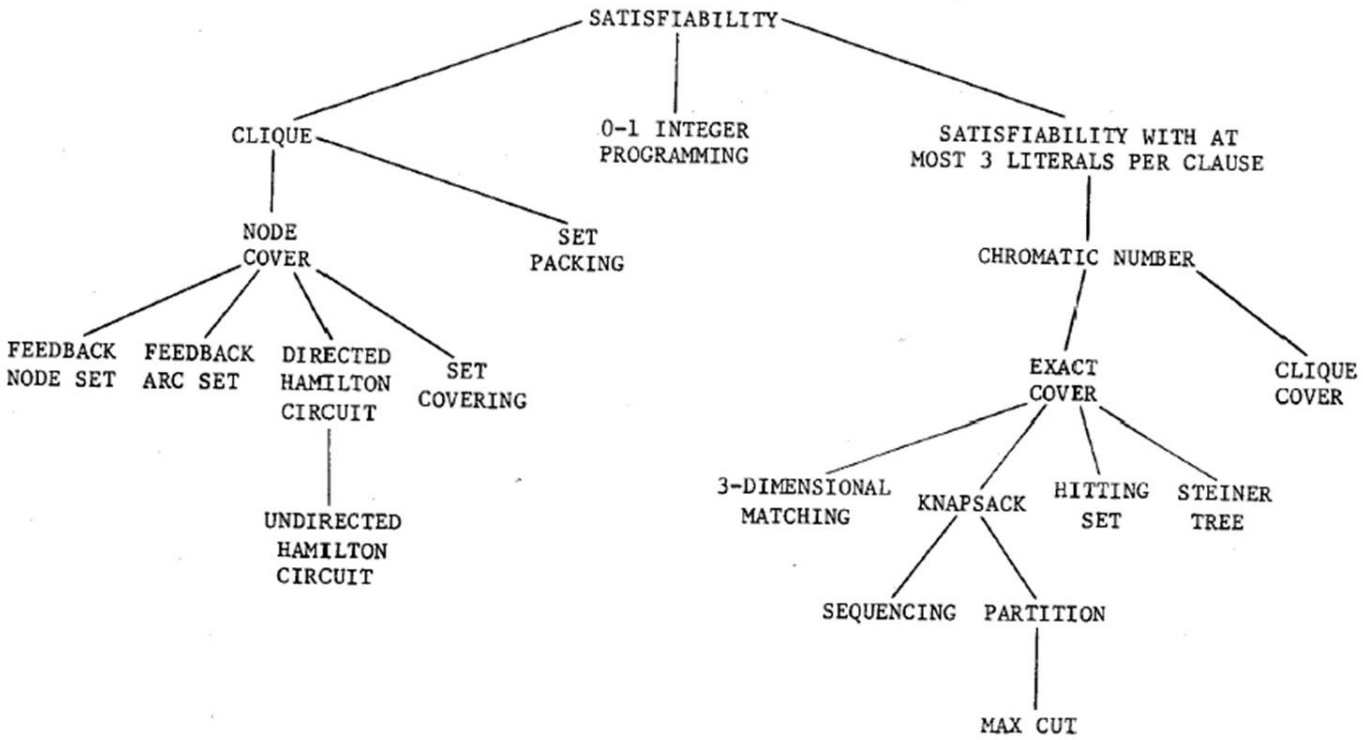
P, NP, NP-complete, and NP-hard set of problems



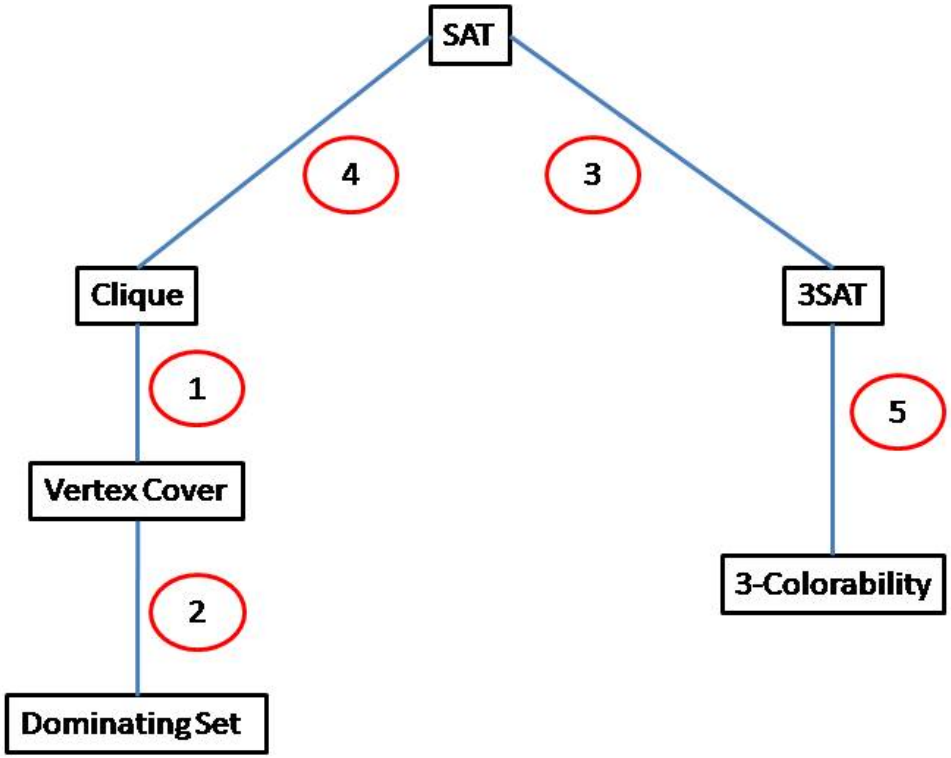
Solving Intractable Problems

- Seek to obtain as much improvement as possible and live hopefully! For example, in the coloring problem, eliminating symmetry may help further. Incorporating rules-of-thumb (*heuristics*) to dynamically decide what to try next may also help. All of these ideas try to make the algorithm work well in practice, on typical instances, while acknowledging that exponential cases are still possible.
- Solve simpler/restricted versions of the problem. Maybe a solution to a slight variant of the problem would still be useful to you, while possibly avoiding exponential complexity.
- Use a polynomial-time probabilistic algorithm: one which gives the right answer only with very high probability. So you are giving up on program correctness, in the interests of speed.
- For optimization problems, use a polynomial-time approximation algorithm: one which is not guaranteed to find the best answer.

Karp's Problems



The order of NP-completeness proofs



Vertex Cover (1/5)

Vertex Cover - Definition

A vertex cover (sometimes node cover) of a graph is a set of vertices such that each edge of the graph is incident to at least one vertex of the set.

Vertex Cover - Decision Problem

Given an indirected graph $G = (V, E)$ and an integer k, determine whether G has a vertex cover containing $\leq k$ vertices

Theorem

The vertex cover problem is NP-complete

Vertex Cover (2/5)

- **Step 1.** The vertex cover problem belongs to NP, since we can guess a cover of size $\leq k$ and check it easily in polynomial time.
- **Step 2.** To prove that the vertex-cover problem is NP-complete, we have to reduce an NP-complete problem to it. We choose the clique problem.

Vertex Cover: Reduction clique problem $\mapsto$ vertex cover (3/5)

- $G = (V, E)$ undirected graph, C a clique in G (i.e., a subgraph of G such that all vertices in C are connected to all other vertices in C).
- We have to transform an **arbitrary** instance of the clique problem into an instance of the vertex cover problem, such that the answer to the clique problem is yes if and only if the answer to the corresponding vertex cover problem is positive.

Vertex Cover (4/5)

- $G = (V, E)$ undirected graph and K an **arbitrary** instance of the clique problem.
- $\overline{G} = (V, \overline{E})$ the **complement graph** of G.
- **Claim**: the clique problem is reduced to the vertex-cover problem represented by $\overline{G}$ and $n - k$ (where $n = |V|$).
- Suppose $C = (U, F)$ is a clique in G. The set of vertices $V \setminus U$ covers all the edges of $\overline{G}$, because in $\overline{G}$ there are no edges connecting vertices in U (they are all in G). Thus, $V \setminus U$ is a vertex cover in $\overline{G}$. Therefore, if G has a clique of size k, then $\overline{G}$ has a vertex cover of size $n - k$.

Vertex Cover (5/5)

- **Conversely**, let D be a vertex cover in $\overline{G}$.
- D covers all the edges in $\overline{G}$, so in $\overline{G}$ there could be no edges connecting vertices in $V \setminus D$. Thus, $V \setminus D$ generates a clique in G. Therefore, if there is a vertex cover of size k in $\overline{G}$, then there is a clique of size $n - k$ in G.
- This **reduction can be performed in polynomial time**, since it requires only the construction of $\overline{G}$ from G.

DOMINATING SET (1/5)

INSTANCE: Graph $G = (V, E)$, positive integer $K \leq |V|$.

QUESTION: Is there a subset $V' \subseteq V$ such that $|V'| \leq K$ and such that every vertex $v \in V' \setminus V$ is joined to at least one member of V' by an edge in E?

Theorem

The dominating set problem is NP-complete

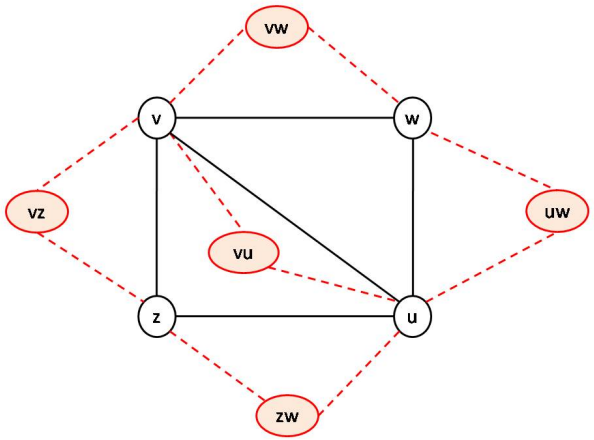
Dominating Set (2/5)

- **Step 1.** The dominating set problem belongs to NP, since we can guess a dominating set of size $\leq k$ and check it is a dominating set easily in polynomial time.

- **Step 2.** To prove that the dominating set problem is NP-complete, we have to reduce an NP-complete problem to it. We choose the vertex cover problem.

Dominating Set (3/5)

- **Arbitrary** instance (G, k) of the vertex cover problem.
- **Goal.** Construct a new graph G' that has a dominating set of a certain size if and only if G has a vertex cover of size $\leq k$.
- Start with G, and remove all isolated vertices (not necessary in any VS since they have no edges)
- Add $|E|$ new vertices and $2 \times |E|$ new edges to it in the following way:
  - For each edge $\{v, w\}$ of G, add a new vertex vw + two new edges $\{v, vw\}$ and $\{w, vw\}$
  - In other words, we transform every edge into a triangle

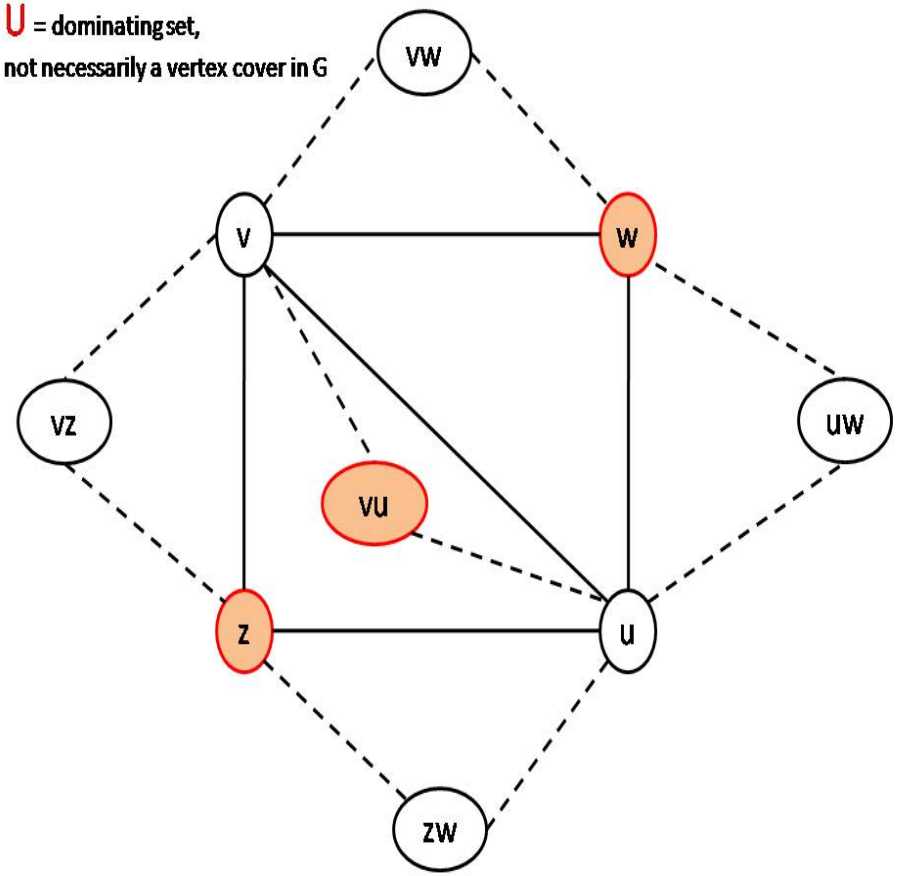


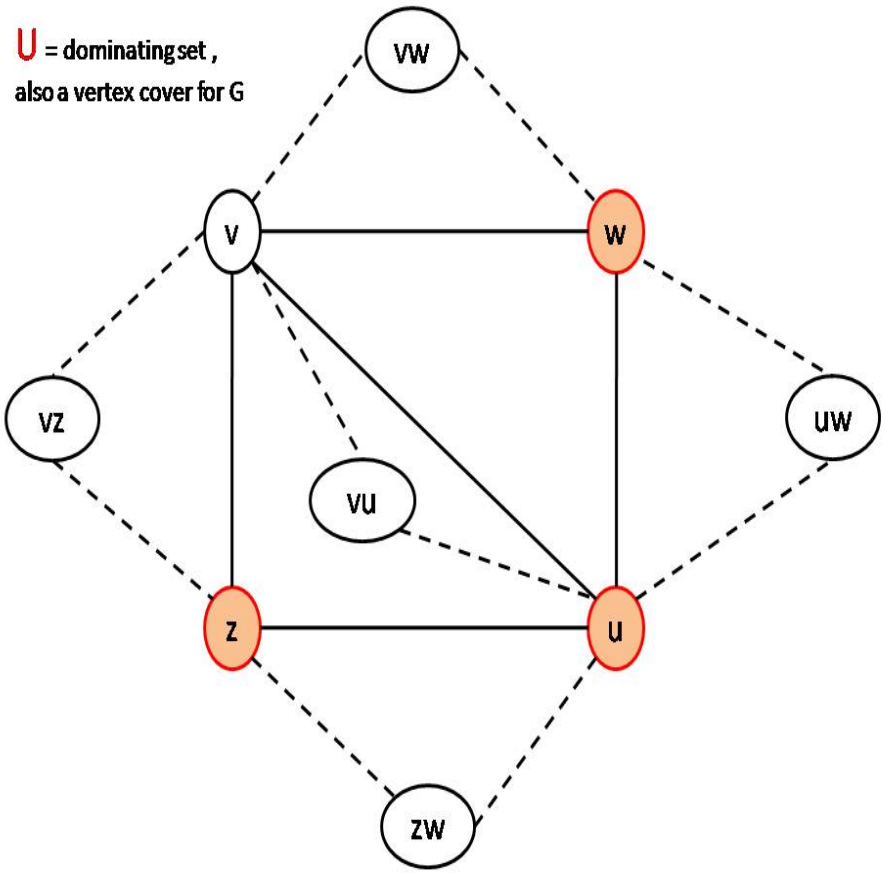
Dominating Set (4/5)

- **Claim**: G' has a dominating set of size m if and only is G has a vertex cover of size m

$\Rightarrow$ Suppose U is a dominating set in G' of size $\leq k$.

- We first get ride of vertices in U corresponding to edges of G. Then, if $v_e \in U$ for some edge $\{u, w\} \in E$, then we replace it by u. Note that:
  - The size of U does not increase in that process
  - The neighbors of a vertex v_e for $e = \{u, w\}$ are only u and w. Thus, by removing v_e from U, the only vertices that might remained uncovered are u and w, but they are covered by u. Therefore U remains a dominated set.
- $\forall e = \{u, w\} \in E$, U must contain u or w (otherwise v_e has no neighbor in U). Thus U is a vertex cover in G.


U = dominating set,
not necessarily a vertex cover in G
VW
V
W
VZ
UW
VU
Z
U
ZW


U = dominating set ,
also a vertex cover for G
VW
V
W
VZ
UW
VU
Z
U
ZW

Dominating Set (5/5)

$\Leftarrow$ Let U be a vertex cover of G.

- We show that it is also a dominating set in G'. For $v \in V' \setminus U$, there are two possibilities.
  - If $v \in V$ then, since it is not an isolated vertex, there is an edge $\{u, v\} \in E$. Since U is a vertex cover and $v \notin U$, we have $u \in U$, so v has a neighbor in U.
  - $v = v_e$ for some edge $e = \{u, w\} \in E$. Since U is a vertex cover, $u \in U$ or $w \in U$, so v has a neighbor in U.
- This **reduction can be performed in polynomial time**, since it requires only the construction of G' from G.

3SAT (1/5)

3SAT

Given a Boolean expression in CNF such that each clause contains exactly three variables, determine whether it is satisfiable

Theorem

3SAT is NP-complete

3SAT (1/5)

3SAT

Given a Boolean expression in CNF such that each clause contains exactly three variables, determine whether it is satisfiable

Theorem

3SAT is NP-complete

3SAT (2/5)

- A solution to 3SAT can be used to solve the regular SAT.
- Firstly, 3SAT clearly belongs to NP: We can guess a truth assignment and verify that it satisfies the expression in polynomial time.
- E arbitrary instance of SAT. Let us show that we can replace each clause of E with several clauses, each of which has exactly three variables
  - $C = x_1 \vee x_2 \vee ... \vee x_k$ arbitrary clause of E such that $k \geq 4$ (positive literal for convenience of notation)
  - Introduce new variables $y_1, y_2, ..., y_{k-3}$
  -

$$C' = (x_1 \vee x_2 \vee y_1) \wedge (x_3 \vee \overline{y}_1 \vee y_2) \wedge (x_4 \vee \overline{y}_2 \vee y_3) \wedge \\ ... \wedge (x_{k-1} \vee x_k \vee \overline{y}_{k-3})$$

3SAT (3/5)

- $C = x_1 \vee x_2 \vee ... \vee x_k$ arbitrary clause of E such that $k \geq 4$ (positive literal for convenience of notation)
- $C' = (x_1 \vee x_2 \vee y_1) \wedge (x_3 \vee \overline{y}_1 \vee y_2) \wedge (x_4 \vee \overline{y}_2 \vee y_3) ... \wedge (x_{k-1} \vee x_k \vee \overline{y}_{k-3})$
- **Claim**: C' is satisfiable if and only if C is satisfiable.
  - **If C is satisfiable**, then one of the x_i's must be set to 1. In that case, we can set the values of the y_i's in C' such that all clauses in C' are satisfied as well. Example: If $x_3 = 1$, then we set $y_1 = 1$ (takes care of first clause), $y_2 = 0$ (takes care of second clause) and the rest of the y_is to 0. In general: if $x_i = 1$, set $y_1, y_2, ..., y_{i-2}$ to be 1, and the rest to be 0, which satisfies C'.
  - **Conversely, if C' is satisfiable**, at least one of the x_is must be 1. Indeed, if all x_is are 0, then the expression becomes $(y_1) \wedge (\overline{y}_1 \vee y_2) \wedge (\overline{y}_2 \vee y_3) ... \wedge (\overline{y}_{k-3})$: this is unsatisfiable.

3SAT (4/5)

Similar transformation for clauses with two or one literals

3SAT (5/5)

Thus, we have reduced a general instance of SAT into an instance of 3SAT such that one instance is satisfiable if and only if the other one is.

The reduction can clearly be done in polynomial time

Clique (1/5)

Clique

Given an undirected graph $G = (V, E)$ and an integer k, determine whether G contains a clique of size $\geq k$.

Theorem

The clique problem is NP-complete

Clique (2/5)

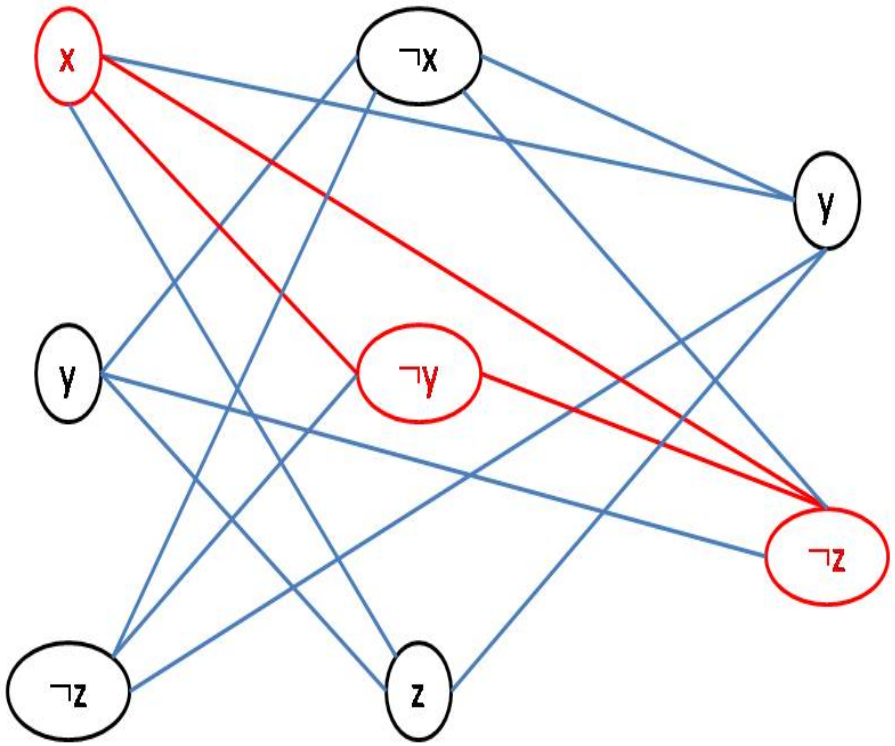
- The clique problem belongs to NP since we can guess a subset of $\geq k$ vertices and check that it is a clique in polynomial time
- Reduction of SAT to the clique problem
- E arbitrary Boolean expression in CNF: $E = E_1 \wedge E_2 \wedge ... \wedge E_m$, and $E_i = (x \vee y \vee z \vee w)$ (four variables only for illustration purposes)
- Building the graph G: A vertex for each appearance of the variable.
- **Question**: How to connect these vertices such that G contains a clique of size $\geq k$ if and only if E is satisfiable.
- **Remark**: Free to choose the value of k

Clique (3/5)

- Graph G
  - **Vertices**: A vertex for each appearance of the variable + a "column" of four vertices with the variables of E_i.
  - **Edges**: Vertices from the the same column (i.e., vertices associated with the variables of the same clause) are not connected. Vertices from different columns are almost always connected unless they correspond to the same variable appearing in complementary form.
- G can be constructed in polynomial time

Clique (4/5)

An example of the clique reduction for the expression $(x \vee y \vee \overline{z}) \wedge (\overline{x} \vee \overline{y} \vee z) \wedge (y \vee \overline{z})$



Clique (5/5)

Claim: G has a clique of size $\geq m$ if and only if E is satisfiable

The reduction can clearly be done in polynomial time

3-Coloring (1/6)

3-Coloring

Given an undirected graph $G = (V, E)$, determine whether G can be colored with three colors

Theorem

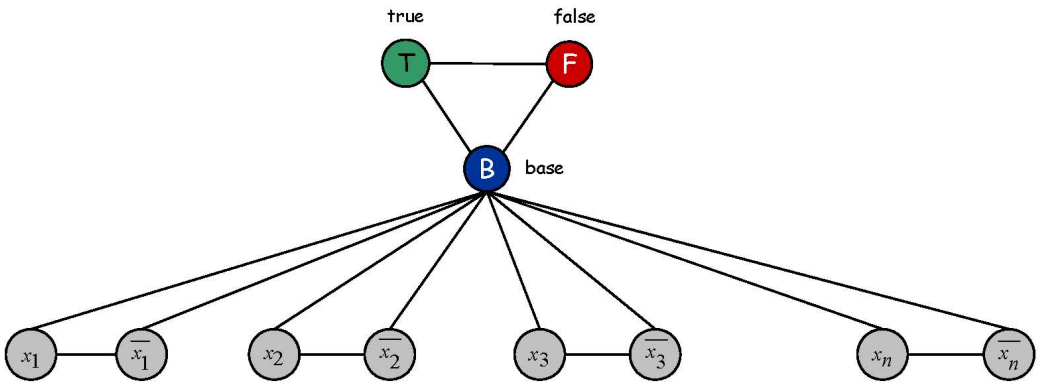
3-coloring is NP-complete

3-Coloring (2/6)

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

Construction

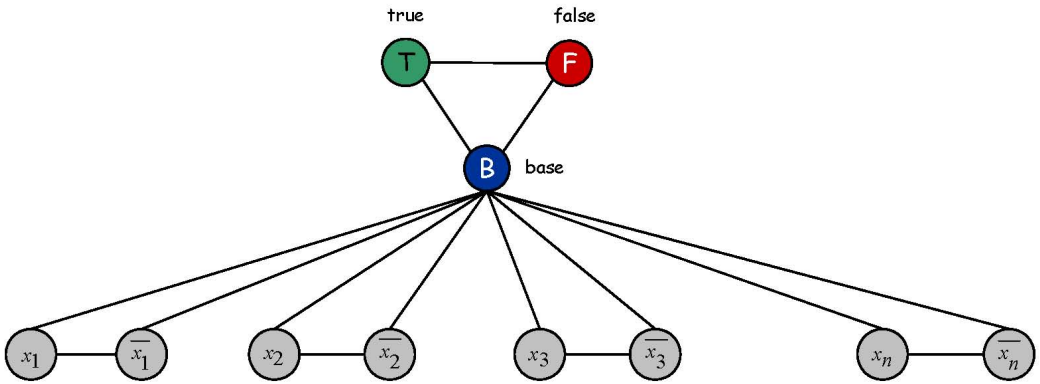
- **(i)** For each literal, create a node.
- **(ii)** Create 3 new nodes T, F, B; connect them in a triangle, and connect each literal to B.
- **(iii)** Connect each literal to its negation.
- **(iv)** For each clause, add gadget of 6 nodes and 13 edges.



3-Coloring (3/6)

Suppose graph is 3-colorable.

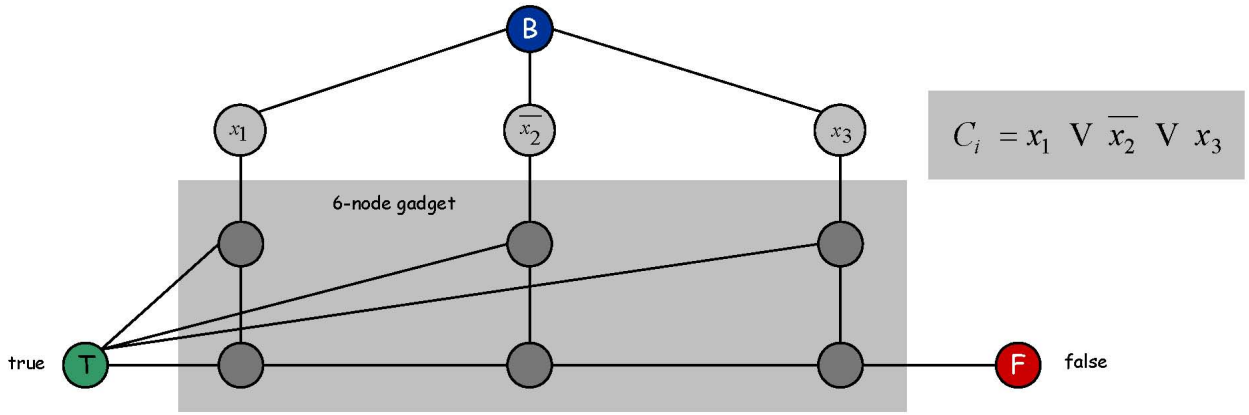
- Consider assignment that sets all T literals to true.
- **(ii)** ensures each literal is T or F.
- **(iii)** ensures a literal and its negation are opposites.



3-Coloring (4/6)

Suppose graph is 3-colorable.

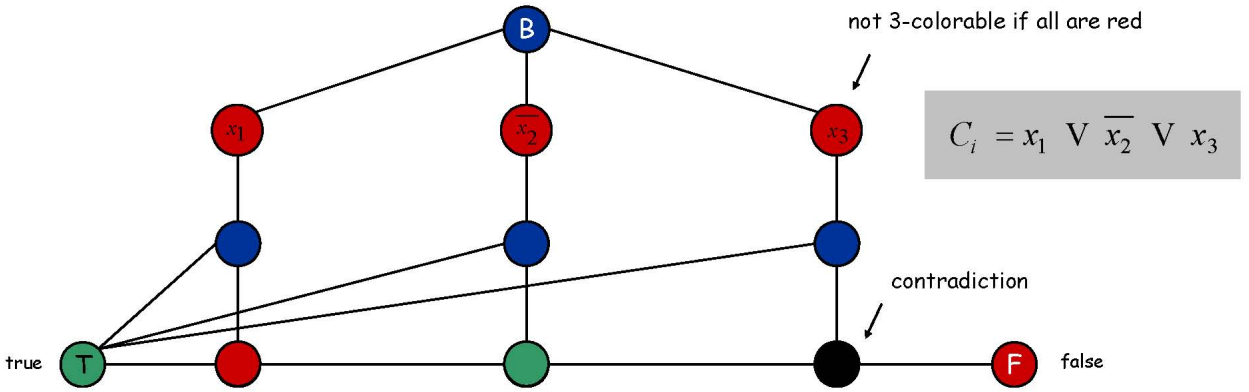
- Consider assignment that sets all T literals to true.
- **(ii)** ensures each literal is T or F.
- **(iii)** ensures a literal and its negation are opposites.
- **(iv)** ensures at least one literal in each clause is T.



3-Coloring (5/6)

Suppose graph is 3-colorable.

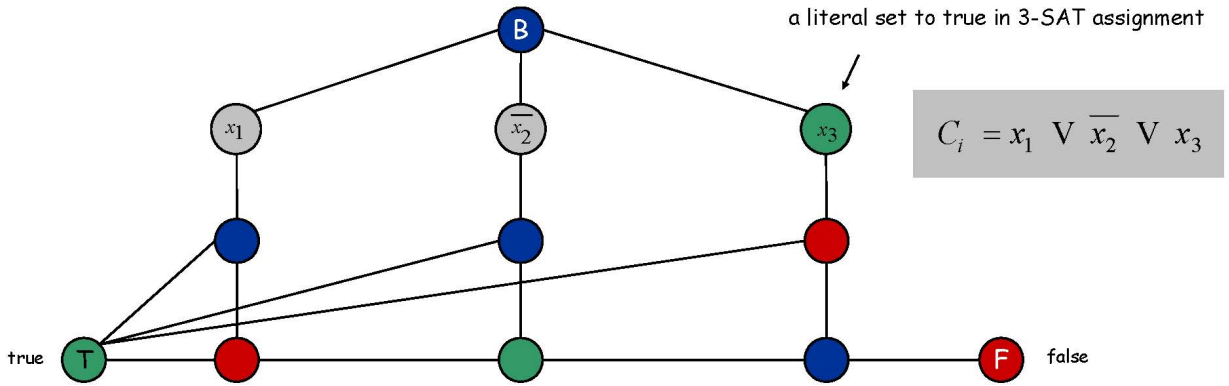
- Consider assignment that sets all T literals to true.
- **(ii)** ensures each literal is T or F.
- **(iii)** ensures a literal and its negation are opposites.
- **(iv)** ensures at least one literal in each clause is T.



3-Coloring (6/6)

Suppose 3-SAT formula Φ is satisfiable.

- Color all true literals T.
- Color node below green node F, and node below that B.
- Color remaining middle row nodes B.
- Color remaining bottom nodes T or F as forced.



1. Cormen, T.H., C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, Second Edition, McGraw Hill, 2001.
2. Garey, M.R., and D.S. Johnson, *Computers and Intractability- A Guide to the Theory of NP-Completeness*, Freeman, 1979.