ICCS313 - Algorithm Analysis, T1 2019/20

Lecture 11: Selected NP-Complete Problems

23 November 2019

Lecturer: Chaya Hiruncharoenvate *Scribe: -*

```
* Final Exam next week, 30 November
* 8:00-9:30 optional review session (bring questions)
* 9:30-11:00 Final Exam (one A4 cheat sheet allowed)
```

1 Circuit SAT: The Mother of NP-Complete

Given a circuit of NAND gates with a single output and no loops (some of the inputs may be hardwired).

Question: is there a setting of the inputs that causes the circuit to output 1?

Proof: First of all, it is clearly in NP, since you can just guess the input and try it.

Now, suppose some problem A is in NP. That means there is a polynomial-time verifier V that takes in an instance x and a solution/proof y, and then checks to see if y is really a solution/proof for x. (E.g., x is a graph and y is a tour). In other words, if x is a NO instance, then there should not exist any y such that $V(x, y) = 1$. But, if x is a YES instance, then there should exist some y that causes $V(x, y)$ to output 1.

We are now going to use V to reduce problem A to CIRCUIT-SAT. Specifically, given some instance x, we want to create a circuit C_x such that $C_x(y) = V(x, y)$. If we can do this, then C_x is a YES-instance of CIRCUIT-SAT (i.e., there exists y such that $C_x(y) = 1$) iff x was a YES-instance of A (i.e., there exists y such that $V(x, y) = 1$).

How are we going to convert V and x into a circuit C_x? If we think of V as a computer program, we can do this by unrolling the loops of V, and having each layer of the circuit encode one time step of V's computation. The width of C_x will be the amount of memory used by V, and the depth of C_x will be the amount of time used by V. (Remember that a computer is just NAND gates with a clock, which means we can encode one step of computation by one layer of circuit.) Since V is polytime, this circuit will have polynomial size. We then hardwire the "x" part of the input wires to our given instance "x", producing the circuit C_x.

We have now shown how to reduce any arbitrary problem A in NP to CIRCUIT-SAT. In other words, if we had an oracle for CIRCUIT-SAT, we could use it to solve any problem A in NP: given an input x we run the above procedure to produce C_x and then feed C_x into our oracle. I.e., we have shown that CIRCUIT-SAT is NP-complete. ■

2 3SAT

3SAT is also NP-Complete

Proof: This reduction is important (and a little tricky) because 3-SAT looks so much simpler than Circuit-SAT. Nonetheless, we're going to prove that any algorithm to solve 3-SAT could be used to solve general circuit-SAT too. To prove this, we want to show how we can take a circuit-SAT instance (i.e., a circuit C where we want to know if there exists an input z that makes C(z)=1) and convert it into a 3-SAT instance that is satisfiable *iff* the circuit we were given is satisfiable.

REDUCTION: We will create one variable for each *wire* in the circuit C. Say we have some gate g whose inputs are wires y_1 and y_2, and whose output wire is y_3. Since g is a NAND gate, we want to make sure that the output is NOT the AND of the two inputs. We can write this as:

```
    (y1 OR y2 OR y3)       // if y1=0 and y2=0 then we can't have y3=0
AND (y1 OR not(y2) OR y3) // if y1=0 and y2=1 then we can't have y3=0
AND (not(y1) OR y2 OR y3) // if y1=1 and y2=0 then we can't have y3=0
AND (not(y1) OR not(y2) OR not(y3))  // can't have all three = 1.
```

(draw a table of y1, y2, and y3 to accompany this)

We will do this for all gates in the circuit. Now, we will also have variables $z_1, \ldots, z_n$ for the input bits to C, and force any wire y_j connected to input z_i to have the same value as variable z_i: (y_j OR not(z_i)) AND (not(y_j) or z_i) Finally, we add one more clause saying that we want the output to be 1. This is easy: if y_m is the output wire, then we just add the clause (y_m).

Then, for any clause, we do the following:

- If it has 3 distinct literals, simply include it in the formula.
- It it has 2 distinct literals, $C_i = (l_1 \vee l_2)$, then include $(l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$ in the formula.
- If it has 1 distinct literal, $C_i = l$, then include $(l \vee p \vee q) \wedge (l \vee p \vee \neg q) \wedge (l \vee \neg p \vee q) \wedge (l \vee \neg p \vee \neg q)$ in the formula.

In other words, we are asking: is there an input and a setting of all the wires such that the output of the circuit is equal to 1, and each gate is doing what it's supposed to? So, the 3-CNF formula produced is satisfiable if and only if the circuit has a setting of inputs that causes it to output 1. The size of the formula is polynomial (actually, linear) in the size of the circuit. The construction can be done in polynomial (actually, linear) time. So, if we had a polynomial-time algorithm to solve 3-SAT, then we could solve circuit-SAT in poly time too. ■

3 Clique

- A *clique* in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of, each pair of which is connected by an edge in E.
- IE, a clique is a complete subgraph of G.
- The *size* of a clique is the number of vertices it contains.
- The *clique problem* is the optimization problem of finding a clique of maximum size in a graph.
- A decision problem: Does a clique of a given size k exists in the graph? The formal definition is:

$$\text{CLIQUE} = \{< G, k >: G \text{ is a graph containing a clique of size } k\}$$

Theorem 3.1. *The clique problem is NP-complete.*

Proof:

- First, we need to show that CLIQUE $\in$ NP. IE, we need to show that given a certificate, $V' \subseteq V$, V' is a clique of size k.

  We can do this in polynomial time by checking whether, for each pair $u, v \in V$, the edge $(u, v) \in E$.

- Next, we prove that 3-SAT $\leq_P$ CLIQUE. IE, prove that CLIQUE is NP-hard.

  Given a 3-CNF problem instance $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ with k clauses. Each clause C_r has exactly three distinct literals $l_1^r, l_2^r, and l_3^r$. We will construct a graph G such that ϕ is satisfiable iff G has a clique of size k.

  We construct the graph G as follows:

  1. For each clause $C_r = (l_1^r \vee l_2^r \vee l_3^r)$, we place a triple of vertices $v_1^r, v_2^r, and v_3^r$ into V. We put an edge between two vertices $v_i^r and v_j^s$ if both of the following hold:
     - $v_i^r and v_j^s$ are in different triples, that is $r \neq s$ and
     - their corresponding literals are *consistent*, that is l_i^r is not the negation of l_j^s
  2. We can build the graph from ϕ in polynomial time.

  For example:

  $$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

  Then, graph G is as shown in Figure 1

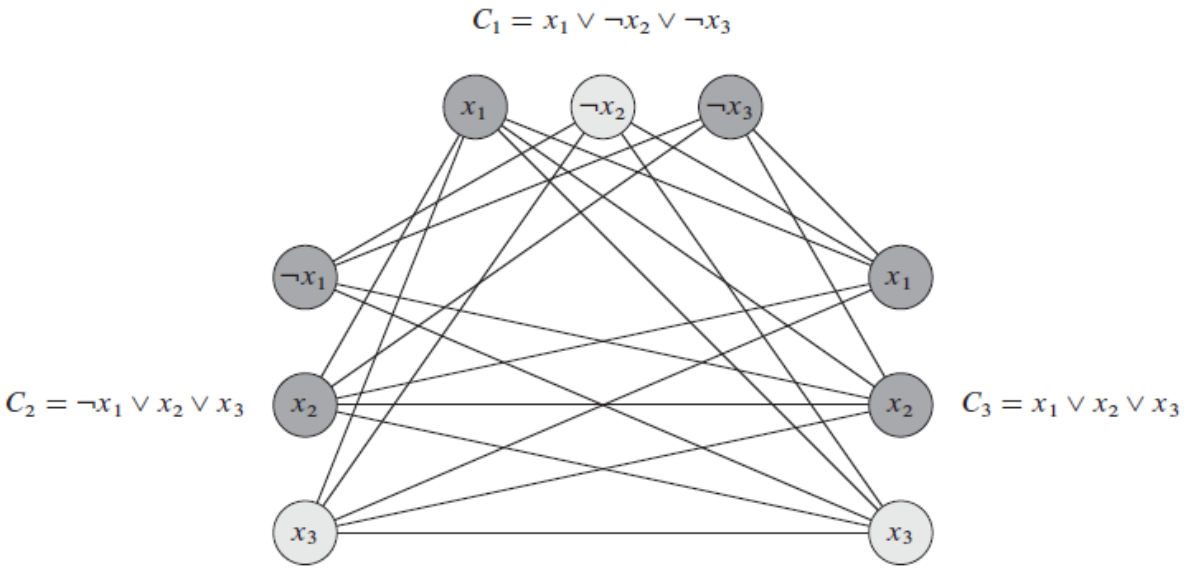

Figure 1: CLIQUE problem construction from 3SAT

- We must show that this transformation is a reduction.

  1. First, suppose ϕ has a satisfying assignment. Then, each clause C_r contains at least one literal l_i^r that is assigned 1, and each such literal corresponds to a vertex V_i^r.
     Picking one such "true" literal from each clauses yields a set V' of k vertices. (We will prove that V' is a clique.)

For any two vertices $v_i^r, v_j^s \in V'$ where $r \neq s$, both corresponding literals l_i^r and l_j^s map to 1 by the given satisfying assignment $\implies$ the literals cannot be complements. Thus, by construction of G, the edge $(v_i^r, v_j^s) \in E$

2. Conversely, suppose G has a clique V' of size k. No edges in G connect vertices in the same triple $\implies$ V' contains exactly one vertex per triple.

   We can assign 1 to each literal l_i^r such that $v_i^r \in V'$ without worrying that we will assign 1 to both a literal and its compliment, because G contains no edges between inconsistent literals.

   Each clause is satisfied $\implies$ ϕ is satisfied.

4 Vertex Cover

A *vertex cover* of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that $(u, v) \in E \implies u \in V' \vee v \in V'$ (or both). That is, a vertex cover for G is a subset of vertices that covers all the edges in E. The *size* of a vertex cover is the number of vertices in it. Figure 2(b) has a vertex cover w, z of size 2.

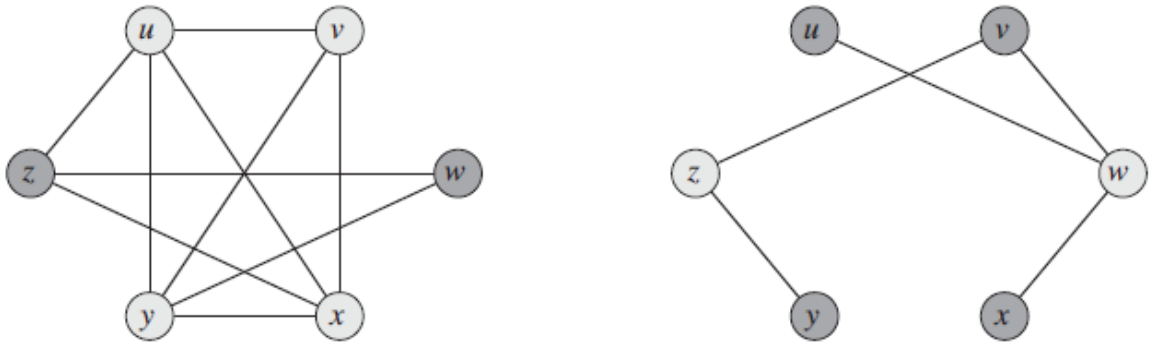

Figure 2: (a) G with clique $V' = u, v, x, y$ (b) $\overline{G}$ produced by the reduction algorithm that has vertex cover $V \setminus V' = \{w, z\}$

■

The *vertex-cover* problem (VC) is to find a vertex cover of minimum size in a given graph.
Decision problem: G has a vertex cover of size k

Theorem 4.1. *VC is NP-complete*

Proof:

- Show that VC $\in$ NP. Given $G = (V, E), k$, the certificate is the vertex cover $V' \subseteq V$. Verify by check that $|V'| = k$ and check for each edge $(u, v) \in E$, $u \in V'$ or $v \in V'$. We can do the verification in polynomial time.

- Show that VC is NP-hard by showing that CLIQUE $\leq_P$ VC.

  We first define the notion of the *complement* of a graph. $\overline{G} = (V, \overline{E})$ where $\overline{E} = \{(u, v) : u, v \in V, u \neq v, (u, v) \notin E\}$

Reduction: Take $< G, k >$ instance of CLIQUE, compute $\overline{G}$ (polynomial time). $< \overline{G}, |V| - k >$ is an instance of VC. Showing that this is an actual reduction:

1. Suppose G has a clique $V' \subseteq V$ with $|V'| = k$. We claim that $V \setminus V'$ is a vertex cover in $\overline{G}$.
   Let $(u, v) \in \overline{E} \implies (u, v) \notin E \implies$ at least one of $u, v \notin V'$.
   Equivalently, at least one of $u, v \in V \setminus V' \implies (u, v)$ is covered by $V \setminus V'$
   Since (u, v) were chose arbitrarily from $\overline{E} \implies$ every edge in $\overline{E}$ is covered by a vertex in $V \setminus V' \implies V \setminus V'$ which as size $|V| - k$ forms a vertex cover for $\overline{G}$
2. Suppose that $\overline{G}$ has a vertex cover $V' \subseteq V$ where $|V'| = |V| - k$
   $\forall u, v \in V[(u, v) \in \overline{E} \implies u \in V' \vee v \in V'$ (or both)]
   Equivalently, $\forall u, v \in V[u \notin V' \wedge v \notin V' \implies (u, v) \notin \overline{E}]$
   In other words, $\forall u, v \in V[u \notin V' \wedge v \notin V' \implies (u, v) \in E]$
   Therefore, $V \setminus V'$ is a clique of size $|V| - |V'| = k$

■

5 Subset Sum

Given a finite set S of positive integers and an integer target $t > 0$. We ask whether there exists a subset $S' \subseteq S$ whose elements sum to t. For example:

$$S = \{1, 2, 7, 14, 49, 98\}, t = 57$$
$$S' = \{1, 7, 49\}$$

Theorem 5.1. *SUBSET-SUM is NP-complete.*

Proof:

- SUBSET-SUM $\in$ NP. Given an instance of the problem $< S, t >$ and the certificate S'. We can check in polynomial time whether $t = \sum_{s \in S'} s$
- SUBSET-SUM $\in$ NP-hard. We will show this by 3-SAT $\leq_P$ SUBSET-SUM.

  Given a CNF formula ϕ over variables $x_1, \ldots, x_n$ with clauses $C_1, \ldots, C_k$, each containing exactly 3 distinct literals. WLOG, we make 2 simplifying assumptions about ϕ

  1. No clauses contain both a variable and its negation. Such clauses are automatically satisfied.
  2. Each variable appears in at least one clause.

  Reduction: Create 2 numbers in S for each variable x_i and 2 numbers in S for each clause C_j. We will create numbers in base 10, where each number contains $n + k$ digits and each digit corresponds to either one variable or one clause. The least significant k digits are labeled by the clauses, and the most significant n digits are labeled by variables.

  - The target t has a 1 in each digit labeled by a variable and a 4 in each digit labeled by a clause.
  - $x_i \to \nu_i, \nu_i' \in S$. ν_i, ν_i' has a 1 in the digit labeled by x_i and 0's in other variable digits.

* If x_i in clause C_j, ν_i contains a 1 in the digit labeled for C_j
* If $\neg x_i$ in clause C_j, ν'_i contains a 1 in the digit labeled for C_j

All ν_i, ν'_i values are unique because:

* For any i, $\nu_i \neq \nu'_i$ in the least significant k digits (labeled for clauses) because the literal and its negation cannot be in the same clause by our first simplifying assumption.
* For any $l \neq i$, no ν_l, ν'_l can equal ν_i, ν'_i in the most significant n digits (labeled for variables) by construction.

– $C_j \to s_j, s'_j \in S$. s_j, s'_j has 0's in all digits other than the one labeled for C_j: s_j has a 1 in this digit, and s'_j has a 2 in this digit. These integers are "slack variables" which we will use to get each clause-labeled digit position to add to the target value of 4.

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
ν_1	=	1	0	0	1	0	0	1
ν'_1	=	1	0	0	0	1	1	0
ν_2	=	0	1	0	0	0	0	1
ν'_2	=	0	1	0	1	1	1	0
ν_3	=	0	0	1	0	0	1	1
ν'_3	=	0	0	1	1	1	0	0
s_1	=	0	0	0	1	0	0	0
s'_1	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s'_2	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s'_3	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s'_4	=	0	0	0	0	0	0	2
t	=	1	1	1	4	4	4	4

Figure 3: Reduction of 3SAT clause $\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$ where $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3), C_2 = (\neg x_1 \vee \neg x_2 \vee \neg x_3), C_3 = (\neg x_1 \vee \neg x_2 \vee x_3), C_4 = (x_1 \vee x_2 \vee x_3)$

Figure 3 shows an example of the reduction.

We can perform the reduction in polynomial time.

– The set S contains $2n + 2k$ values, each of which has $n + k$ digits, and the time to produce each digit is polynomial in $n + k$.
– The target t has $n + k$ digits, and the reduction produces each in constant time.

Now, show that 3SAT formula is satisfiable $\iff$ there exists $S' \subseteq S$ whose sum is t.

– ($\Rightarrow$) Suppose ϕ has a satisfying assignment. For $i = 1, \ldots, n$, if $x_i = 1$ in this assignment, include ν_i in S'. Otherwise, include ν'_i.

Having included either ν_i or ν'_i but not both, for all i, and having put 0 in digits labeled for variables in all s_j, s'_j, the sum of the values in the variable-labeled digits must be 1, which matches those digits of the target t.

Because each clause is satisfied, the clause contains some literals with the value 1. Therefore, each digit labeled for a clause has at least one 1 contributed to its sum by ν_i or ν'_i value in S.

In fact, 1,2,3 literals may be 1 in each clause $\rightarrow$ the each clause-labeled digit has a sum of either 1,2,3 from ν_i or ν'_i value in S.

We achieve the target of 4 in each digit labeled for clause C_j by including the slack variable s_j, s'_j in S' (including one, two, or both)

– ($\Leftarrow$) Suppose $\exists S' \subseteq S$ that sums to t. The subset S' must include exactly one of ν_i and ν'_i for each $i = 1, \ldots, n$. Otherwise, the digits for each variable would not sum to 1.

If $\nu_i \in S'$, set $x_i = 1$, otherwise $\nu'_i \in S'$ and set $x_i = 0$.

We claim that every clause C_j for $j = 1, \ldots, k$ is satisfied by this assignment.

* To achieve the sum of 4 in the digit labeled by C_j, S' must include at least one ν_i or ν'_i value that has a 1 in the digit labeled for C_j since the contribution of the slack variables is at most 3.
  - If S' includes a ν_i that has a 1 in C_j's position, then x_i appears in C_j and we set x_i to 1 earlier.
  - If S' includes a ν'_i that has a 1 in C_j's position, the literal $\neg x_i$ appears in C_j. Because we set $x_i = 0$ when $\nu'_i \in S'$, clause C_j is again satisfied.
* All clauses of ϕ are satisfied.

■

6 Hamiltonian Cycle

Given a directed graph $G = (V, E)$, is there a *Hamiltonian cycle* that visits each vertex only once and returns back to the starting vertex?

Theorem 6.1. *HAM-CYCLE is NP-Complete*

Proof:

- HAM-CYCLE is NP. Given G and a ordered list of vertices $v'_1, \ldots, v'_n, v'_1$ we can check in polynomial time if the given certificate is a Hamiltonian cycle in G.
- We will show that 3SAT $\leq_P$ HAM-CYCLE. For ϕ with n variables and k clauses, there are 2^n possible truth assignments.

  We construct n paths, $P_1, \ldots, P_n$, where P_i consists of nodes $v_{i1}, v_{i2}, \ldots, v_{ib}$. We take the quantity b to be somewhat larger than k, say $b = 3k + 3$. There are edges from v_{ij} to $v + i, j + 1$ and in the reverse direction $v_{i,j+1}$ to v_{ij}. Thus, P_i can be traversed in either direction.

  We hook these paths together as follows:

  – For each $i = 1, \ldots, n - 1$, define edges from v_{i1} to $v_{i+1}, 1$ and to $v_{i+1,b}$ and also from v_{ib} to $v_{i+1,1}$ and to $v_{i+1,b}$.

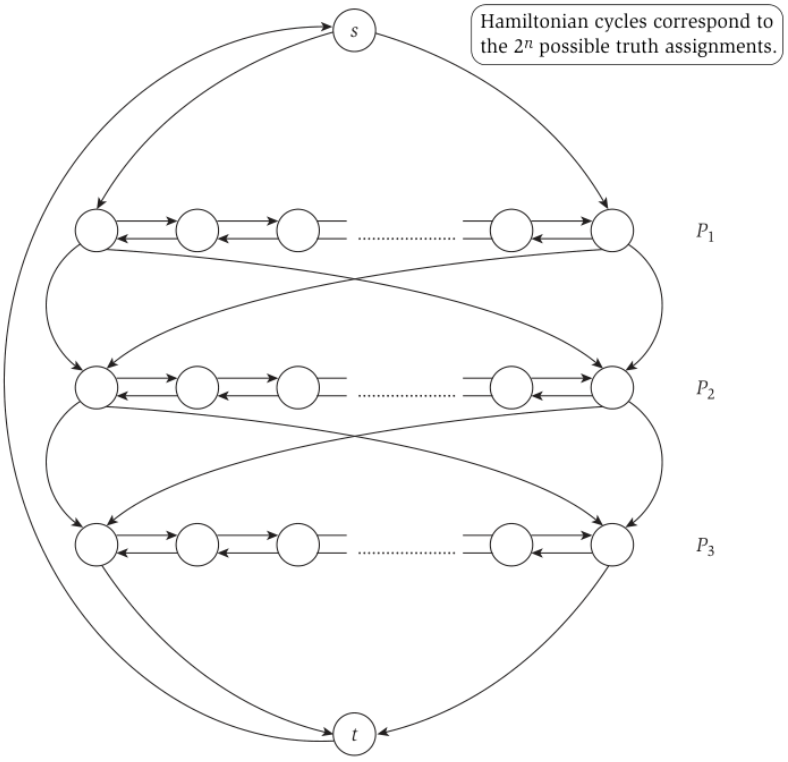

Figure 4: Graph construction from 3SAT (1)

– Add two extra nodes s, t. Define edges $s \to v_{11}, v_{1b}$ and $v_{n1}, v_{nb} \to t$ and $t \to s$

Figure 4 shows the graph construction so far.

Consider the graph that we just constructed, there is only one edge leaving t. Thus, the Hamiltonian cycle must use the edge (t, s). After entering s, the cycle can traverse P_1 either l-to-r or r-to-l. Same with $P_2, \ldots, P_n$ and enters t.

In other words, there are exactly 2^n different Hamiltonian cycles, corresponding to n independent choices of how to traverse P_i.

We will identify each Hamiltonian cycle with a truth assignment as follows:

– If the cycle traverses P_i l-to-r, $x_i = 1$.

– Otherwise, $x_i = 0$

Now, we add the nodes to model the clauses. Let's consider an example, a clause $C_1 = (x_1 \vee \neg x_2 \vee x_3)$ means (in Ham cycle) that "the cycle should traverse P_1 l-to-r or P_2 r-to-l or P_3 l-to-r.

So, we add a node c_1. For some l, add edges $v_{1,l}, v_{2,l+1}, v3, l \to c_1$ and $c_1 \to v_{1,l+1}, v_{2,l}, v_{3,l+1}$ as shown in Figure 5. This new node can be spliced into any Hamiltonian cycle that traverse the direction that we desire, but the node cannot be spliced into the cycle that does not follow our desired directions.

More generally, we will define a node c_j for each clause C_j. We reserve the node positions $3j$ and $3j+1$ in each path P_i for the variables participated in clause C_j. Suppose C_j contains a term $t = x_i$, we add edges $(v_{i,3j}, c_j)$ and $(c_j, v_{j,3j+1})$. If the term $t = \neg x_i$, we add edges $(v_{i,3j+1}, c_j)$ and $(c_j, v_{i,3j})$.

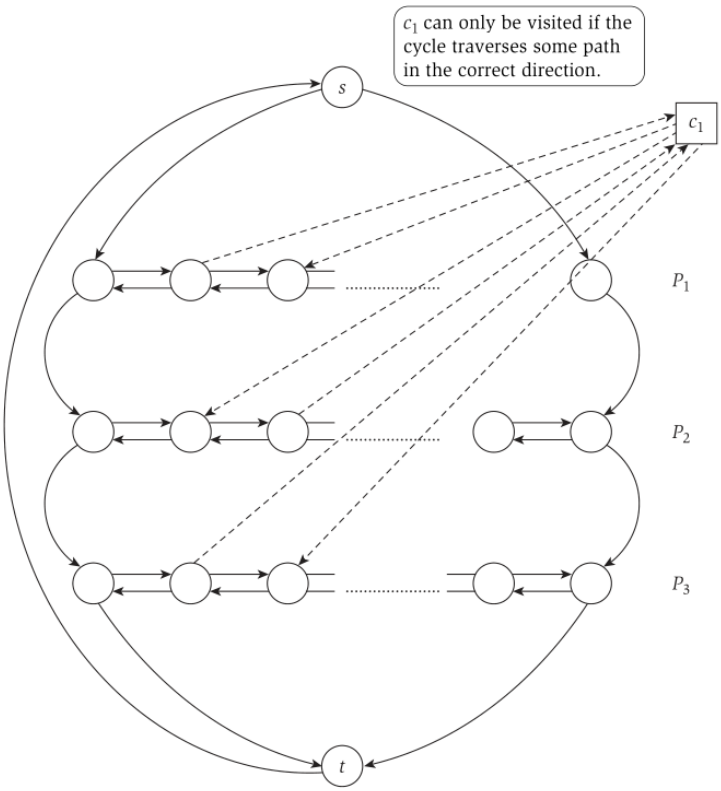

Figure 5: Graph construction from 3SAT (2)

This completes the construction of the graph. Now, we claim that 3SAT instance is satisfiable $\iff$ G has a Hamiltonian cycle.

- ($\Rightarrow$) Suppose there is a satisfying assignment for 3SAT instance. Then, we define a Ham cycle as described by our plan ($x_i = 1$ traverses P_i l-to-r, $x_i = 0$ traverses P_i r-to-l). For each clause C_j, since it is satisfied by the assignment, there will be at least one path P_i in which we will be going in the "correct" direction and visit c_j.
- ($\Leftarrow$) Suppose there is a Hamiltonian cycle H in G. Observe that if H enters a node c_j on an edge from $v_{i,3j}$, it must depart on an edge to $v_{i,3j+1}$. Otherwise, $v_{i,3j+1}$ will have only one unvisited neighbor left ($v_{i,3j+2}$), and the tour will not be able to visit this node and still maintain the Hamiltonian property. Symmetrically if it enters from $v_{i,3j+1}$, it must depart to $v_{i,3j}$.

  Then, we assign $x_i = 1$ if P_i traverses l-to-r and $x_i = 0$ otherwise. Any clause C_j will have at least one literal assign to 1 based on our construction of G.

■

7 Traveling Salesman Problem

Given a weighted, directed graph $G = (V, E)$, is there a tour starting and ending at city v_{i_1} of length at most D?

Theorem 7.1. *TSP is NP-Complete*

Proof:

- TSP is NP. The certificate is a permutation of cities. We can check the length of the corresponding tour $\leq D$
- HAM-CYCLE $\leq_P$ TSP. Given a directed graph $G = (V, E)$, we define the following instance G' of TSP. We have a city v'_i for each node $v_i \in V$. Define $d(v'_i, v'_j) = 1$ if $(v_i, v_j) \in V$ and $d(v'_i, v'_j) = 2$ otherwise.

  We claim that G has a Hamiltonian cycle $\iff$ G' has a tour of length at most n.

  - ($\Rightarrow$) Suppose that G has a Ham cycle. This ordering of the corresponding cities defines a tour of length n in G'.
  - ($\Leftarrow$) Suppose that G' has a tour of length at most n. The expression for the length of this tour is a sum of n terms (n cities), each of which is at least 1. Hence, each pair of nodes in G that correspond to consecutive cities on the tour must be connected by an edge. Thus, the ordering of these corresponding nodes must form a Hamiltonian cycle.

■