EECS660 Topic 7: Classification of Computational Problems Kong

Consider we are trying to compute/solve a given problem using a computer.

Easy vs. Hard Problems:

Even though a problem that can only be computed using a high degree polynomial time-bounded algorithm may not be considered as an "easy" problem by some of us, a problem that has no known polynomial algorithm or is only computable using exponential time algorithms is definitely a very hard problem to solve! If we adopt the usual wisdom that a problem is easy if it is not hard, then an easy problem is one that is computable in polynomial time and a hard problem is one that has no known polynomial algorithm or is only computable in exponential time

Q: How do we classify different problems according to their computability in polynomial time?

Let's consider two popular computer models in term of their speed of computing.
Two Computer Models:

1. **Primitive (Sequential) Computer:** Capable of executing only one instruction per clock-cycle.
2. **Ideal (Parallel) Computer:** Capable of executing infinite number of instructions per clock-cycle.

Let S_{PC} (S_{IC}) be the set of all computational problems that are computable in polynomial time using our primitive (ideal) computer.

Since we can allow all except one processor idle in our ideal computer, $S_{PC} \subseteq S_{IC}$.

Q: Is it true that $S_{IC} \subseteq S_{PC}$?

If true, $S_{PC} = S_{IC}$, then our ideal computer is actually no more powerful than any primitive computer in the sense that any problem that is computable in polynomial time using our ideal computer is also computable in polynomial time (possibly with a much higher degree) using any primitive computer! It seems to be unlikely but, until now, no one is able to prove or disprove that $S_{IC} \subseteq S_{PC}$.

Let's try to classify the problems in S_{PC}, which contains S_{IC} as a subset.

Consider the simplest type of computational problems, ***decision problems***, which are problem with only yes <u>or</u> no answer (not both).

Format in Specifying Decision Problems:

Instance: Specifying the input to problem

Question: Specifying a question to be answer.

Examples:

1. Let x be a Boolean variable with either true or false value. A literal of x is either x itself or its negation $\overline{x}$. Given Boolean variables $x_1, x_2, \ldots, x_n$, a Boolean expression (function) $E(x_1, x_2, \ldots, x_n)$ is an expression containing the literals of $x_1, x_2, \ldots, x_n$ and some logical connectives such as conjunction (·), disjunction (+), negation $(\overline{\ })$, etc. $E(x_1, x_2, \ldots, x_n)$ is in ***conjunctive normal form*** (CNF) if $E(x_1, x_2, \ldots, x_n) = C_1C_2\ldots C_m$, where C_i is a disjunction of $\leq$ n literals (*clause*) in which each literal of x_i can appear at most once, $1 \leq j \leq m$, $1 \leq i \leq n$. If there exists an assignment of truth values to the variables $x_1, x_2, \ldots, x_n$ in a Boolean expression $E(x_1, x_2, \ldots, x_n)$ such that $E(x_1, x_2, \ldots, x_n)$ = true, then $E(x_1, x_2, \ldots, x_n)$ is ***satisfiable***; otherwise, $E(x_1, x_2, \ldots, x_n)$ is ***unsatisfiable***.

   Examples:

   $E(x_1, x_2, x_3) = (x_1 + \overline{x_2} + \overline{x_3})(\overline{x_1} + x_2)(\overline{x_1} + \overline{x_2} + x_3)$ is satisfiable by assigning $x_1 = \mathit{false}, x_2 = \mathit{false}, x_3 = \mathit{true}.$

   $E(x_1, x_2, x_3) = (x_1)(\overline{x_1} + x_2)(\overline{x_3})(\overline{x_1} + \overline{x_2} + x_3)$ is unsatisfiable.

   The Satisfiability (SAT) Problem:

   Instance: Given a Boolean expression $E(x_1, x_2, \ldots, x_n)$ with m clauses in CNF.

   Question: Is $E(x_1, x_2, \ldots, x_n)$ satisfiable?

2. **The k-Satisfiability (k-SAT) Problem:**

   Instance: Given a Boolean expression $E(x_1, x_2, \ldots, x_n)$ with m clauses in CNF such that each clause has exactly k literals, $k \geq 3$.

   Question: Is $E(x_1, x_2, \ldots, x_n)$ satisfiable?

3. **The Vertex Coloring Problem:**

   Instance: Given an undirected graph G = (V,E) and a positive integer $k \leq |V|$.

   Question: Does G contain a proper coloring of V using $\leq$ k colors such that no two adjacent vertices can receive the same color?

4. **The Clique Problem:**

   Instance: Given an undirected graph G = (V,E) and a positive integer $k \leq |V|$.

   Question: Does G contain a clique, which is a complete subgraph of G, of size $\geq$ k?

5. **The Dominating Set Problem:**

   Instance: Given an undirected graph G = (V,E) and a positive integer $k \leq |V|$.

   Question: Does G contain a dominating set of size $\geq$ k?

6. **The Eulerian Cycle Problem:**
   Instance: Given an undirected graph G = (V,E).
   Question: Does G contain an Eulerian cycle?

7. **The Hamiltonian Cycle Problem:**
   Instance: Given an undirected graph G = (V,E).
   Question: Does G contain a Hamiltonian cycle?

8. **The Traveling Salesperson Problem (TSP):**
   Instance: Given a complete graph G = (V,E), a cost function c: E $\rightarrow$ R^+, and a cost k $\in$ R^+.
   Question: Does G contain a traveling sales person tour, which is a Hamiltonian cycle in G, T such that $\sum_{e \in E(T)} c(e) \geq k$?

9. **The Shortest Path Problem:**
   Instance: Given a graph G = (V,E), cost function c: E $\rightarrow$ R^+, cost k > 0, and s, t $\in$ V.
   Question: Does G contain a path P from s to t such that $\sum_{e \in E(P)} c(e) \leq k$?

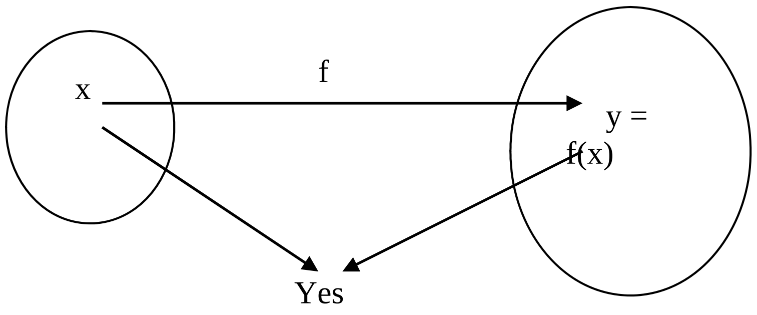
10. **The Longest Path Problem:**
    Instance: Given a graph G = (V,E), cost function c: E $\rightarrow$ R^+, cost k > 0, and s, t $\in$ V.
    Question: Does G contain a simple path P from s to t such that $\sum_{e \in E(P)} c(e) \geq k$?

Decision vs. Optimization Problems:

Many optimization problems can be reformulated as a decision problem. Using the solution obtained from solving the optimization problem, the corresponding decision problem can be solved easily. However, even though the decision version of a problem is solvable in polynomial time, its corresponding optimization problem still may not be solvable in polynomial time!

Comparing Decision Problems:

Let D_1 and D_2 be decision problems. Assume that there exists a polynomial transformation (function) f: domain(D_1) $\rightarrow$ domain(D_2) such that $\forall$ x $\in$ domain(D_1) $\exists$ y $\in$ domain(D_2), y = f(x), and (D_1,x) has yes-answer iff (D_2,y) has yes-answer.



Q: Which one of these two decision problems is harder than the other?

Observe that if D_2 is computable in polynomial time, we can also solve D_1 in polynomial time by first transforming an instance (D_1,x) of D_1 to a corresponding instance (D_2,y) of D_2 and then solve (D_2,y) for the solution of (D_1,x). Hence, if D_2 is an easy problem, D_1 must also be an easy problem. Hence, we can conclude that D_2 is at least as hard as D_1.

When a polynomial transformation exists from D_1 to D_2 satisfying the above condition, D_1 is said to be ***polynomially reducible*** to D_2, denoted by $D_1 \propto D_2$, and the construction of (D_2,y) from (D_1,x) is called a ***reduction***.

Theorem: Let D be the set of all decision problems and $\propto$ be a relation defined on D. The relation $\propto$ is both reflexive and transitive.

Formalizing and Classifying Easy & Hard Decision Problems:

Primitive computer	$\rightarrow$	Deterministic Turing Machine (DTM)
Ideal computer	$\rightarrow$	Nondeterministic Turing Machine (NDTM)

P class: The set of all decision problems that is computable in $O(n^{O(1)})$ time (polynomial time) using a DTM.

NP class: The set of all decision problems that is computable in $O(n^{O(1)})$ time (polynomial time) using a NDTM.

Remark: Observe that a decision problem $\pi \in P$ implies that
(1) π is a decision problem, and
(2) π is computable in polynomial time using a DTM.
Hence, all decision problems in P can be considered as easy problems!

Q: Are all problems in NP-class hard?

Since $P \subseteq NP$, no!

Q: Are all problems in NP-class easy?

Don't know yet! If true, $NP \subseteq P$, which implies that $P = NP$ and our powerful theoretical computer is actually no better than a primitive computer in terms of computing efficiency!

Goal: We would like to be able to determine whether a given problem is hard so that we will not invest too much time searching for simple polynomial-time algorithm for solving it.

Q: If indeed there are some problems in NP that are harder than the others, what are the hardest problems in NP?

Recall that if $\pi \in$ NP, and if all other problems in NP are polynomially reducible to π, then π must be at least as hard as any problem in NP. This class of problems, called ***NP-Complete*** (NPC) class, contains the hardest problems in NP.

NP-Complete class:

The set of all decision problems π in NP such that $\forall\ \pi^* \in$ NP, $\pi^* \propto \pi$.

Remark: If $\exists\ \pi \in$ NPC $\cap$ P, then all problems in NP must also be P, implying that P = NP. Also, the same polynomial time algorithm for solving π can be used to solve all other problems in NPC.

Q: Is NPC = $\emptyset$?

No!

Proving $\pi \in$ NPC:

Two steps:

1. Prove that $\pi \in$ NP.
2. Prove that $\forall\ \pi^* \in$ NP, $\pi^* \propto \pi$.

To prove that $\pi \in$ NP, we need to show that π is computable in polynomial time using a NDTM. Equivalently, we can show that there exists a polynomial time nondeterministic algorithm for solving π.

Designing Nondeterministic Algorithms:

Three Primitive Operations:

1. ***Choice(S;P)***: The function Choice() generates all possible subsets from S satisfying a given property P. Once selected, a subset will then start its own computational process.
2. ***Success***: This Boolean function will terminate all computational processes and return true-value if the current process has a yes-answer.
3. ***Failure***: This Boolean function will terminate the current computational process and return true-value if the current process has a no-answer.

In the nondeterministic computing model, all these operations can be executed in $\Theta(1)$ (constant) time.

Examples:

1. SAT $\in$ NP.

   NP-algorithm for SAT-problem:

```
(x_1, x_2,…, x_n) ← Choice(B×B×…×B; B = {T,F});   //  Θ(1); #possible sets = $2^n$
if E(x_1, x_2,…, x_n) = T                          //  O(mn)
    then Success                                   //  Θ(1)
    else Failure                                   //  Θ(1)
endif;
```

   Complexity analysis:
   T(m,n) = cost(selection) + cost(verification) = O(mn).

2. The Clique Problem $\in$ NP.

   NP-algorithm for the Clique Problem:

```
C ← Choice(V; |C| = k);                            //  Θ(1); #possible sets = $\binom{n}{k}$
if C forms a clique                                //  O($k^2$)
    then Success                                   //  Θ(1)
    else Failure                                   //  Θ(1)
endif;
```

   Verification of clique:

```
clique = true;
for all u ∈ C do
    for all v ∈ C, u ≠ v, do
        if (u,v) ∉ E
            then  clique = false;
                  break;
        endif;
    endfor;
endfor;
if clique
    then return true
    else return false
endif;
```

Complexity analysis:
   Let |V| = n.
   T(n) = cost(selection) + cost(verification) = $O(k^2) \in O(n^2)$.

3. The Dominating Set Problem $\in$ NP.

```
NP-algorithm for the Dominating Set Problem:
D ← Choice(V; |D| = k);                        //  Θ(1); #possible sets = $\binom{n}{k}$
if D forms a dominating set                    //  O(nk)
    then Success                               //  Θ(1)
    else Failure                               //  Θ(1)
endif;

Verification of dominating set:
for all u ∈ V do
    dominate = true;
    if u ∉ D
        then  dominate = false;
              for all w ∈ D do
                  if (u,w) ∈ E
                      then  dominate = true;
                  endif;
              endfor;
    endif;
    if dominate = false
        then break
    endif;
endfor;
if dominate
    then return true
    else return false
endif;
```

Complexity analysis:

Let $|V| = n$.

$T(n) = \text{cost(selection)} + \text{cost(verification)} = O(nk) \in O(n^2)$.

Analyzing Nondeterministic Algorithm:

Observe that a nondeterministic algorithm consists of two parts:

(i) (Nondeterministic) Selection of subsets for testing.

(ii) (Deterministic) Verification.

$T(n) = \text{cost(selection)} + \text{cost(verification)}$

Since cost(selection) = $\Theta(1)$ in our nondeterministic model, T(n) will depend solely on the cost in verifying a given set for possible solution. Hence, the nondeterministic algorithm is polynomial time-bounded iff a solution can be verified in polynomial time deterministically.

Alternate Definition for NP Class:

NP is the set of all decision problems whose solution can be verified in polynomial time using a DTM.

Proving Polynomial Reduction:

Since we do not know the exact structure of NP, it is very difficult to prove the statement that $\forall \pi^* \in$ NP, $\pi^* \propto \pi$. By using the transitivity property of $\propto$, however, we can obtain the following useful theorem.

Theorem: Let $\pi \in$ NP. If $\exists \pi^+ \in$ NPC such that $\pi^+ \propto \pi$, then $\forall \pi^* \in$ NP, $\pi^* \propto \pi$.
Proof. Assume that $\exists \pi^+ \in$ NPC such that $\pi^+ \propto \pi$. By the definition of NPC class, $\pi^+ \in$ NPC implies that $\forall \pi^* \in$ NP, $\pi^* \propto \pi^+$. Since $\pi^* \propto \pi^+$ and $\pi^+ \propto \pi$, by transitivity, $\pi^* \propto \pi$.

Corollary: A decision problem $\pi \in$ NP $\in$ NPC iff
(1) $\pi \in$ NP, and
(2) $\exists \pi^+ \in$ NPC such that $\pi^+ \propto \pi$.

Remark: In order to use the above result in proving the NP-Completeness of a decision problem, we must have at least one NPC problem that has been proven using the original definition of NP-Completeness. There are indeed only two NPC problems, the ***Satisfiability Problem*** (Cook, 1971) and certain ***Tiling Problem*** (Levin, 1973), known to be proven using the original definition. Based on Cook's result, Karp (1972) proved that 21 well-known problems are also in NPC.

Theorem: SAT $\in$ NPC.

S. Cook, *The Complexity of Theorem Proving Procedures*, Proc. Third ACM Symp. Theory of Computing, pages 151-158, 1971.

L.A. Levin, *Universal Sorting Problems*, Problemy Peredachi Informatsii, 9(3): 265-266, 1973.

R.M. Karp, *Reducibility among Combinatorial Problems*, Complexity of Computer Computations, Ed. by R.E. Miller & J.W. Thatcher, Plenum Press, 1972.

NP-Completeness Properties for Satisfiability Problems:

(1) SAT $\in$ NPC.
(2) kSAT $\in$ NPC, $k \geq 3$..
(3) 3SAT $\in$ NPC.
(4) 2SAT $\in$ P.

Theorem: 3SAT $\in$ NPC.
Proof. To show that 3SAT $\in$ NP, we can use the same NP-algorithm as in the SAT problem. To complete our proof, we will show that SAT $\propto$ 3SAT. More specifically, we will prove that for any given instance of the SAT problem, there is a corresponding instance of the kSAT problem such that the given instance of the SAT problem has a yes-answer iff the corresponding instance of the kSAT problem has a yes-answer.
Let $E(x_1, x_2, ..., x_n) = C_1C_2...C_m$ be any Boolean expression with m clauses in CNF. We will construct a corresponding Boolean expression $E^*(x_1, x_2, ..., x_n, y_1, y_2, ..., y_\alpha) = C_1^*C_2^*...C_m^*$ in CNF such that each $C_i^*, 1 \le i \le m,$ is a conjunction of clause(s) having exactly three literals, and $E(x_1, x_2, ..., x_n)$ has a yes-answer iff $E^*(x_1, x_2, ..., x_n, y_1, y_2, ..., y_\alpha)$ has a yes-answer.

For each clause $C_i, 1 \le i \le m,$ in $E(x_1, x_2, ..., x_n),$ we will transform it into a conjunction of clause(s) in $E(x_1, x_2, ..., x_n, y_1, y_2, ..., y_\alpha)$ according to the number of literals in C_i .

Transformation:

Instance of SAT	**Instance of 3SAT**
$E(x_1, x_2, ..., x_n) = C_1C_2...C_m$	$E^*(x_1, x_2, ..., x_n, y_1, y_2, ..., y_\alpha) = C_1^*C_2^*...C_m^*$

Case 1: If $C_i = x_{i1},$ then introduce new Boolean variables y_1, y_2 such that
$C_i^* = (x_{i1} + y_1 + y_2)(x_{i1} + y_1 + \overline{y_2})(x_{i1} + \overline{y_1} + y_2)(x_{i1} + \overline{y_1} + \overline{y_2})$.
Case 2: If $C_i = (x_{i1} + x_{i2})$, then introduce new Boolean variable y such that
$C_i^* = (x_{i1} + x_{i2} + y)(x_{i1} + x_{i2} + \overline{y})$.
Case 3: If $C_i = (x_{i1} + x_{i2} + x_{i3})$, then $C_i^* = (x_{i1} + x_{i2} + x_{i3})$.
Case 4: If $C_i = (x_{i1} + x_{i2} + ... + x_{i\beta}), \beta > 3,$ then introduce $\beta - 3$ new Boolean variable $y_1, y_2, ..., y_{\beta-2}$ such that

$$C_i^* = (x_{i1} + x_{i2} + y_1)(x_{i3} + \overline{y_1} + y_2)(x_{i4} + \overline{y_2} + y_3)...$$
$$(x_{i,s-1} + \overline{y_{s-3}} + y_{s-2})(x_{i,s} + \overline{y_{s-2}} + y_{s-1})(x_{i,s+1} + \overline{y_{s-1}} + y_s)...$$
$$(x_{i,\beta-2} + \overline{y_{\beta-4}} + y_{\beta-3})(x_{i,\beta-1} + x_{i,\beta} + \overline{y_{\beta-3}}).$$

This transformation from C_i to C_i^* can be carried out in polynomial time since, in the worst (Case 4), $\beta - 3$ new variables are introduced, resulting in $\beta - 2$ clauses, each of which having exactly three literals. Hence, by applying the above transformation to all $C_i, 1 \le i \le m,$ we can transform $E(x_1, x_2, ..., x_n)$ to $E(x_1, x_2, ..., x_n, y_1, y_2, ..., y_\alpha)$ by introducing α new variables, where α is the sum of all new variables in the construction. Hence, this transformation is a polynomial transformation.

To complete the proof that SAT $\propto$ 3SAT, we must show that $E(x_1, x_2, ..., x_n)$ has a yes-answer iff $E(x_1, x_2, ..., x_n, y_1, y_2, ..., y_\alpha)$ has a yes-answer. We can prove this assertion by proving that $C_i = true$ iff $\mathrm{C}_\mathrm{i}^* = true$ in the above construction. Observe that the first three cases in the above transformation can be verified easily. We will prove only Case 4.

($\rightarrow$): Assume that $\beta > 3, C_i = (x_{i1} + x_{i2} + ... + x_{i\beta}) = true.$ Hence, at least one of the literals $x_{i1}, x_{i2}, ..., x_{i\beta}$ must be $true$. If $x_{i1} = true$ or $x_{i2} = true,$ then we can assign $y_1 = y_2 = ... = y_{\beta-3} = false,$ forcing $\mathrm{C}_\mathrm{i}^* = true.$ Assume that $x_{i1} = x_{i2} = false,$ and $x_{is} = true, 3 \le s \le \beta.$ By assigning $y_1 = y_2 = ... = y_{s-2} = true$ and $y_{s-1} = y_s = ... = y_{\beta-3} = false,$ $\mathrm{C}_\mathrm{i}^* = true.$

($\leftarrow$): Assume that

$C_i^* = (x_{i1} + x_{i2} + y_1)(x_{i3} + \overline{y_1} + y_2)(x_{i4} + \overline{y_2} + y_3)...$

$(x_{i,s-1} + \overline{y_{s-3}} + y_{s-2})(x_{i,s} + \overline{y_{s-2}} + y_{s-1})(x_{i,s+1} + \overline{y_{s-1}} + y_s)...$

$(x_{i,\beta-2} + \overline{y_{\beta-4}} + y_{\beta-3})(x_{i,\beta-1} + x_{i,\beta} + \overline{y_{\beta-3}}).$

If $y_1 = false,$ then either $x_{i1} = true$ or $x_{i2} = true,$ forcing $C_i = true.$

If $y_1 = true$ and $y_2 = false,$ then $x_{i3} = true,$ forcing $C_i = true.$

If $y_2 = true$ and $y_3 = false,$ then $x_{i4} = true,$ forcing $C_i = true.$

…

If $y_{\beta-4} = true$ and $y_{\beta-3} = false,$ then $x_{i,\beta-2} = true,$ forcing $C_i = true.$

However, if $y_{\beta-3} = true,$ then either $x_{i,\beta-1} = true$ or $x_{i,\beta} = true,$ forcing $C_i = true.$

Hence, if $\mathrm{C}_\mathrm{i}^* = true,$ $\mathrm{C}_\mathrm{i} = true.$

Q.E.D.

Example: A polynomial reduction from SAT to 3SAT problem.

Let $E(x_1, x_2, x_3, x_4, x_5) = (x_1)(\overline{x_1} + x_3)(\overline{x_1} + \overline{x_2} + x_4 + \overline{x_5})(x_1 + x_2 + \overline{x_3} + x_4 + \overline{x_5})$ **.**

$E^*(x_1, x_2, x_3, x_4, x_5, y_1, y_2, y_3, y_4, y_5, y_6) =$

$(x_1 + y_1 + y_2)(x_1 + y_1 + \overline{y_2})(x_1 + \overline{y_1} + y_2)(x_1 + \overline{y_1} + \overline{y_2})$

$(\overline{x_1} + x_3 + y_3)(\overline{x_1} + x_3 + \overline{y_3})$

$(\overline{x_1} + \overline{x_2} + y_4)(x_4 + \overline{x_5} + \overline{y_4})$

$(x_1 + x_2 + y_5)(\overline{x_3} + \overline{y_5} + y_6)(x_4 + \overline{x_5} + \overline{y_6}).$

Theorem: The Clique Problem $\in$ NPC.
Proof. From previous discussion, we know that the Clique Problem $\in$ NP. We will show that SAT $\propto$ Clique Problem.
Given an instance of the SAT problem, which is a Boolean expression $E(x_1, x_2, ..., x_n) = C_1C_2...C_q$ in n variables and q clauses such that each clause has exactly three literals. We will construct an instant for the Clique Problem, which is an undirected graph G = (V,A) and a positive integer k ≤ |V| such that $E(x_1, x_2, ..., x_n)$ is satisfiable iff G has a clique of size k.

Transformation:
Let $E(x_1, x_2, ..., x_n) = C_1C_2...C_q$ with $C_i = x_{i,1} + x_{i,2} + ... + x_{i,m_i}, 1 \le i \le q.$

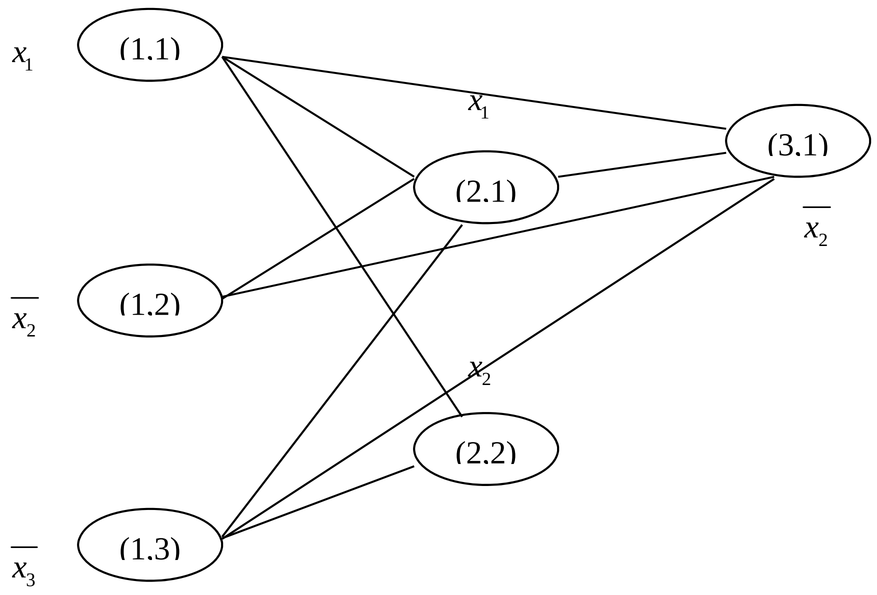
Construct an undirected graph G = (V,A) as follows:
V = {(i,α(i)) : 1 ≤ i ≤ q, 1 ≤ α(i) ≤ m$_i$},
A = {((i,j),(s,t)) : i ≠ s and $x_{i,j} \ne \overline{x_{s,t}}$ }.
Also, take k = q.

Since |V| = $\sum_{i=1}^{q} m_i = n,$ which equals to the total number of literals in $E(x_1, x_2, ..., x_n),$ and |A| ≤ $\frac{n(n-2)}{2},$ the graph G can be constructed in polynomial time.
Hence, this is a polynomial transformation from SAT to the Clique Problem.

We will now prove that $E(x_1, x_2, ..., x_n)$ is satisfiable iff G has a clique of size q.
(→): Assume that $E(x_1, x_2, ..., x_n)$ is satisfiable. Hence, there exists $x_{i,\alpha_i} \in C_i,\ x_{i,\alpha_i} = true,$ $1 \le i \le q, 1 \le \alpha_i \le m_i.$ Define $C = \{(i,\alpha_i) : 1 \le i \le q\}.$ By construction, $C \subseteq$ V and C must form a clique in G. If not, there must exist two vertices in $C,$ say (i,α_i) and (j,α_j), such that ((i,α_i),(j,α_j)) ∉ A. By the construction of G, $x_{i,\alpha_i} = \overline{x_{j,\alpha_j}}$. Hence, x_{i,α_i} and x_{j,α_j} can not be true at the same time, which is a contradiction to the fact that $x_{i,\alpha_i} = true,$ $1 \le i \le q, 1 \le \alpha_i \le m_i.$
(←): Assume that C = {v$_1$, v$_2$, …, v$_q$} is a clique in G with q vertices. By the construction of G, no two vertices in C can come from the same clause; otherwise, their first component will be the same. Hence, C = {(1,α_1), (2,α_2), …, (q,α_q)}. Since $x_{i,\alpha_i} \in C_i, 1 \le i \le q, 1 \le \alpha_i \le m_i,$ by assigning $x_{i,\alpha_i} = true,$ $E(x_1, x_2, ..., x_n) = C_1C_2...C_q = true.$
Q.E.D.

Example: A polynomial reduction from SAT to Clique problem.
Let $E(x_1, x_2, x_3) = (x_1 + \overline{x_2} + \overline{x_3})(x_1 + x_2)(\overline{x_2})$.

G = (V,A) with V = {(1,1), (1,2), (1,3), (2,1), (2,2),(3,1)}



Proving NP-Completeness of π:

1. Prove that π ∈ NP by showing that a solution of π can be verified in polynomial time. *You must show your algorithm and then prove that T(n) = O($n^{O(1)}$).*
2. Choose a problem π^+ ∈ NPC and prove that $\pi^+ \propto \pi$.
   (i) Construct an algorithm to transform any given instance of π^+ to a corresponding instance of π.
   (ii) Prove that the transformation can be carried out in polynomial time.
   (iii)Prove that an instance of π^+ has yes-answer **iff** its corresponding instance of π must also have yes-answer.