Time Complexity Measure

- A language L on an alphabet Σ is *polynomial-time decidable* if there is a Turing machine $\mathcal{M}$ that accepts L, and a polynomial $p(n)$ such that the number of steps in an accepting computation by $\mathcal{M}$ with input x is $\leq p(|x|)$.
- The class of all polynomial-time acceptable languages is denoted as $\mathbf{P}$.
- A *total* function f on Σ^* is *polynomial-time computable* if there is a Turing machine $\mathcal{M}$ that computes f and a polynomial $p(n)$ such that the number of steps in the computation by $\mathcal{M}$ with input x is $\leq p(|x|)$.
- In general, we do not have to deal with *all* possible values of $|x|$. Requiring for *all but a finite number of* $|x|$ would be sufficient. Suppose these finite number of cases use c_1, c_2, ..., c_m steps, then $p(|x|) + \max(c_1, c_2, \ldots, c_m)$ can be used for measuring the number of steps.
- For polynomial-time computable/decidable languages, the number of steps can be expressed as $O(|x|^r)$ for some r, where r is the degree of polynomial $p(n)$.

Cook-Karp Thesis

- **The problem of determining membership of strings in a given language L is *tractable* if and only if $L \in \mathbf{P}$.**

  That is, the number of steps to solve the given problem is bounded by a polynomial.

- **Let $L \in \mathbf{P}$, let f be a polynomial-time computable function on Σ^* and let**

$$Q = \{x | f(x) \in L\}$$

  Then, $Q \in \mathbf{P}$.

  Proof: Since $L \in \mathbf{P}$, there is a Turing machine $\mathcal{M}_1$ that accepts a string in L in $O(|x|^r)$ steps. Since f is polynomial-time computable, there is a second Turing machine $\mathcal{M}_2$ to compute it in $O(|x|^s)$ steps.

  Construct a Turing machine that runs $\mathcal{M}_2$ first converting the input x to $f(x)$ and runs $\mathcal{M}_1$ to check if $f(x)$ is acceptable. This takes $O(|x|^{\max(r,s)})$ steps. Therefore, Q is polynomial-time computable.

- **If f and g are polynomial-time computable and $h(x) = f(g(x))$, then h is also polynomial-time computable.**

  What is the required number of steps?

The Class NP

- A language L is in the class **NP** if
  1. there is a *nondeterministic* Turing machine $\mathcal{M}$ that accepts L, and
  2. a polynomial $p(n)$ such that for each $x \in L$ there is an accepting computation of m steps with $m \leq p(|x|)$.

  NP is for *nondeterministic polynomial*.

- **Theorem: $\mathbf{P} \subset \mathbf{NP}$. If $L \in \mathbf{NP}$, then L is recursive.**

  Proof: Since deterministic Turing machines are special cases of nondeterministic Turing machines, a language in **P** is also in **NP**.

  To prove L is recursive, we need a program to determine if a given x is a member of L.

  Starting with the initial configuration, we can list all possible execution paths whose length is $\leq p(|x|)$, where x is a given string. Therefore, $x \in L$ if and only if there is a path from our list that accepts x.

  So, languages in **NP** are computable.

The Satisfiability Problem

- **Given a Boolean expression, is it possible to assign truth values to its variables so that the result is TRUE?**
- **CNF Form: A Boolean expression is in CNF form (*i.e.*, Conjunctive Normal Form) if it is written as $E_1 \& E_2 \& \cdots \& E_n$, where each E_i has a form of $v_{i1} \vee v_{i2} \vee \cdots \vee v_{ik}$ and each v_{ij} is a variable x_{ij} or its negation $\neg x_{ij}$.**

$$(p \vee q \vee r) \& (\neg q \vee r) \& (p \vee \neg s) \& (\neg p \vee \neg s \vee \neg q)$$

The Satisfiability Problem

- **The Satisfiable (even in CNF form) is in NP.**

  Proof: Because the Turing machine is nondeterministic, we first "guess" a solution and then "verify" it.

  1. The "guessing" phase uses $O(|n|)$ steps, where $|n|$ is the encoded length of the input Boolean expression n.
  2. The "verifying" phase is actually a parser that evaluates a Boolean expression. A "good" parser takes $O(|n|)$ steps to evaluate an expression; a not-so-good one, which always starts from the beginning, uses $O(|n|^2)$ steps (*i.e.*, it takes $|n|$ passes each of which scans the string of length $|n|$).
  3. Thus, determining if n is satisfiable takes no more than $O(|n|)$ steps. Note that for a nondeterministic Turing machine, as long as a "guess" could make the expression satisfiable, the given expression is accepted.

- If Boolean expression n has k variables, there are 2^k different combinations of the possible values for the variables. To determine if n is satisfiable, all $O(2^k)$ cases must be checked.

Is P = NP?

- We know many problems are in **P**. Example: sorting, hashing, string matching and so on.
- We know a problem, the Satisfiability problem, is in **NP**.
- Is there a polynomial-time Turing machine (or algorithm) for the satisfiability problem?

  Don't know!
- **The biggest problem ever in computer science: $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$**
  1. If $\mathbf{P} = \mathbf{NP}$, then many problems are not so difficult.
  2. If $\mathbf{P} \neq \mathbf{NP}$, some problems are definitely more difficult than the others.

Polynomial-Time Reducibility

- **Let L and Q be two languages. If there is a polynomial-time computable function f such that**

  $$x \in Q \Leftrightarrow f(x) \in L \quad (i.e., Q = \{x \mid f(x) \in L\})$$

  we shall say Q is polynomial-time reducible to L, denoted as $Q \leq_{\mathbf{p}} L$.

  Please compare with many-to-one reducibility $\leq_{\mathrm{m}}$. Thus, Q is no more difficult than L.

- **Transitivity: If $P \leq_{\mathbf{p}} Q$ and $Q \leq_{\mathbf{p}} R$, then $P \leq_{\mathbf{p}} R$.**

- **NP-Hard: A language L is NP-hard if for every language $Q \in \mathbf{NP}$, we have $Q \leq_{\mathbf{p}} L$.**

  That is, language L is more difficult than all **NP** languages. *Note that L does not have to be a language in* **NP**.

- **NP-Complete: A language L is NP-Complete if and only if $L \in \mathbf{NP}$ and L is NP-hard.**

  Therefore, L is the most difficult (or the hardest) language of all **NP** languages. But, we shall show all **NP** languages are *equally* difficult (or easy).

  Compare with m-complete.

Cook's Theorem

- **Cook's Theorem: The Satisfiability problem is NP-complete.**

  Proof: Too complicated.

- **If there is an NP-complete language L such that $L \in \mathbf{P}$, then P=NP.**

  Proof: Suppose language L is **NP-complete** and $L \in \mathbf{P}$. Then, for every **NP** language Q, there exists a polynomial-time computable function f such that

  $$x \in Q \Leftrightarrow f(x) \in L$$

  Hence, Q is in **P**.

- **P=NP $\Leftrightarrow$ the Satisfiability problem is in P.**

  Proof: ($\Rightarrow$) Obvious.

  ($\Leftarrow$) If the Satisfiability problem is in **P**, since it is also **NP-complete**, the above yields **P=NP**.

Proving NP-Completeness

- **If Q is an NP-complete language and $Q \leq_{\mathbf{p}} L$, then L is NP-hard.**

  Proof: We want to show that every **NP** language can be polynomial-time reducible to L.

  1. Since Q is **NP-complete**, for any **NP** language R we have $R \leq_{\mathrm{p}} Q$.
  2. Since $Q \leq_{\mathrm{p}} L$, we have $R \leq_{\mathrm{p}} L$.
  3. Therefore, L is **NP-hard**.

- **If Q is an NP-complete language, $L \in$ NP and $Q \leq_{\mathbf{p}} L$, then L is also NP-complete.**

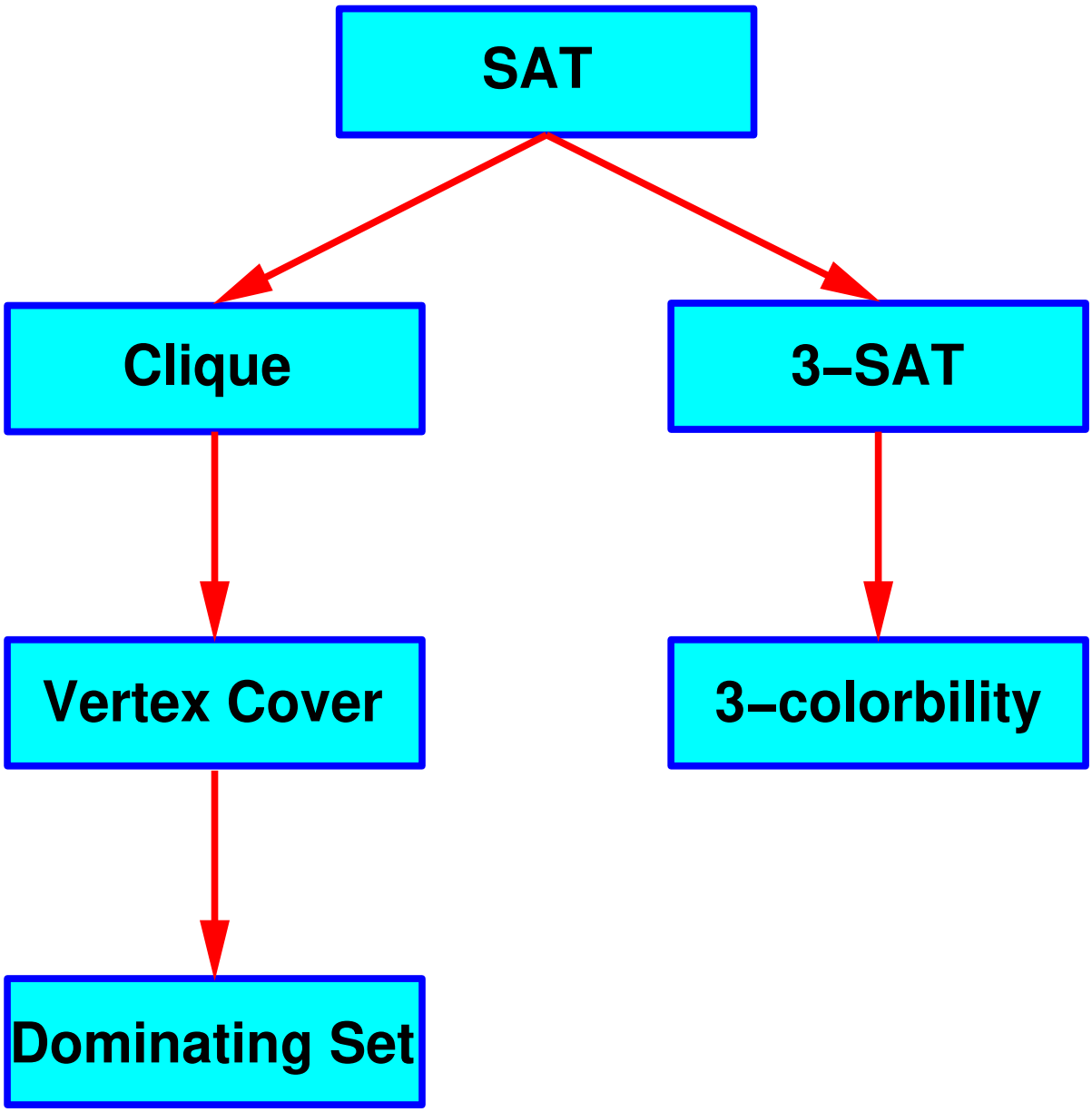
  Proof: Since L is **NP** and **NP-hard**, it is **NP-complete** by definition.

- To show a language L is **NP-complete**, we need to

  1. find an **NP-complete** language Q,
  2. show that L is **NP**, and
  3. find a reduction converting Q to L.

- Since **NP-complete** languages can be reduced to each other, they are equally difficult. Moreover, if you can solve one of them, you solve *all* **NP-complete** problems!

Order of NP–Completeness Proofs



NP-Complete Problems: I
k-Clique

- **Does an undirected graph have a clique of size k?**
- A k-clique, or a clique of size k, in a graph G is a *complete subgraph* of G with k vertices.
- A **complete graph** is a graph such that every pair of distinct vertices is connected with an edge.
- k-Clique is in **NP**, since we can use a nondeterministic Turing machine to guess a subgraph of k vertices and verify if it is a clique.

  Guessing takes k steps and verifying needs to check if every pair of distinct vertices is connected with an edge, which takes $O(k^2)$ steps. Therefore, k-Clique is in **NP**.
- To prove k-Clique is NP-complete, we shall transform SAT to k-Clique.

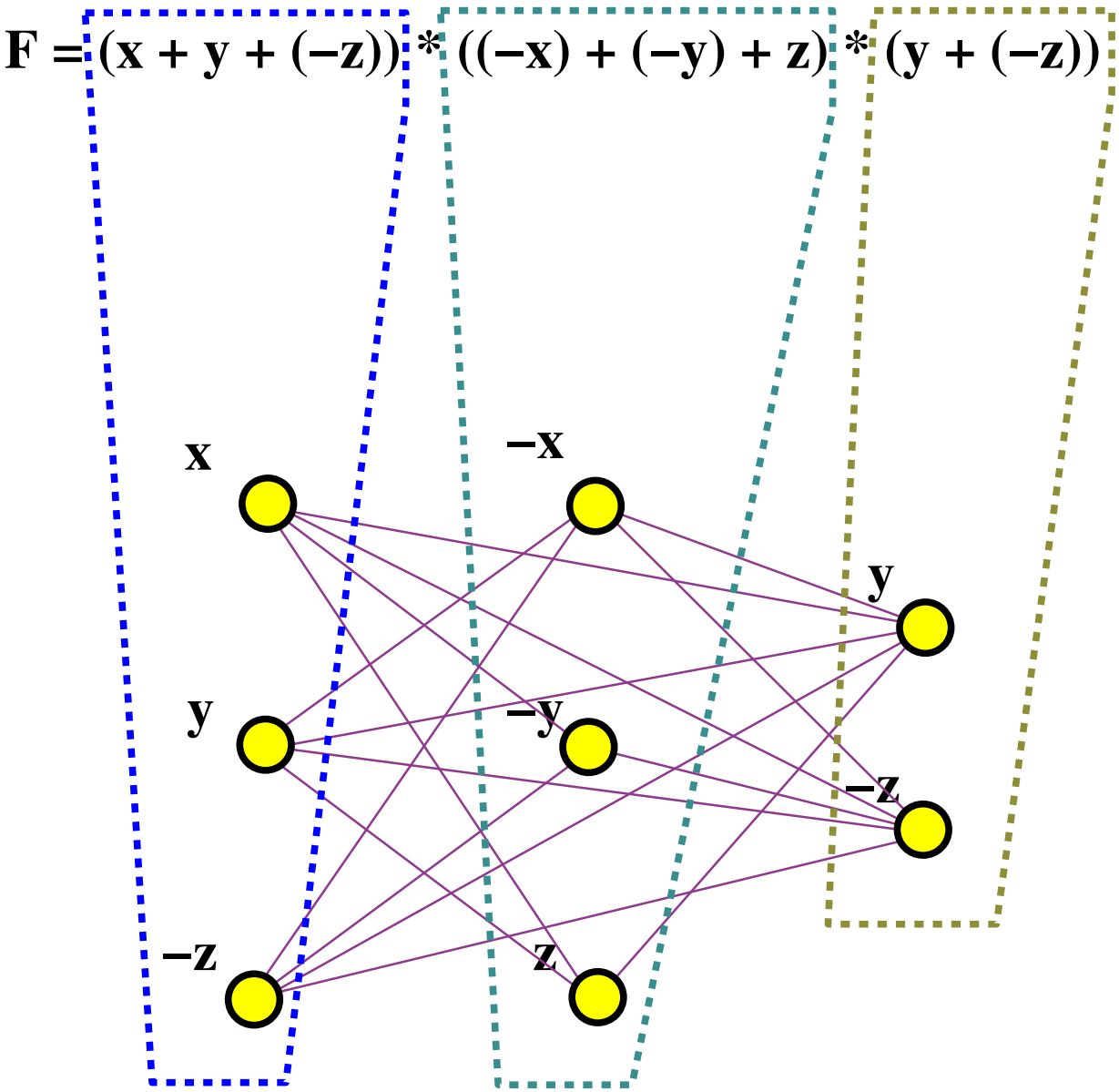
k-Clique is NP-Complete

- Given a Boolean expression in CNF form, $F = F_1 \ \& \ F_2 \ \& \ \cdots F_q$, we shall construct a graph G such that "F **is satisfiable** $\Leftrightarrow$ **graph** G **has a** q**-clique**."
- Let clause F_i be $F_i = f_{i1} \vee f_{i2} \vee \cdots \vee f_{ik_i}$, where k_i is the number of terms in clause F_i.
- For each F_i, list its terms on a column. These terms become the vertices of the constructed graph.
- For a term f_{ij} on F_i's column and f_{mn} on F_m's column, where $i \neq m$, if f_{ij} and f_{mn} are **not** complement to each other, then connect an edge from f_{ij} to f_{mn}.

  1. No edge between vertices on the same column
  2. No edge between x and $\bar{x}$

  Call this graph G.
- This construction can be done in polynomial-time. **Prove it yourself.**
- Then, we shall show F **is satisfiable** if and only if G **has a** q**-clique**. Note that q is the number of clauses of F.

From Boolean Expression to Graph
3–Clique Construction



k-Clique is NP-Complete
From SAT to k-Clique

- If F is satisfiable, then F_i must be TRUE for all i, $1 \leq i \leq q$, since the operator is &.
- In F_i, one term must be TRUE, since the operator is $\vee$. Let this term be X_i. Thus, we have q terms, X_1, X_2, ..., X_q, one from each column.
- We claim that these terms (or vertices) form a q-clique. In order words, we want to show that there is an edge between X_i and X_j, where $1 \leq i, j \leq q$ and $i \neq j$.

  Prove by contradiction:

  1. If there were no edge between X_i and X_j, then X_i and X_j would be complement to each other by construction. That is, $X_j = \bar{X}_i$.
  2. However, because X_i and X_j are both TRUE, we have $X_j = \bar{X}_i =$ FALSE. Thus, we have a contradiction.

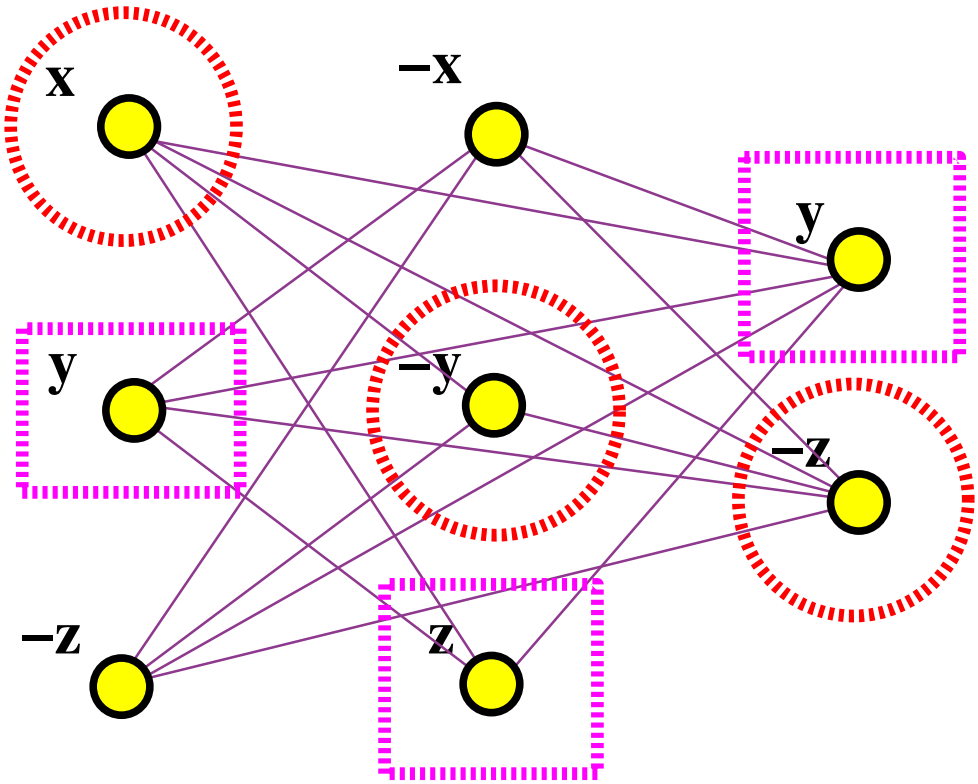
- Therefore, we have a q-clique.

From Boolean Expression to Graph An Example

F = (x + y + (–z)) * ((–x) + (–y) + z) * (y + (–z))

x = TRUE y = z = FALSE

y = z = TRUE x = ANY



The above shows two solutions and their corresponding 3–cliques

k-Clique is NP-Complete
From k-Clique to SAT

- Suppose we have a q-clique. Let the vertices be X_1, X_2, ..., X_q. Then, X_i must be on F_i's column, since there is no edges between any two vertices on the same column (by construction).
- Define sets S and $\bar{S}$ as follows:

$$S = \{y \mid y \text{ is a var. } \& \; y = X_i \text{ for some } i\}$$
$$\bar{S} = \{y \mid \bar{y} \text{ is a var. } \& \; \bar{y} = X_i \text{ for some } i\}$$

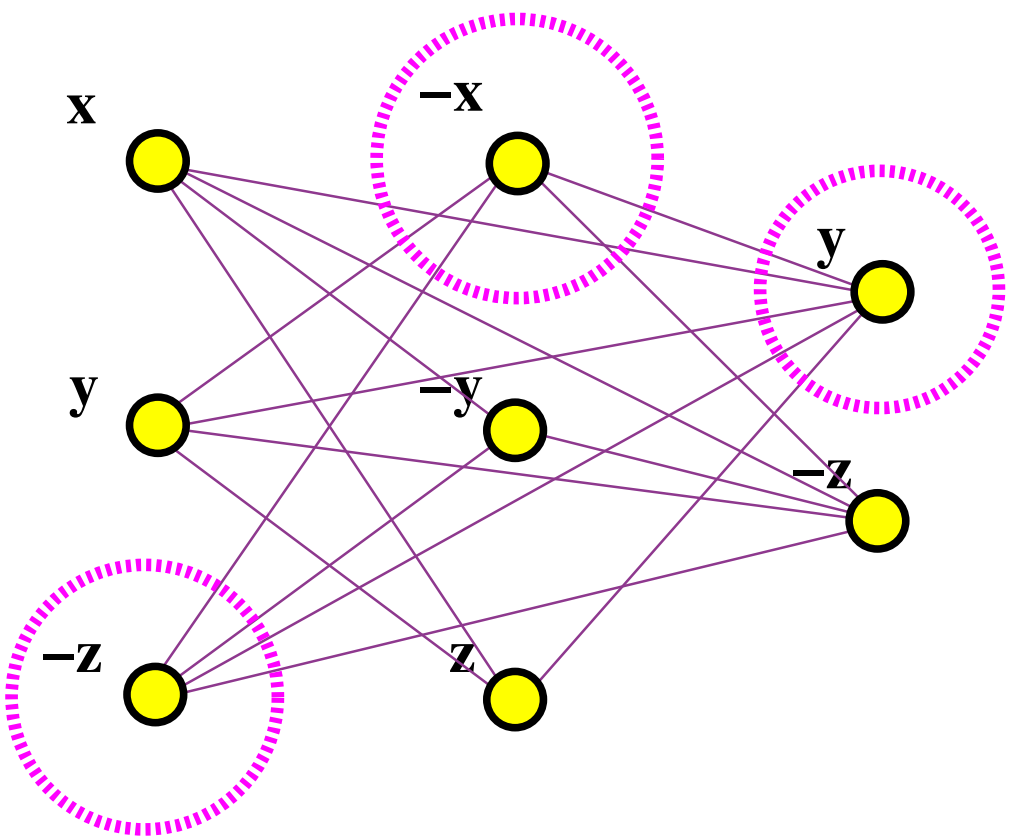
  Thus, S (*resp.*, $\bar{S}$) contains all variables whose names (*resp.*, complements) are identical to one of the vertices in the q-clique.

  We claim that setting all variables in S to TRUE and all variables in $\bar{S}$ to FALSE will satisfy the given Boolean expression F.

  1. $S \cap \bar{S} = \emptyset$; otherwise, our value assignments would be contradictory.
  2. Since F_i contains a variable in S or in $\bar{S}$, this variable (or its complement) gives a TRUE and as a result F_i is TRUE. Thus, F is satisfied.

From Boolean Expression to Graph from k–Clique to a Solution

F = (x + y + (–z)) * ((–x) + (–y) + z) * (y + (–z))



S = { y } S' = { x, z }

y = TRUE x = z = FALSE

Note that –z, –x and y form a 3–clique

NP-Complete Problems: II
Vertex Cover

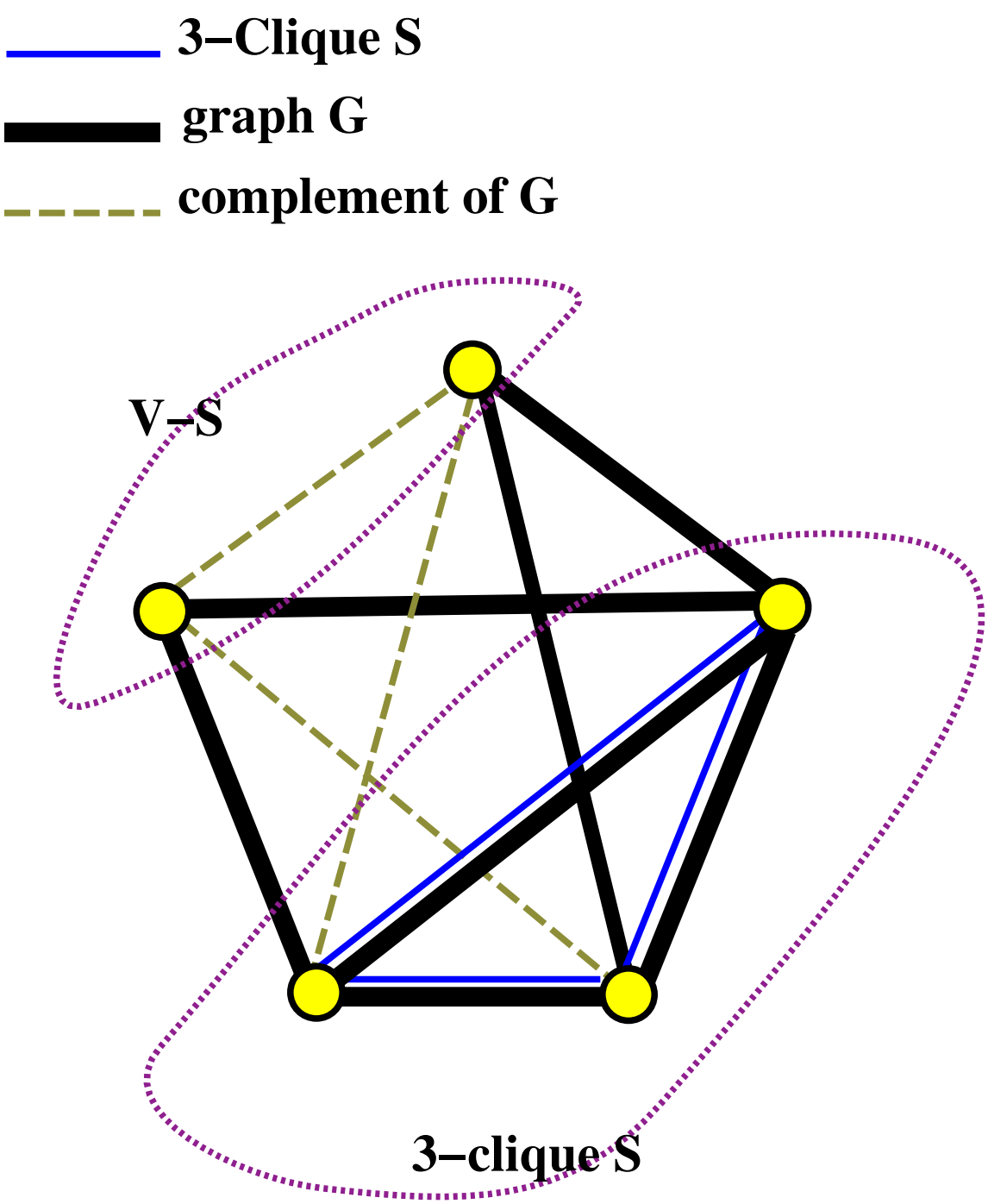
- **Definition:**
  Given a graph $G = (V, E)$, a *vertex cover* is a subset $S \subset V$ such that each edge of G is incident with some vertex in S
- **The Vertex Cover Problem:**
  Does an undirected graph have a vertex cover of size k?
- **The vertex cover problem is NP-complete**
- We can transform the k-clique problem to the vertex cover problem. Since k-clique is NP-complete, so is vertex cover.
- The *complement graph* of a graph $G = (V, E)$, written $\bar{G} = (V, \bar{E})$, has the edge set defined as follows:

  $$\overline{uv} \in \bar{E} \Leftrightarrow \overline{uv} \notin E$$

  where $\overline{uv}$ is the edge connecting vertices u and v.
- We want to prove the following

  $S \subset V$ is a clique in $G \Leftrightarrow V - S$ is a vertex cover of $\bar{G}$

Vertex Cover



$S \subset V$ **is a clique in** $G \Leftrightarrow$ $V - S$ **is a vertex cover of** $\bar{G}$

- **If** $S \subset G$ **is a clique, then** $V - S$ **is a vertex cover of** $\bar{G}$
  1. If $S \subset G$ is a clique, then for any two vertices u and v in S, their edge $\overline{uv}$ is an edge of the clique and hence $\overline{uv} \notin \bar{E}$.
  2. Edges in $\bar{G}$ cannot connect two vertices in S. Hence, an edge in $\bar{G}$ either connects two vertices in $V - S$ or one vertex in S while the other in $V - S$.
  3. Now, given an edge of $\bar{G}$, it must be incident with a vertex in $V - S$ and $V - S$ is a vertex cover of $\bar{G}$.

$S \subset V$ **is a clique in** $G \Leftrightarrow$ $V - S$ **is a vertex cover of** $\bar{G}$

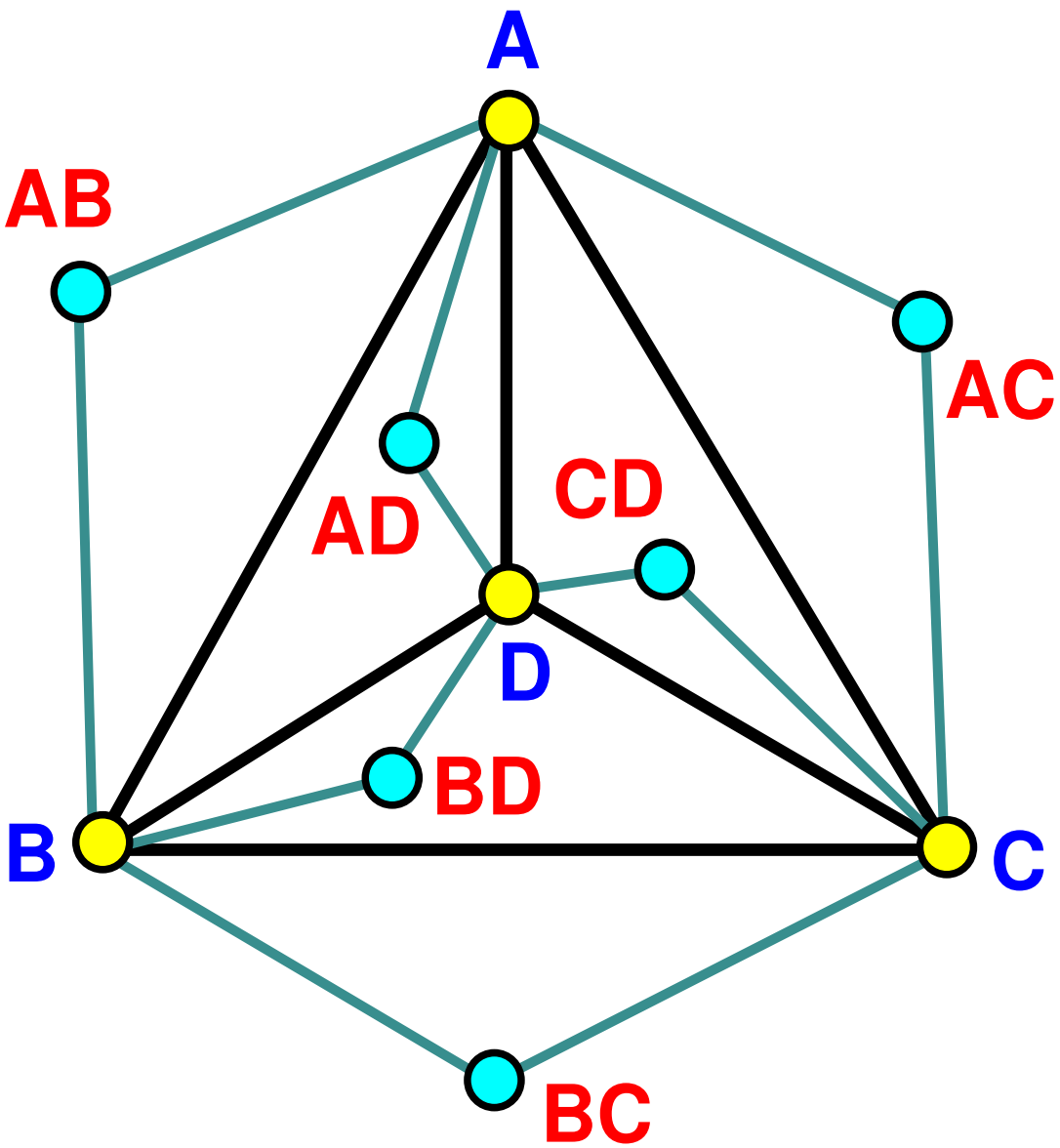
- **if** $V - S$ **is a vertex cover of** $\bar{G}$**, then** $S \subset V$ **is a clique in** G
  1. Consider vertices u and v in S. We shall prove by contradiction that edge $\overline{uv}$ must be in S.
  2. Suppose there were no edge connecting u and v in the original graph G. Then, $\overline{uv}$ must be an edge of $\bar{G}$.
  3. Since $V - S$ is a vertex cover of $\bar{G}$, every edge in $\bar{G}$ must have one of its two vertices in $V - S$.
  4. Since $\overline{uv}$ is an edge of $\bar{G}$, one of its vertices must be in $V - S$. However, this is impossible because by assumption u and v are vertices of S.
  5. Therefore, every pair of vertices of S is connected with an edge and S is a clique.

NP-Complete Problems: III
Dominating Set

- **Does an undirected graph have a dominating set of size k?**
- Given a graph $G = (V, E)$ and a vertex set $D \subseteq V$, D is a **dominating set** if every vertex of G is either in D or is adjacent to at least one vertex in D.
- Finding a dominating set of size k is in **NP**.
- To prove finding a dominating set of size k is **NP**-complete, we reduce vertex cover to dominating set.
- Given $G = (V, E)$, we shall construct a new graph $G' = (V', E')$ such that if there is a vertex cover of size k in G, there is a dominating set of size k in G'.
- For each edge (a, b) of G, we construct
  - A new vertex ab
  - Two new edges (ab, a) and (ab, b)

  More precisely, for each edge in G, we construct a triangle on top on this edge.
- Let the new graph that consists of all old and new vertices and edges be G'.

Dominating Set Construction

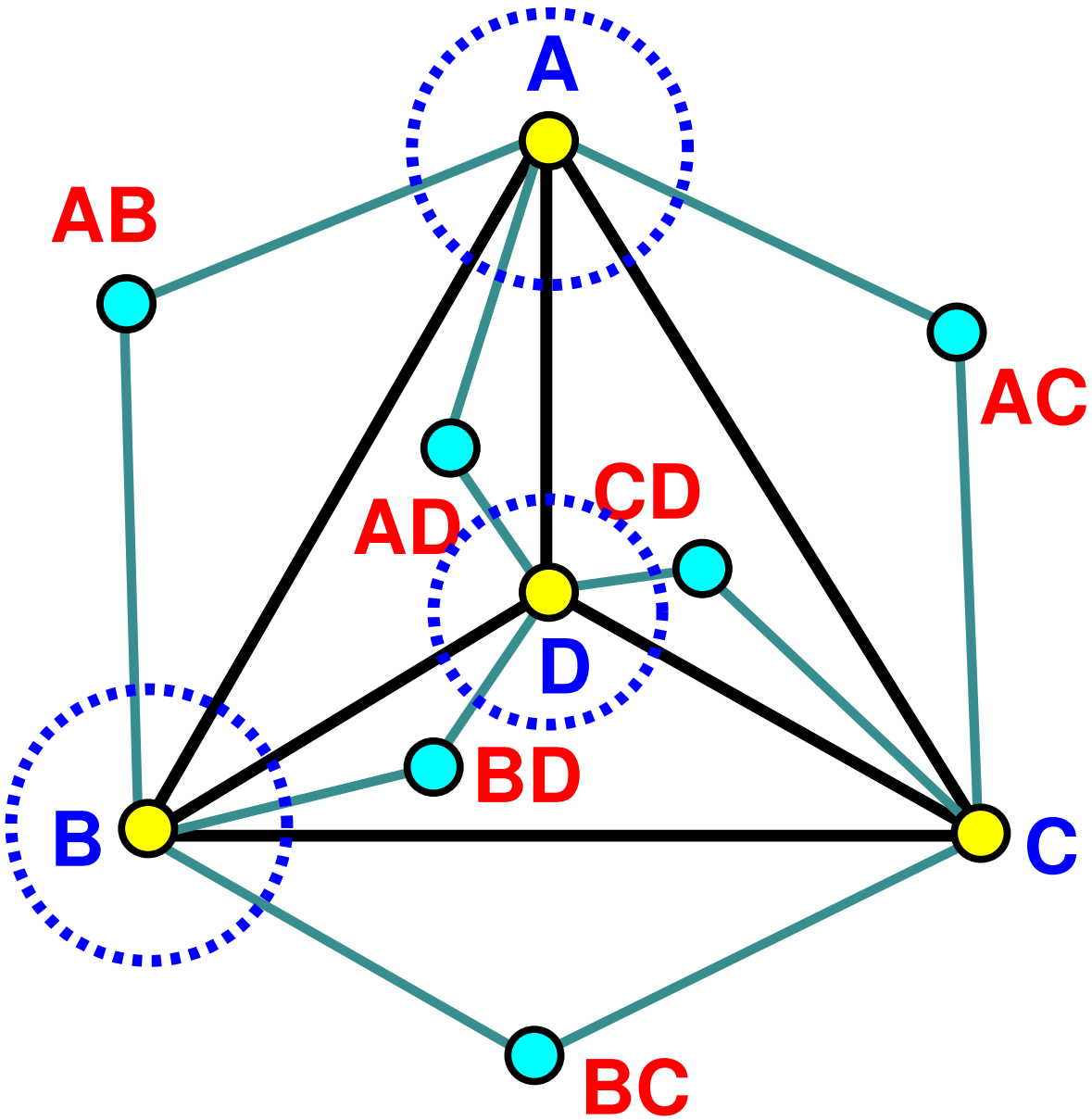


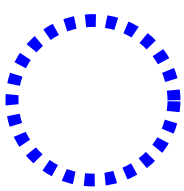
Original Graph: ——
New Graph: ——

A Vertex Cover of Size k in $G \Rightarrow$ A Dominating Set of Size k in G'

- If D is a vertex cover of size k in G, then every edge in G is incident to a vertex in D.
- Note that D does not contain any new vertices.
- We shall show that D is a dominating set for G'.
  1. For a vertex a in G, either $a \in D$ or $a \notin D$.
  2. If $a \notin D$, since D is a vertex cover of G, an edge that is incident to a (*i.e.*, (a, b)) forces $b \in D$. Therefore, D dominates all vertices in G.
  3. Consider a vertex ab in G'. It is adjacent to vertices a and b (both in G). Then, either a or b must be in D because D is a vertex cover of G, and vertex ab is adjacent to a vertex in D. Therefore, D also dominates all vertices in G'.
  4. In summary, D is a dominating set of size k for G'.

Vertex Cover to Dominating Set



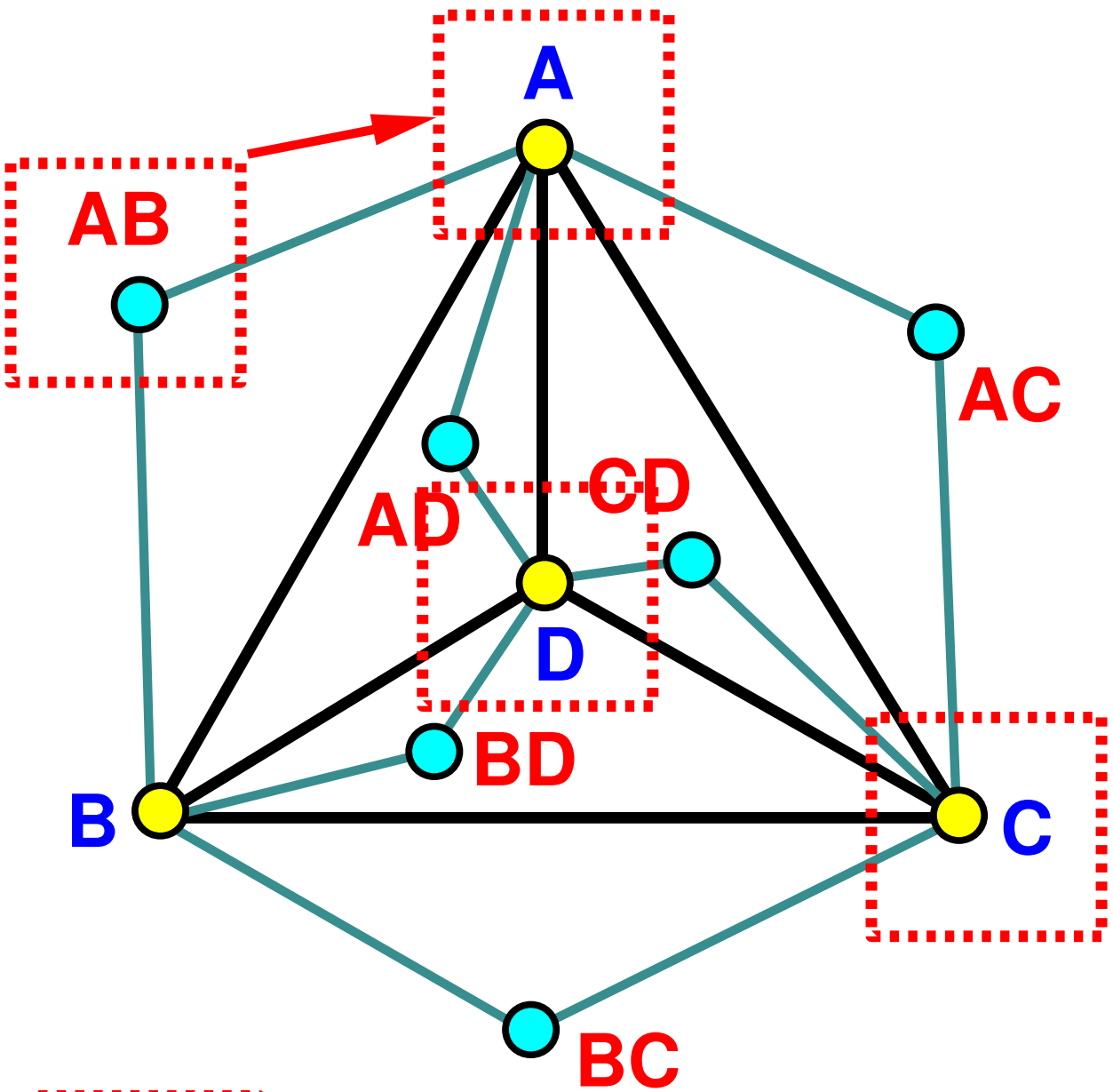


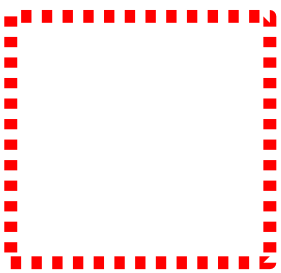
vertex cover

A Vertex Cover of Size k in $G \Leftarrow$ A Dominating Set of Size k in G'

- Let D be a dominating set of size k for G'.
- We can assume that no new vertices of form ab are in D. If $ab \in D$, we can replace ab with a or b. However, if both a and b are in D, we can replace ab with any vertex in G that is not in D. The new D is still a dominating set of size k.
- We want to show that $D \subseteq V$ is a vertex cover. More precisely, given an arbitrary edge (a, b), we want to show either $a \in D$ or $b \in D$.
- Because D dominates *all* new vertices, including ab (*i.e.*, the one on edge (a, b)), and because ab is not in D, one of its neighbor a or b must be in D. Therefore, D is a vertex cover of G!

Dominating Set to Vertex Cover





dominating set

NP-Complete Problems: IV
3-Satisfiability

- A Boolean expression F is in **3-CNF** form if it is written as $F = F_1 \ \& \ F_2 \ \& \ \cdots \ \& \ F_n$ and for each clause F_i there are **exactly three** terms. More precisely, each clause has a form of $f_{i1} \vee f_{i2} \vee f_{i3}$, where each f_{ij} is either a variable or its complement.
- **What is 3-SAT?**

  Does a Boolean expression in 3-CNF form satisfiable?
- **The 3-SAT problem is NP-complete.**

  We shall transform the Satisfiability problem to 3-SAT.

NP-Complete Problems: IV
3-SAT is NP-Complete

- 3-SAT is NP. **Prove it yourself.**
- Given a Boolean expression F in CNF form, we shall construct a new expression G in 3-CNF form and then prove that

$$F \text{ is satisfiable } \Leftrightarrow \ G \text{ is satisfiable}$$

- **Construction:**
  1. Suppose $F = F_1 \ \& \ F_2 \ \& \ \cdots \ \& \ F_n$. We shall convert each F_i to an expression in 3-CNF form.
  2. Let $F_i = f_{i1} \vee f_{i2} \vee \cdots \vee f_{ik}$, where f_{ij} is a variable or the complement of a variable.
  3. If F_i has three terms. We are done.
  4. If F_i has more than four terms, then let G_i be defined as follows:

$$G_i = (f_{i1} \vee f_{i2} \vee y_1) \ \& \ (f_{i3} \vee \bar{y}_1 \vee y_2) \ \& \ (f_{i4} \vee \bar{y}_2 \vee y_3) \ \& \ \cdots \ \& (f_{i,k-1} \vee f_k \vee \bar{y}_{k-3})$$

  5. We introduce $k-3$ new variables, $y_1, y_2, \ldots, y_{k-3}$, where k is the number of terms in F_i.

6. Since every F_i has a corresponding G_i, the desired 3-CNF expression is $G = G_1 \ \& \ G_2 \ \& \ \cdots \ \& \ G_n$, where G_i is repeated as follows:

$$G_i = (f_{i1} \vee f_{i2} \vee y_1) \ \& \ (f_{i3} \vee \bar{y}_1 \vee y_2) \ \& \ (f_{i4} \vee \bar{y}_2 \vee y_3) \ \& \ \cdots \ \& (f_{i,k-1} \vee f_k \vee \bar{y}_{k-3})$$

7. If F is satisfiable, F_i must have a f_{ip} whose value is TRUE. In this case, setting all y_1, y_2 ..., y_{i-2} to TRUE and all other y_i's to FALSE satisfies G_i. Since all G_i's are TRUE, so is G.
8. It G is satisfiable, then one of f_{ij}'s must be TRUE. Otherwise, if all f_{ij}'s are FALSE, G_i becomes

$$y_1 \ \& \ (\bar{y}_1 \vee y_2) \ \& \ (\bar{y}_2 \vee y_3) \ \& \ \cdots \ \& \ \bar{y}_{k-3}$$

and is clearly not satisfiable. **Prove it yourself.** Since each F_i has at least f_{ij} TRUE, F is TRUE and is satisfiable.

9. If F_i has two terms, then G_i is

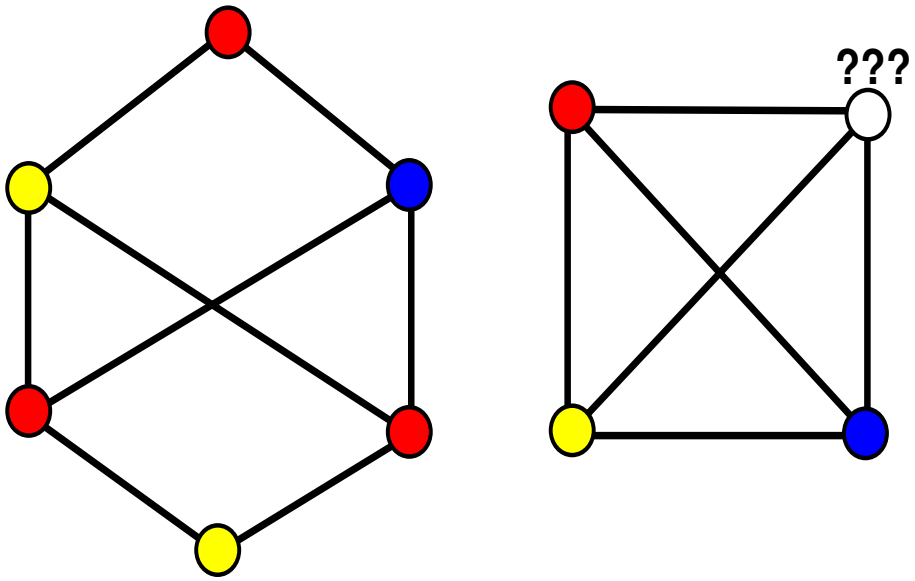
$$G_i = (f_{i1} \vee f_{i2} \vee y) \ \& \ (f_{i1} \vee f_{i2} \vee \bar{y})$$

10. If F_i has only one term, then G_i is

$$G_i = (f_{i1} \vee y \vee z) \ \& \ (f_{i1} \vee \bar{y} \vee z) \ \& \ (f_{i1} \vee y \vee \bar{z}) \ \& \ (f_{i1} \vee \bar{y} \vee \bar{z})$$

NP-Complete Problems: V
3-Colorability

- **Can an undirected graph be colored using three different colors?**
- A **valid coloring** of a graph is an assignment of colors to the vertices such that no two adjacent vertices have the same color.



- In the above, the left graph is 3-colorable and the right one is not. In fact, any complete graph with number of vertices larger than 3 is not 3-colorable.
- We shall reduce 3-SAT to 3-colorability.

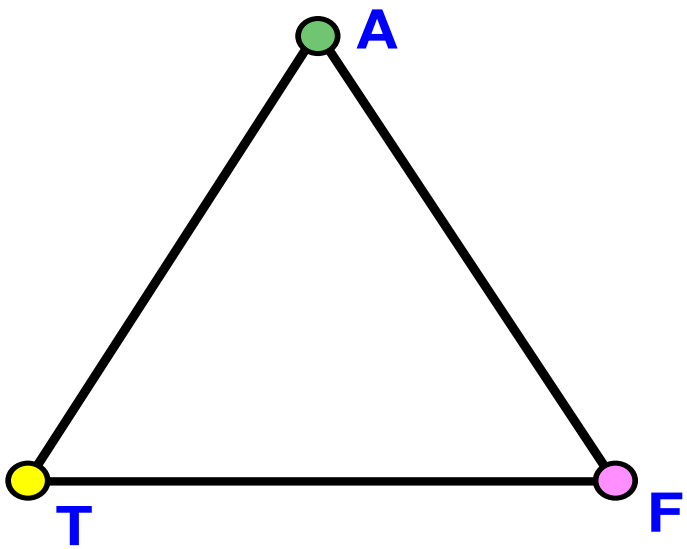
NP-Complete Problems: V
Construction

- Given a Boolean expression F in 3-CNF, we want to construct a graph G such that

  F is satisfiable $\Leftrightarrow$ G can be colored with 3 colors

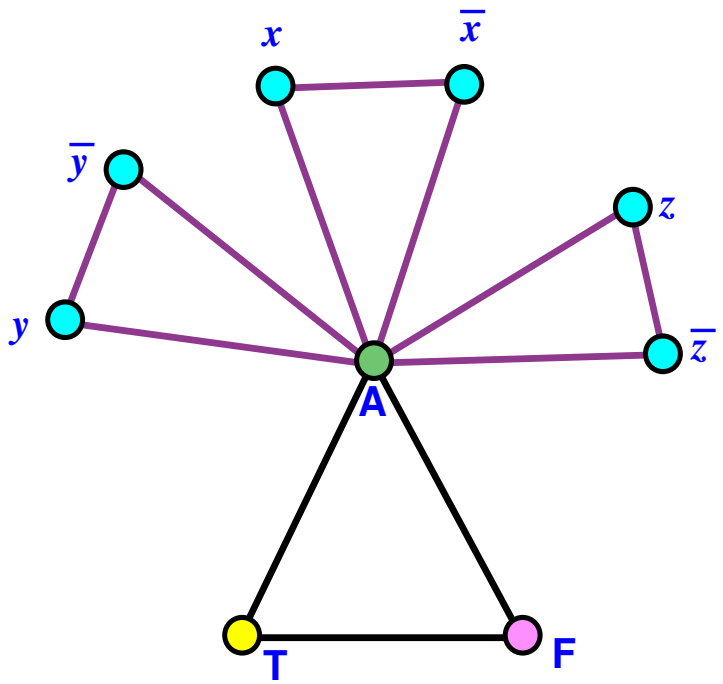
- **Construction Step 1:**

  1. Construct a *main* triangle with three vertices T, F and A for TRUE, FALSE and Auxiliary.
  2. We will use color names T, F and A.
  3. Note that a triangle requires exactly three colors.



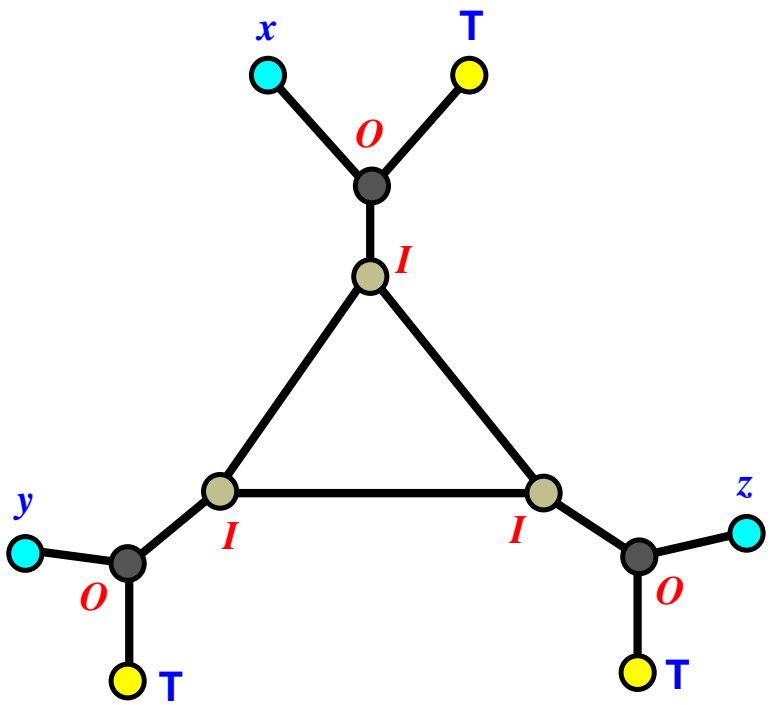
- **Construction Step 2:**

1. For each involved variable, say x, construct a *variable* triangle with vertices x, $\bar{x}$ and A.
2. Thus, all of these new variable triangles and the *main* triangle share a common vertex A.
3. This construction forces variable x and its complement $\bar{x}$ to use different colors. Since A has been used, x and $\bar{x}$ must be colored with T or F.

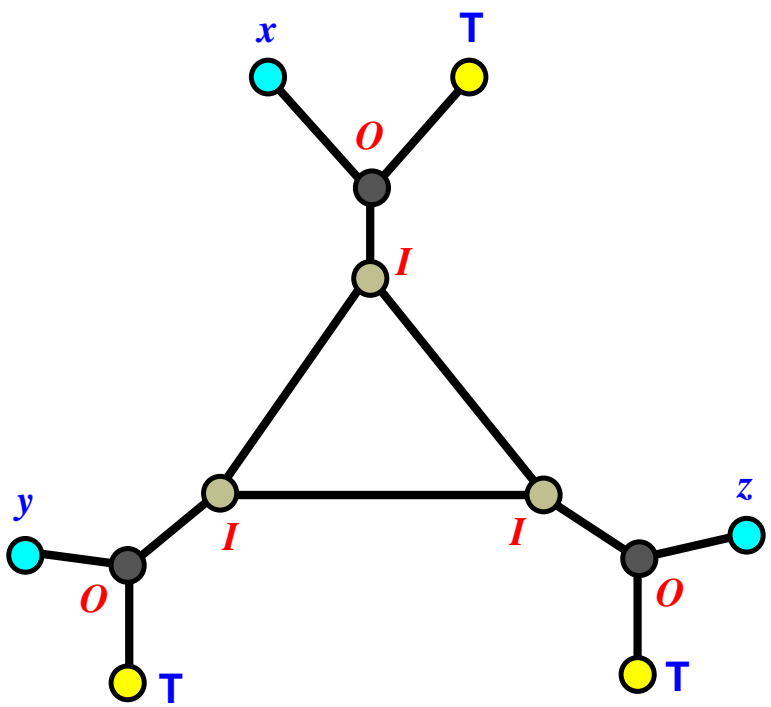


- **Construction Step 3:**

  1. For each clause, add six vertices, named *inner* and *outer* vertices.
  2. The inner vertices form a triangle. Thus, the inner vertices use all 3 colors.
  3. Each inner vertex is attached to an outer vertex.
  4. Each outer vertex is attached to T in the main triangle and to the involved variable. (We have 3 variables in each clause.)
  5. The following shows the construction for $(x \vee y \vee z)$.

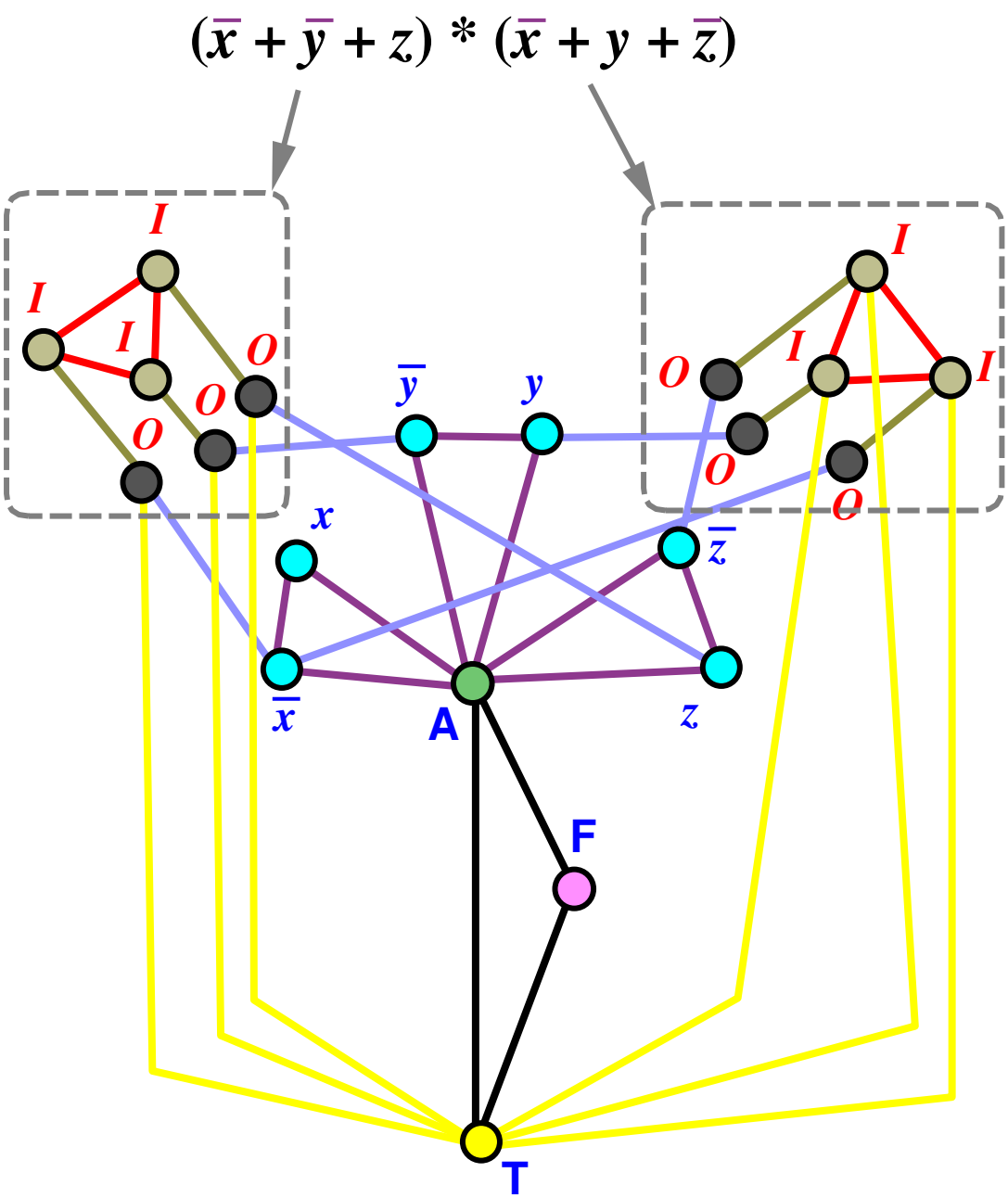


6. This construction forces one of the involved variable in each clause to be colored as T.
7. Suppose all involved variables of a clause are colored with F. Then, all *outer* vertices must be colored with A.
8. Since every *inner* vertex is adjacent to an *outer* vertex, none of the *inner* vertex can be colored with A. As a result, the *inner* vertices are not 3-colorable.
9. Consequently, **if the graph is 3-colorable, one variable in each clause must be colored with** T.



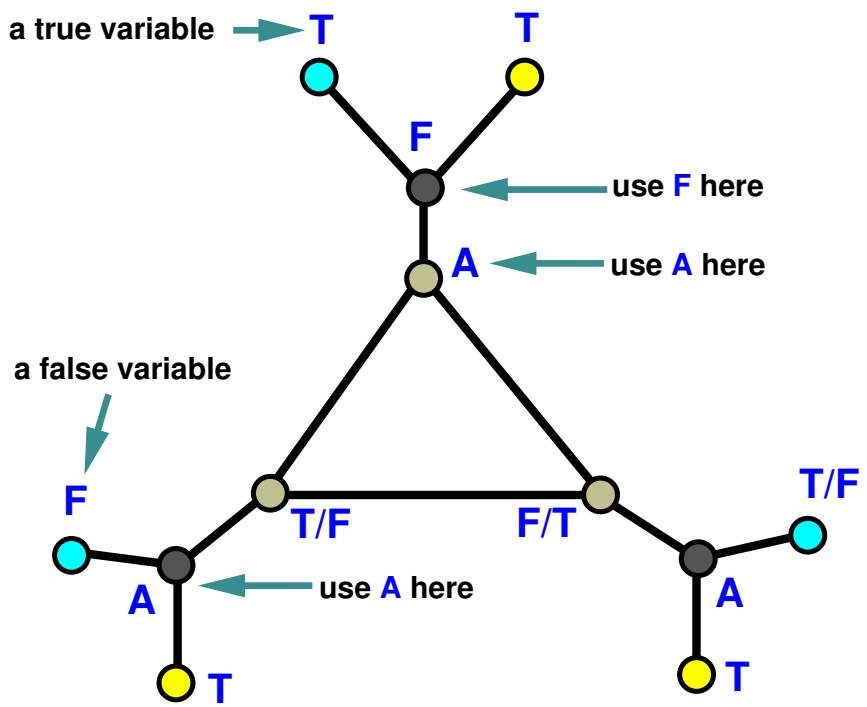
- The following is a complete construction for

$$(\bar{x} \vee \bar{y} \vee z) \wedge (\bar{x} \vee y \vee \bar{z})$$



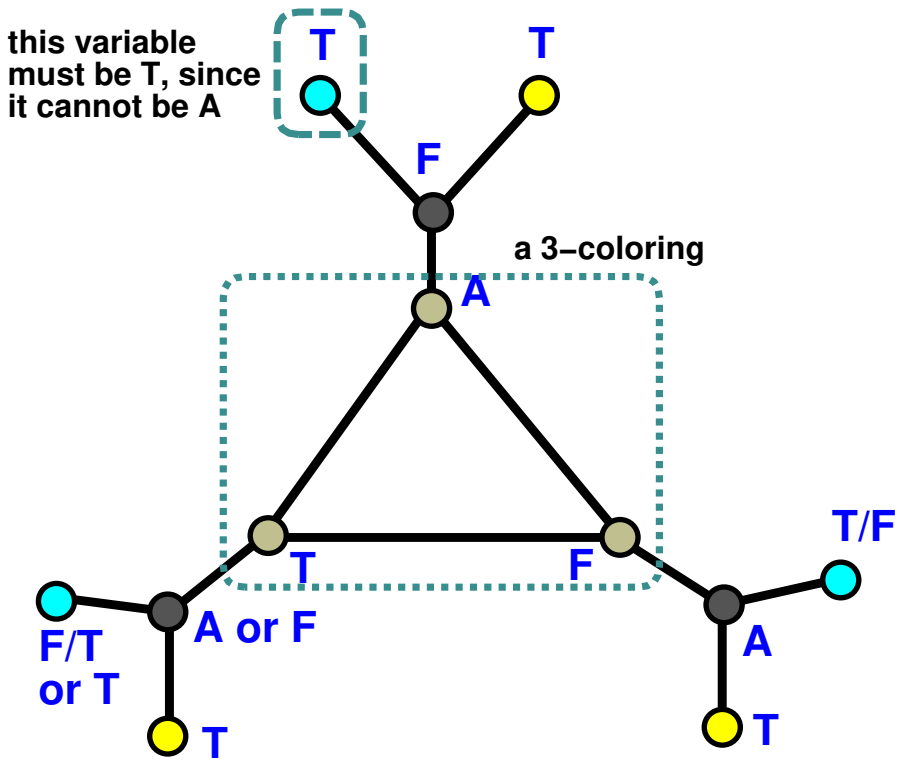
F is Satisfiable $\Rightarrow$ G is 3-Colorable

- If F is satisfiable, every clause has a TRUE variable.
- Those TRUE (*resp.*, FALSE) variables are colored with T (*resp.*, F).
- For a T variable, its adjacent *outer* vertex uses F and the corresponding *inner* vertex uses A.
- For a F variable, its adjacent *outer* vertex uses A. Then, the other *inner* and *outer* vertices can be colored accordingly.



F is Satisfiable $\Leftarrow$ G is 3-Colorable

- If G is 3-colorable, pick the *inner* vertex with color A.
- Then, its adjacent *outer* vertex must be colored F.
- The adjacent variable of this *outer* vertex must be T since a variable cannot be colored as A due to the structure of *variable* triangles.
- Thus, every clause has a TRUE term and F is satisfiable.



More NP-Complete Problems

- **Directed Hamilton Circuit:** Given a directed graph $G = (V, E)$, is there a Hamilton circuit (path)? A Hamilton circuit is a direct cycle that contains all vertices exactly once.
- **Undirected Hamilton Circuit:** Given an undirected graph $G = (V, E)$, is there a Hamilton circuit (path)? A Hamilton circuit is a cycle that contains all vertices exactly once.
- **Traveling Salesperson Problem:** Given a weighted graph, find the minimum weighted Hamilton circuit.
- **Solving Quadratic Equations:** Given positive integers a, b and c, are there positive integers x and y such that $ax^2 + by + c = 0$?
- **Race Condition Detection:** Given an arbitrary program, find all race conditions. (But, finding all deadlocks is NP-Hard.)
- **Register Sufficiency:** Given a directed acyclic graph $G = (V, E)$ and a positive integer K, is there a computation for G that uses fewer than K registers?

- **Code Generation with Registers:** Given a directed acyclic graph $G = (V, E)$ in which no vertex has out-degree larger than 2 and a positive integer K, is there a program with K or fewer instructions for computing all root vertices of G. The number of registers can be one or even unlimited.
- For further reference, see

  Michael R. Garey and David S. Johnson, *Computers and Intractability: A guide to the Theory of NP-Completeness*, Freeman, 1979. Garey and Johnson had a column for *Journal of Algorithms*.