Lecture 19: NP-Completeness 1

Revised Sun May 25, 2003

Outline of this Lecture

- Polynomial-time reductions.
  CLRS pp.984-5
- The class $\mathcal{NPC}$.
  CLRS p. 986
- Proving that problems are $\mathcal{NPC}$.
  SAT, CLIQUE, INDEPENDENT SET, VERTEX COVER
  CLRS pp. 995-1007
- Optimization vs. Decision problems

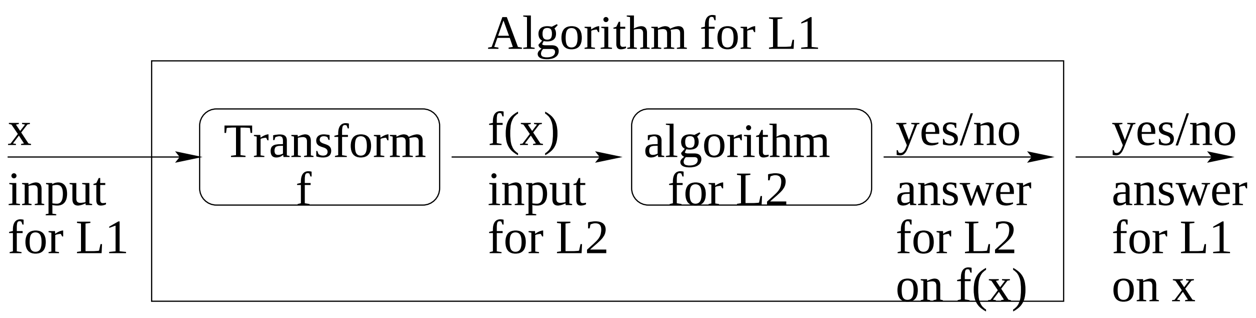
Reductions between Decision Problems

What is Reduction?

Let L_1 and L_2 be two decision problems.

Suppose algorithm A_2 solves L_2. That is, if y is an input for L_2 then algorithm A_2 will answer Yes or No depending upon whether $y \in L_2$ or not.

The idea is to find a transformation f from L_1 to L_2 so that the algorithm A_2 can be part of an algorithm A_1 to solve L_1.



Polynomial-Time Reductions

Definition: A Polynomial-Time Reduction from L_1 to L_2 is a transformation f with the following properties:

- f transforms
  an input x for L_1 into an input $f(x)$ for L_2 s.t.
  $f(x)$ is a yes-input for L_2
  if and only if
  x is a yes-input for L_1.

- $f(x)$ is computable in polynomial time (in $size(x)$).

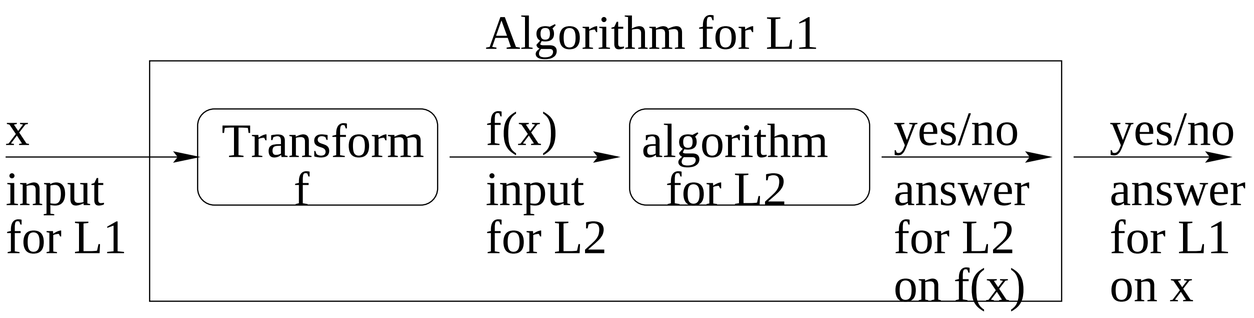
If such an f exists, we say that
L_1 is polynomial-time reducible to L_2,
and write $L_1 \leq_P L_2$.

Polynomial-Time Reductions

Question: What can we do with a polynomial time reduction $f : L_1 \to L_2$?

Answer: Given an algorithm A_2 for the decision problem L_2, we can develop an algorithm A_1 to solve L_1.

In particular (proof on next slide)
if A_2 is a polynomial time algorithm for L_2 and $L_1 \leq_P L_2$
then we can construct a polynomial time algorithm for L_1.



$\boxed{\textbf{Polynomial-Time Reduction } f : L_1 \to L_2}$

Theorem:
If $L_1 \leq_P L_2$ and $L_2 \in \mathcal{P}$, then $L_1 \in \mathcal{P}$.

Proof: $L_2 \in \mathcal{P}$ means that we have a polynomial-time algorithm A_2 for L_2. Since $L_1 \leq_P L_2$, we have a polynomial-time transformation f mapping input x for L_1 to an input for L_2. Combining these, we get the following polynomial-time algorithm for solving L_1:

(1) take input x for L_1 and compute $f(x)$;

(2) run A_2 on input $f(x)$, and return the answer found (for L_2 on $f(x)$) as the answer for L_1 on x.

Each of Steps (1) and (2) takes polynomial time. So the combined algorithm takes polynomial time. Hence $L_1 \in \mathcal{P}$.

$\boxed{\textbf{Warning}}$

We have just seen

Theorem:
If $L_1 \leq_P L_2$ and $L_2 \in \mathcal{P}$, then $L_1 \in \mathcal{P}$.

Note that this **does not imply** that
If $L_1 \leq_P L_2$ and $L_1 \in \mathcal{P}$, then $L_2 \in \mathcal{P}$.
This statement is not true.

Reduction between Decision Problems

Lemma (Transitivity of the relation $\leq_P$):
If $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$, then $L_1 \leq_P L_3$.

Proof: Since $L_1 \leq_P L_2$, there is a polynomial-time reduction f_1 from L_1 to L_2.
Similarly, since $L_2 \leq_P L_3$ there is a polynomial-time reduction f_2 from L_2 to L_3.

Note that $f_1(x)$ can be calculated in time polynomial in $size(x)$.
In particular this implies that $size(f_1(x))$ is polynomial in $size(x)$.
$f(x) = f_2(f_1(x))$ can therefore be calculated in time polynomial in $size(x)$.

Furthermore x is a yes-input for L_1 if and only if $f(x)$ is a yes-input for L_3(why). Thus the combined transformation defined by

$$f(x) = f_2(f_1(x))$$

is a polynomial-time reduction from L_1 to L_3.
Hence $L_1 \leq_P L_3$.

$\boxed{\text{Example of Polynomial-Time Reduction}}$

CYC: Does an undirected graph G have a cycle?

TRIPLE: Does a triple (n, e, t) of nonnegative integers satisfy $e \neq n - t$?

Reduction f: We define f to be

$$f : \text{CYC}(G) \to \text{TRIPLE}(n, e, t),$$
$$f : G \mapsto (n, e, t),$$

where n, e and t are the number of vertices, edges and connected components of G respectively.

We will show that
1) G has a cycle if and only if $f(G) \in$ TRIPLE.
2) $f(G)$ can be calculated in time polynomial in $size(G) = n + e$.

This proves that

$$\text{CYC} \leq_P \text{TRIPLE.}$$

$\boxed{\text{Example Continued}}$

G has a cycle if and only if $f(G) \in$ TRIPLE

follows from

Lemma: A graph G has no cycles if and only if $e = n - t$, where n, e, and t are the number of vertices, edges, and connected components respectively.

Proof: First, if G is connected, then $t = 1$ and G has no cycles if and only if it is a tree. For a tree, $e = n-1$.

Suppose G has t components, $G_1, G_2, \ldots, G_t$, where G_i has n_i vertices and e_i edges. G does n then $e_i = n_i - 1$ by the first part of the proof. Hence

$$e = \sum_{i=1}^{t} e_i = \sum_{i=1}^{t} (n_i - 1) = n - t.$$

Example Continued

To show that $f(G)$ can be calculated in time polynomial in $size(G)$ we need to solve

Problem: Design a polynomial time algorithm that, given undirected graph $G = (V, E)$, computes the number of connected components of G.

Question: How do you design an efficient algorithm for this problem?

Possible solutions: Modify BFS or DFS, adding a counter.The next page gaves an $O(|E| + |V|)$ time algorithm

Example Continued

One solution:
A modified DFS that runs in time $O(n+e)$:

```
COMP(G)
{   for each u in V
        color[u]=W;
    number=0;
    for each u in V
        if(color[u]==W)
        {   number=number+1;
            DFSVisit(u);
        }
    return(number);
}

DFSVisit(u)
{   color[u]=G;
    for each v in adj[u]
        if(color[v]==W)
            DFSVisit(v);
    color[u]=B;
}
```

Example Continued

Recall that we defined

$$f : \text{CYC}(G) \rightarrow \text{TRIPLE}(n, e, t),$$
$$f : G \mapsto (n, e, t),$$

where n, e and t are the number of vertices, edges and connected components of G respectively.

Using the algorithm on the previous slide we see that we can calculate t in time polynomial in $size(G)$. Since we can also calculate e and n in polynomial time **we can calculate $f(G)$ in polynomial time.**

Recall too that we saw G **has a cycle if and only if $f(G) \in$ TRIPLE** so G **is a yes-input for CYC if and only if $f(G)$ is a yes-input for TRIPLE.**

Combining these two facts shows that f is a Polynomial-Time Reduction from CYC to TRIPLE or

$$\text{CYC} \leq_P \text{TRIPLE}$$

$\boxed{\text{Finally: The Class } \mathcal{NP}\text{-Complete } (\mathcal{NPC})}$

We have finally reached our goal of introducing

Definition: The class $\mathcal{NPC}$ of $\mathcal{NP}$-complete problems consists of all decision problems L such that

(a) $L \in \mathcal{NP}$;

(b) for every $L' \in \mathcal{NP}$, $L' \leq_P L$.

Intuitively, $\mathcal{NPC}$ consists of all the **hardest** problems in $\mathcal{NP}$.

Note: it is not obvious that there exist any such problems at all.
There are an infi nite number of problems in $\mathcal{NP}$. How can we prove that some problem is at least as hard as *all* of them?
We will see later that there are actually many such problems.

$\mathcal{NP}$-Completeness and Its Properties

The major reason we are interested in NP-Completeness is the following theorem which states that either *all* $\mathcal{NP}$-Complete problems are polynomial time solvable or *all* $\mathcal{NP}$-Complete problemsare not polynomial time solvable.

Theorem: Suppose that L is $\mathcal{NPC}$.

- If there is a polynomial-time algorithm for L, then there is a polynomial-time algorithm for every $L' \in \mathcal{NP}$.

  Proof: By the theorem on Page 5.

- If there is no polynomial-time algorithm for L, then there is no polynomial-time algorithm for any $L' \in \mathcal{NPC}$.

  Proof: By the previous conclusion.

$\boxed{\text{The Classes } \mathcal{P}, \mathcal{NP}, \text{co-}\mathcal{NP}, \text{and } \mathcal{NPC}}$

Proposition: $\mathcal{P} \subseteq \mathcal{NP}$.
Simple proof omitted

Question 1: Is $\mathcal{NPC} \subseteq \mathcal{NP}$?
Yes, by definition!

Question 2: Is $\mathcal{P} = \mathcal{NP}$?
Open problem! Probably very hard
It is generally believed that $\mathcal{P} \neq \mathcal{NP}$.
Proving this (or the opposite) would win you the US$1,000,000 Clay Prize.

Question 3: Is $\mathcal{NP} = \text{co}-\mathcal{NP}$?
Open problem! Probably also very hard
Note: if $\mathcal{NP} \neq \text{co}-\mathcal{NP}$, then $\mathcal{P} \neq \mathcal{NP}$ (why?).

Proving that problem L is $\mathcal{NPC}$-Complete

1. Show $L \in \mathcal{NP}$.

2. Show that $L' \leq_P L$ for a suitable $L' \in \mathcal{NPC}$.

Question 1: How do we get one problem in $\mathcal{NPC}$ to start with?
Answer: We need to prove, from scratch that one problem is in $\mathcal{NPC}$.

Question 2: Which problem is ”suitable”?
Answer: There is no general procedure to determine this. You have to be knowledgeable, clever and (sometimes) lucky.

Proving that problems are $\mathcal{NPC}$-Complete

In the rest of this lecture we will discuss some specific $\mathcal{NP}$-Complete problems.

1. SATand 3-CNF $-$ SAT.
   We will assume that they are $\mathcal{NP}$-Complete. (From textbook)

2. DCLIQUE:
   by showing $3 - \mathrm{CNF} - \mathrm{SAT} \leq_{\mathrm{P}} \mathrm{DCLIQUE}$
   The reduction used is very unexpected!

3. Decision Vertex Cover DVC:
   by showing $\mathrm{DCLIQUE} \leq_{\mathrm{P}} \mathrm{DVC}$
   The reduction used is very natural.

4. Decision Independent Set (DIS):
   by showing $\mathrm{DVC} \leq_{\mathrm{P}} \mathrm{IS}$
   The reduction used is very natural.

The Satisfiability Problem is $\mathcal{NPC}$

Cook (1971): $\mathsf{SAT} \in \mathcal{NPC}$.

Remark: For a proof, see pp. 997-998 of the textbook.

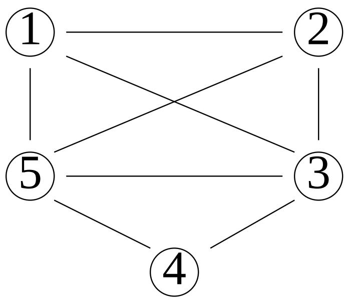
Remark: Since Cook showed that $\mathsf{SAT} \in \mathcal{NPC}$, thousands of problems have been shown to be in $\mathcal{NPC}$ using the reduction approach described earlier.

Remark: With a little more work we can also show that $3-\mathsf{CNF}-\mathsf{SAT} \in \mathcal{NPC}$ as well. pp. 998-1002.

Note: For the purposes of this course you only need to know the validity of the two facts above but do not need to know how to prove they are correct.

Problem: CLIQUE

Clique: A *clique* in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices such that each pair $u, v \in V'$ is connected by an edge $(u, v) \in E$. In other words, a clique is a complete subgraph of G (a vertex is a clique of size 1, an edge a clique of size 2).

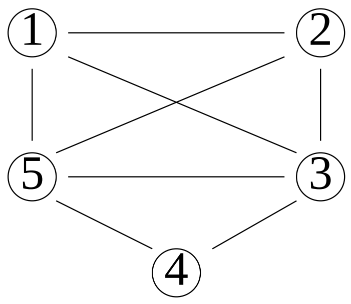


Find a clique with 4 vertices

CLIQUE: Find a clique of maximum size in a graph.

$\boxed{\mathcal{NPC}\text{ Problem: DCLIQUE}}$

The Decision Clique Problem (DCLIQUE): Given an undirected graph G and an integer k, determine whether G has a clique with k vertices.



Find a clique with 4 vertices

Theorem: DCLIQUE $\in \mathcal{NPC}$.

Proof: We need to show two things.
(a) That DCLIQUE $\in \mathcal{NP}$ and
(b) That there is some $L \in \mathcal{NPC}$ such that

$$L \leq_P \text{DCLIQUE}.$$

Proof that DCLIQUE $\in \mathcal{NPC}$

Theorem: DCLIQUE $\in \mathcal{NPC}$.

Proof: We need to show two things.
(a) That DCLIQUE $\in \mathcal{NP}$ and
(b) That there is some $L \in \mathcal{NPC}$ such that

$$L \leq_P \text{DCLIQUE}.$$

Proving (a) is easy. A certificate will be a set of vertices $V' \subseteq V$, $|V'| = k$ that is a possible clique. To check that V' is a clique all that is needed is to check that all edges (u, v) with $u \neq v$, $u, v \in V'$, are in E. This can be done in time $O(|V|^2)$ if the edges are kept in an adjacency matrix (and even if they are kept in an adjacency list – how?).

To prove (b) we will show that

$$3 - \text{CNF} - \text{SAT} \leq_P \text{DCLIQUE}.$$

This will be the hard part.
We will do this by building a ‘gadget’ that allows a reduction from the $3 - \text{CNF} - \text{SAT}$ problem (on logical formulas) to the DCLIQUE problem (on graphs).

Proof that DCLIQUE $\in \mathcal{NPC}$ (cont)

Recall (from Lecture 18) that the input to 3−CNF − SAT is a logical formula ϕ of the form

$$\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$$

where each C_i is of the form

$$C_i = y_{i,1} \vee y_{i,2} \vee y_{i,3}$$

where each $y_{i,j}$ is a variable or the negation of a variable.

As an example

$$C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)\,,\ C_2 = (\neg x_1 \vee x_2 \vee x_3)\,,\ C_3 = (x_1 \vee x_2 \vee x_3)$$

We will define a polynomial transformation f from 3 − CNF − SAT to DCLIQUE

$$f : \phi \mapsto (G, k)$$

that builds a graph G and integer k such that (G, k) is a Yes-input to DCLIQUE if and only if ϕ is a Yes-input to 3 − CNF − SAT, i.e, ϕ is satisfiable.

Proof that DCLIQUE $\in \mathcal{NPC}$ **(cont)**

Suppose that ϕ is a $3-$CNF$-$SAT formula with n clauses, i.e., $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$. We start by setting $k = n.$
We now construct graph $G = (V, E).$

(I) For each clause $C_i = y_{i,1} \vee y_{i,2} \vee y_{i,3}$ we create 3 vertices, v_1^i, v_2^i, v_3^i, in V so G has $3n$ verices. We will *label* these vertices with the corresponding variable or variable negation that they represent. (Note that many vertices might share the same label)

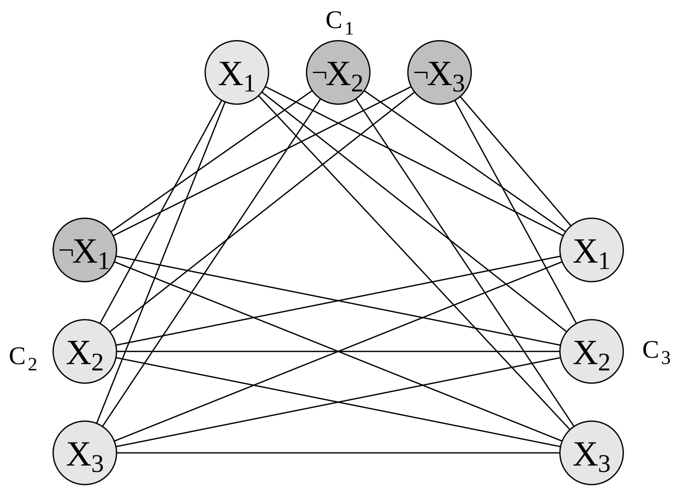
(II) We create an *edge* between vertices v_j^i and $v_{j'}^{i'}$ if and only if the following two conditions hold:
(a) v_j^i and $v_{j'}^{i'}$ are in different triples, i.e., $i \neq i'$, and
(b) v_j^i is not the *negation* of $v_{j'}^{i'}$.

On the next slide we will see an example of this construction

Proof that DCLIQUE $\in \mathcal{NPC}$ (cont)

Here is a formula $\phi = C_1 \wedge C_2 \wedge C_3$ and its corresponding graph:

$C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)\,,\; C_2 = (\neg x_1 \vee x_2 \vee x_3)\,,\; C_3 = (x_1 \vee x_2 \vee x_3)$



Note that the assignment $X_3 = T$, $X_2 = F$ satisfies ϕ (independent of the value of X_1). This corresponds to the clique of size 3 comprising the $\neg x_2$ node in C_1, the x_3 node in C_2 and the x_3 node in C_3.

So ϕ is satisfiable and G has a 3-clique.

Proof that DCLIQUE $\in \mathcal{NPC}$ (cont)

Theorem: A $3-$CNF formula ϕ with k clauses is satisfiable if and only if $f(\phi) = (G, k)$ is a Yes-input to DCLIQUE.

For the proof of this statement see page 1004 in CLRS.

Note that the graph G has $3k$ vertices and at most $3k(3k-1)/2$ edges and can be built in $O(k^2)$ time so f is a *Polynomial*-time reduction.

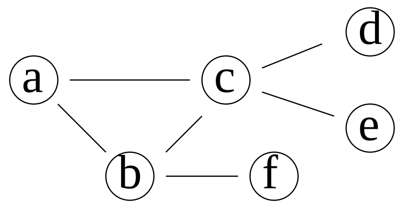
We have therefore just proven that

$$3-\text{CNF} \le_P \text{DCLIQUE}.$$

Since we already know that $3-$CNF $\in \mathcal{NPC}$ and have seen that DCLIQUE $\in \mathcal{NP}$ we have just proven that DCLIQUE $\in \mathcal{NPC}$.

$\boxed{\text{Problem: VC}}$

Vertex Cover: A *vertex cover* of G is a set of vertices such that every edge in G is incident to at least one of these vertices.

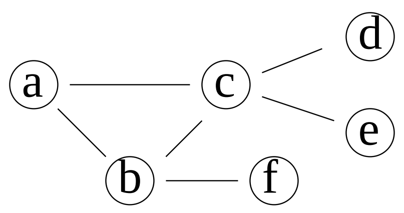


Find a vertex cover of G of size two

The Vertex Cover Problem (VC): Given a graph G, find a vertex cover of G of minimum size.

$\boxed{\mathcal{NPC}\ \textbf{Problem: DVC}}$

The Decision Vertex Cover Problem (DVC): Given a graph G and integer k, determine whether G has a vertex cover with k vertices.



Find a vertex cover of G of size two

Theorem: DVC $\in \mathcal{NPC}$.

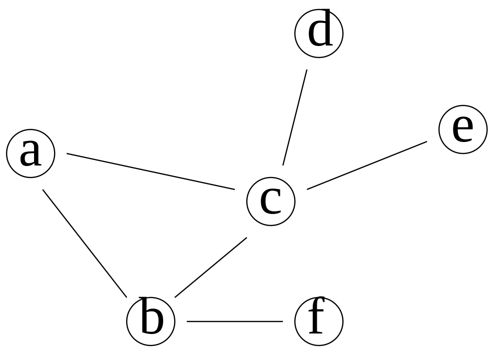
Proof: In Lecture 18 (slide 43), we showed that DVC $\in \mathcal{NP}$.

We show that DCLIQUE $\leq_P$ DVC. The conclusion then follows from the fact that DCLIQUE $\in \mathcal{NPC}$. A proof of DCLIQUE $\leq_P$ DVC will be given in the next three slides.

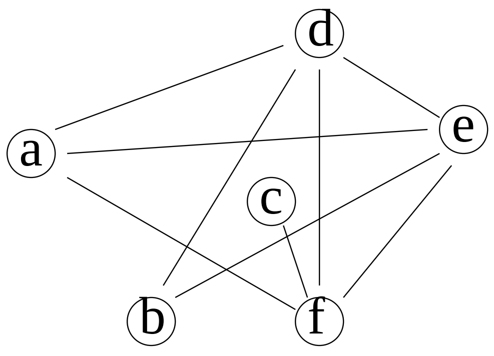
DVC $\in \mathcal{NPC}$: Complement of a Graph

The **complement of a graph** $G = (V, E)$ is defined by $\overline{G} = (V, \overline{E})$, where

$$\overline{E} = \{(u, v) \mid u, v \in V,\ u \neq v,\ (u, v) \notin E\}.$$



Graph G

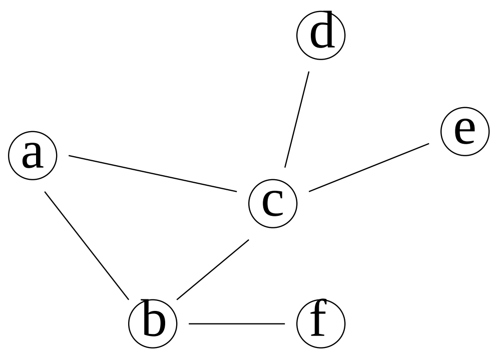


Complement of G

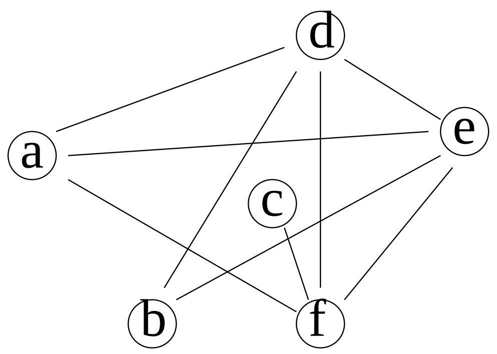
DVC $\in \mathcal{NPC}$: Vertex Cover Lemma

Lemma: A graph G has a vertex cover of size k if and only if the complement graph $\overline{G}$ has a clique of size $|V| - k$.

Proof: Let V' be a vertex cover in G and let $V'' = V \setminus V'$. If u and v are distinct vertices in V'', then they are not connected by an edge in E and so $(u, v) \in E'$. Hence V'' are the vertices of a clique in $\bar{G}$.
Similarly, if V'' is a clique in $\bar{G}$, then $V' = V \setminus V''$ is a vertex cover in G.



Graph G



Complement of G

$\boxed{\textbf{Proof: } \text{DVC} \in \mathcal{NPC}}$

Proof of DCLIQUE $\leq_P$ DVC:

Let $\overline{k} = |V| - k$. We define a transformation f from DCLIQUE to DVC:

$$f : (G, k) \mapsto (\overline{G}, \overline{k})$$

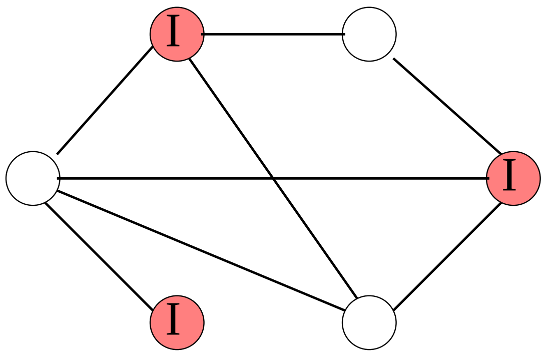
- f can be computed (that is, $\overline{G}$ and $\overline{k}$ can be determined) in time $O(|V|^2)$ time .
- f is a reduction, i.e., (G, k) is a Yes-input for DCLIQUE if and only if $f(G, k)$ is a Yes-input for DVC (by the Lemma in the previous slide).

Hence f is a polynomial-time reduction from DCLIQUE to DVC.

Remark: In this very special case f is also invertible and polynomial-time reduction from DVC to DCLIQUE as well.

Problem: Independent Set

Definition: An *independent set* is a subset I of vertices in an undirected graph G such that no pair of vertices in I is joined by an edge of G.



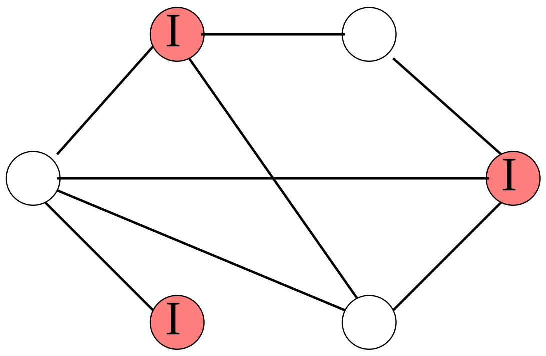
$\mathcal{NPC}$ Problem: Decision Independent Set (DIS)

Optimization Problem: Given an undirected graph G, find an independent set of maximum size.

Decision Problem (DIS): Given an undirected graph G and an integer k, does G contain an independent set consisting of k vertices?

Theorem: DVC $\in \mathcal{NPC}$.

Proof: It is very easy to see that DIS $\in \mathcal{NP}$. A certificate is a set of vertices $S \subseteq V$ and, in $O(|S|^2) = O(|V|^2)$ time we can check whether or not S is an independent set. In the next slide we will see that DCLIQUE $\leq_{\mathrm{P}}$ DIS, completing the proof.



$\boxed{\text{DIS} \in \mathcal{NPC}}$

Lemma: A graph G has an independent set of size k if and only if the complement graph $\overline{G}$ has a clique of size k.

Proof: Let $S \subseteq V$ be an independent set in G. If u and v are distinct vertices in S, then they are not connected by an edge in E and so $(u, v) \in E'$. Hence S are the vertices of a clique in $\bar{G}$.
Similarly, if S is a clique in $\bar{G}$, then S is an independent set in G.

We can now define a transformation from DCLIQUE to DIS:

$$f : (G, k) \mapsto (\overline{G}, k)$$

From the Lemma above we have that (G, k) is a Yes-input to DCLIQUE if and only if $f(G, k)$ is a Yes-input to DIS. Since f can be calculated in polynomial time we have just shown that DCLIQUE $\leq_{\text{P}}$ DIS and completed the proof that DIS $\in \mathcal{NPC}$.

Decision versus Optimization Problems

The theory of $\mathcal{NP}$-Completeness revolves around decision problems. It was set up this way because it's easier to compare the difficulty of decision problems than that of optimization problems.

At first glance, this might seem unhelpful since we usually don't care at all about decision problems. We're interested in finding an optimal solution to our problem (the optimization version) not whether such a solution exists (decision version).

In reality, though, being able to solve a decision problem in polynomial time will often permit us to solve the corresponding optimization problem in polynomial time (using a polynomial number of calls to the decision problem). So, discussing the difficulty of decision problems is often really equivalent to discussing the difficulty of optimization problems.

In the next two slides we see an example of this phenomenon for VERTEX COVER by showing that having a polynomial algorithm for Decision Vertex Cover (DVC) would yield a polynomial algorithm for fi nding a minimal Vertex Cover.

Decision versus Optimization Problems (cont)

Here are the two problems and third related one

VC: Given undirected graph G find a minimal size vertex cover.
DVC: Given undirected graph G and k, is there a vertex cover of size k?
MVC: Given an undirected graph G, find the size of a minimal vertex cover.

Suppose that $DVC(G, k)$ returns Yes if G has a vertex cover of size k and No, otherwise.

Consider the following algorithm for solving MVC:

```
$k = 0$;
while (not DVC($G, k$)) $k = k + 1$;
return($k$);
```

Note that MVC calls DVC at most $|V|$ times so, if there is a polynomial time algorithm for DVC, then our algorithm for MVC is also polynomial.

Decision versus Optimization Problems (cont)

Here is an algorithm for calculating $VC(G)$ that uses the algorithm for MVC on the previous page. First set $k = MVC(G)$ and then run $VC(G,k)$ which is defined by

```
$VC(G,t)$                find VC of size $t$
{   check all vertices in $G$ to find first vertex $u$ such that
        $MVC(G_u) == t-1$;
        such a vertex exists, why?
    Output $u$;
    if ($t > 1$)
        $VC(G_u, t-1)$;
}
```

(Show why this algorithm works).

Note that this algorithm calls MVC at most $|V|^2$ times so, if MVC is polynomial in $size(G)$, then so is VC.
We already saw that if DVC is polynomial in $size(G)$ so is MVC, so we've just shown that if we can solve DVC in polynomial time, we can solve VC in polynomial time.

NP-Hard Problems

A problem L is $\mathcal{NP}$-hard if some problem in $\mathcal{NPC}$ can be polynomially reduced to it (but L does not need to be in $\mathcal{NP}$).

In general, the Optimization versions of $\mathcal{NP}$-Complete problems are $\mathcal{NP}$-Hard.

For example, recall
VC: Given undirected graph G find a minimal size vertex cover.
DVC: Given undirected graph G and k, is there a vertex cover of size k?

If we can solve the optimization problem VC we can easily solve the decision problem DVC. Simply run VC on graph G and find a minimal vertex cover S. Now, given (G, k), solve $DVC(G, k)$ by checking whether $k \geq |S|$. If $k \geq |S|$ answer Yes, if not, answer No.