Assignment 6
Algorithm Design and Analysis

bitjoy.net

January 2, 2016

I choose problem 1,2,3,4,5,6.

1 Integer Programming

We first show that Integer Programming is in **NP**. Obviously, given an n-vector x, $Ax \geq b$ can be verified in polynomial time, so it is in **NP**.

We then prove that $3SAT \leq_p ILP$. Given a $3SAT$ instance like this:

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_4) \tag{1}$$

We will have corresponding variables y_1, y_2, y_3, y_4 in our ILP, if $x_i = true$, then $y_i = 1$ otherwise $y_i = 0$.

$$\begin{array}{lrcll} \max & 0 & & & \\ s.t. & y_1 + y_2 + y_3 & \geq & 1 & \\ & y_1 + (1 - y_2) + y_4 & \geq & 1 & \\ & y_i & \in & \{0,1\} & i = 1, ..., 4 \end{array} \tag{2}$$

If (1) can be satisfied, the corresponding y_is are the optimal solution to (2), that's to say (2) has optimal solution. If (2) has optimal solution, the corresponding x_is make (1) be $true$. If (1) can't be satisfied, (2) has no optimal solution, and vice versa. So, $3SAT \leq_p ILP$.

Integer-programming is in **NP**-complete.

2 Mine-sweeper

Here is the MINESWEEPER language:

MINESWEEPER: $\{G, \xi \mid G$ is a graph and ξ is a partial integer labeling of G, and G can be filled with mines in such a way that any node v labeled m has exactly m neighboring nodes containing mines.$\}$

Deciding if a graph is in the MINESWEEPER language is **NP**-complete.

First, given a graph with nodes labeled with integers or containing mines, we can verify it in polynomial time. Just check whether each node labeled with m has exactly m neighboring nodes containing mines.

Second, we prove $3SAT \leq_p MINESWEEPER$. Suppose we have one $3SAT$ clause

$$(x_1 \vee \neg x_2 \vee x_3) \tag{3}$$

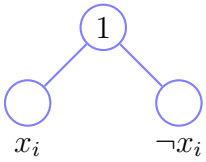


Figure 1: Gadget for each variable.

For each variable, we construct a gadget like Figure 1.

If $x_i = true$, then x_i is filled with mine, otherwise $\neg x_i$ is filled with mine. The top node labeled with '1' forces that only one of x_i and $\neg x_i$ can be $true$.

For clause (3), we have:

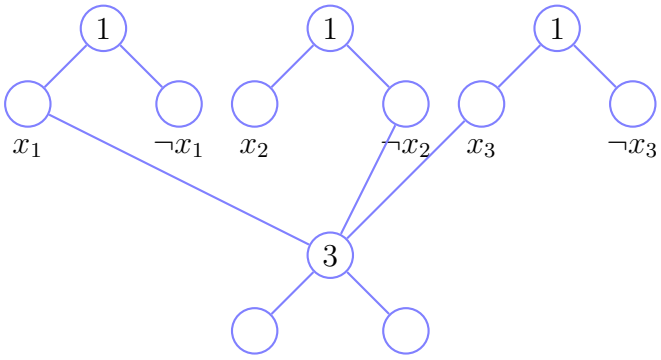


Figure 2: Gadget for each clause.

For each $true$ assignment of this clause, we can find a placement of mines makes the graph consistent. Figure 3 is one of $true$ assignment examples.

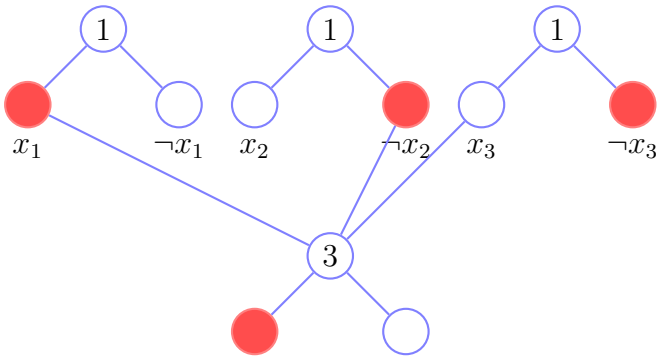


Figure 3: $\{x_1 = true, x_2 = false, x_3 = false\}$ is one of true assignment for clause (3), we find a placement of mines makes the graph consistent.

For the $false$ assignment of this clause, we can't find any placement of mines makes the graph consistent. Figure 4 is the $false$ assignment.

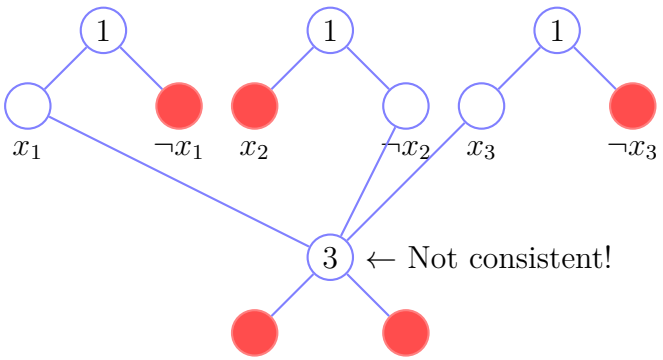


Figure 4: $\{x_1 = false, x_2 = true, x_3 = false\}$ is the false assignment for clause (3), we can't find any placement of mines makes the graph consistent.

So, if $3SAT$ can be satisfied, graph G is in the MINESWEEPER language, otherwise not.

Mine-sweeper is in **NP**-complete.

3 Half-3SAT

First, Half-3SAT is in **NP**. Given a particular assignment of n variables, we can check if m clauses satisfy Half-3SAT conditions in polynomial time. Just check each clause, if half are $true$, half are $false$, it's Half-3SAT, otherwise not.

Second, we prove $3SAT \leq_p Half-3SAT$. Given a instance of $3SAT$ with m clauses, we construct a $Half-3SAT$ instance with $4m$ clauses like this. First m clauses are exactly the same as $3SAT$. Next, we create m clauses of the form

$$(p \vee \neg p \vee q) \tag{4}$$

which is always $true$. Next, we create $2m$ of clauses of the form

$$(p \vee q \vee r) \tag{5}$$

These $2m$ clauses are always $true$ or always $false$.

If $3SAT$ is satisfied, we set last $2m$ clauses be $false$. So there are $2m$ $true$ clauses and $2m$ $false$ clauses. Thus, $Half-3SAT$ is satisfied.

If $Half-3SAT$ is satisfied, as there are m (4) clauses, which is always $true$, so the last $2m$ (5) clauses can only be $false$. Thus, the first m clauses are $true$, $3SAT$ is satisfied.

Half-3SAT is in **NP**-complete.

4 Solitaire Game

First, Solitaire Game is in **NP**. Given a final board status, we can determine whether it satisfies the winning condition in polynomial time. Just check whether each column contains only stones of a single color and each row contains at least one stone. For $n \times n$ board, it takes $O(2n)$, so Solitaire Game is in **NP**.

Second, we prove $3SAT \leq_p SOLITAIRE$. Given a instance of $3SAT$ with 2 clauses and 4 variables:

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_4) \tag{6}$$

	x_1	x_2	x_3	x_4
c_1	▲	▲	▲	
c_2	▲	■		▲
×	×	×	×	×
×	×	×	×	×

Table 1: Initial board for (6), ▲for x_i and ■for $\neg x_i$, one row for one clause.

For each variables, ▲for x_i and ■for $\neg x_i$. So we get Table 1 for (6).

For $(y_1 \vee y_2 \vee y_3)$, y_i can be x_j or $\neg x_j$. For each clause, if $y_i = false$, we remove the corresponding stone. If (6) can be satisfied, then Table 1 is a winnable game configuration. Table 2 is one of $true$ examples.

	x_1	x_2	x_3	x_4
c_1	▲	▲	▲	
c_2	▲			▲
×	×	×	×	×
×	×	×	×	×

Table 2: $\{x_1 = true, x_2 = true, x_3 = true, x_4 = true\}$ is the true assignment for clause (6), $\neg x_2 = false$, so removing the red stone in c_2, we win the game.

For any $false$ assignment of (6), Table 1 is not a winnable game configuration. Table 3 is one of $false$ examples.

	x_1	x_2	x_3	x_4
c_1		▲	▲	
c_2				
×	×	×	×	×
×	×	×	×	×

Table 3: $\{x_1 = false, x_2 = true, x_3 = true, x_4 = false\}$ is the false assignment for clause (6), similarily, removing $< c_1, x_1 >, < c_2, x_1 >, < c_2, x_2 >, < c_2, x_4 >$, we lose the game.

So, if $3SAT$ can be satisfied, G is a winnable game configuration, otherwise not.

Solitaire Game is in **NP**-complete.

5 Directed Disjoint Paths Problem

First, Directed Disjoint Paths Problem is in **NP**. Given k paths $P_1, P_2, ..., P_k$, we can determine whether they are Directed Disjoint Paths in polynomial time. Just go through every path and record the nodes, if more than one path go through a node, they are not Disjoint Paths.

Second, we prove $3SAT \leq_p$ *Directed Disjoint Paths Problem*. Given an arbitrary $3SAT$ formula, we construct a network routing problem like this: For each clause C_i, we create a source/sink pair (s_i, t_i), and a node $x_j^i(\neg x_j^i)$ for each literal $x_j(\neg x_j)$ in the clause. Add edges $< s_i, x_j^i >$ and edges $< x_j^i, t_i >$. In addition, we create source/sink pair (s_j, t_j) for each variable x_j, we get a total of $k = q + n$ source/sink pairs, where q is the number of clauses and n is the number of literals.

For example, given a $3SAT$ instance with two clauses and three variables:

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \tag{7}$$

We construct a graph G like Figure 5.

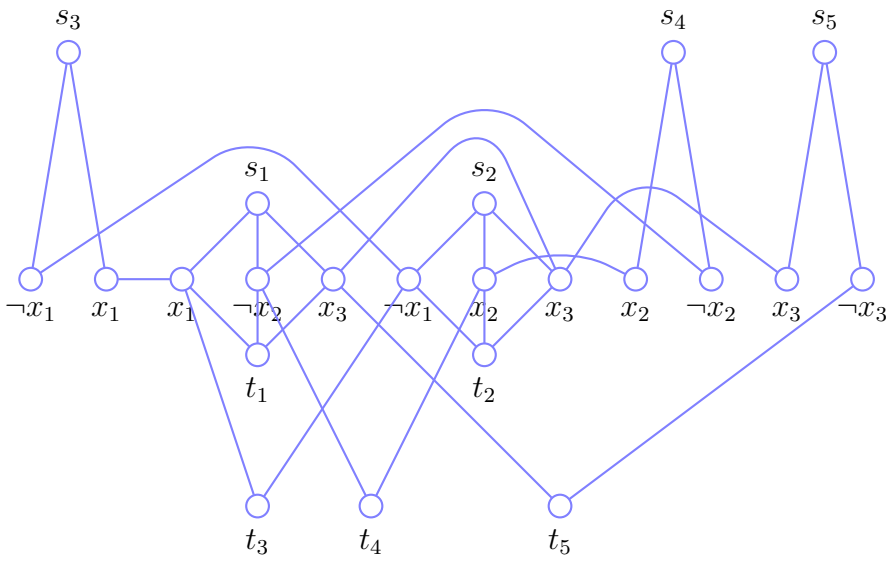


Figure 5: Graph G for $3SAT$ formula (7).

We now show that if $3SAT$ can be satisfied, there are node-disjoint paths for k pairs of nodes (s_i, t_i).

Given a *true* assignment of $3SAT$, we construct the k paths as follows. For each (s_j, t_j) pair that representing a variable x_j, we choose the path that corresponds to the literal that is not set to true in the satisfying assignment. For example, if $x_j = true$, we choose path (s_j, t_j) that goes through $\neg x_j$, if $x_j = false$, we choose path (s_j, t_j) that goes through x_j.

$\{x_1 = true, x_2 = true, x_3 = false\}$ is a *true* assignment of formula (7), according to rules above, we get Figure 6, which has $k = 5$ node-disjoint paths.

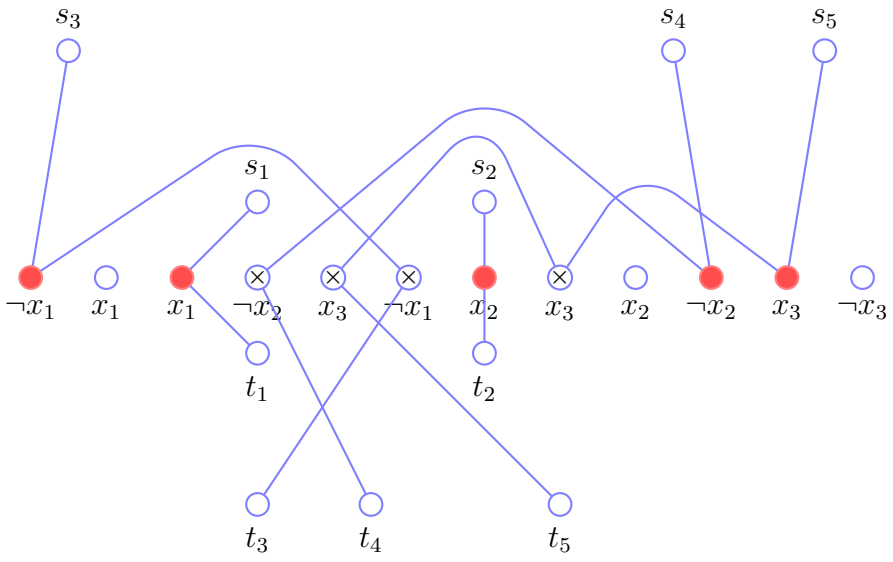


Figure 6: When $\{x_1 = true, x_2 = true, x_3 = false\}$, we find $k = 5$ node-disjoint paths.

$\{x_1 = false, x_2 = true, x_3 = false\}$ is a $false$ assignment of formula (7), according to rules above, we can't find $k = 5$ node-disjoint paths. Figure 7 shows this case.

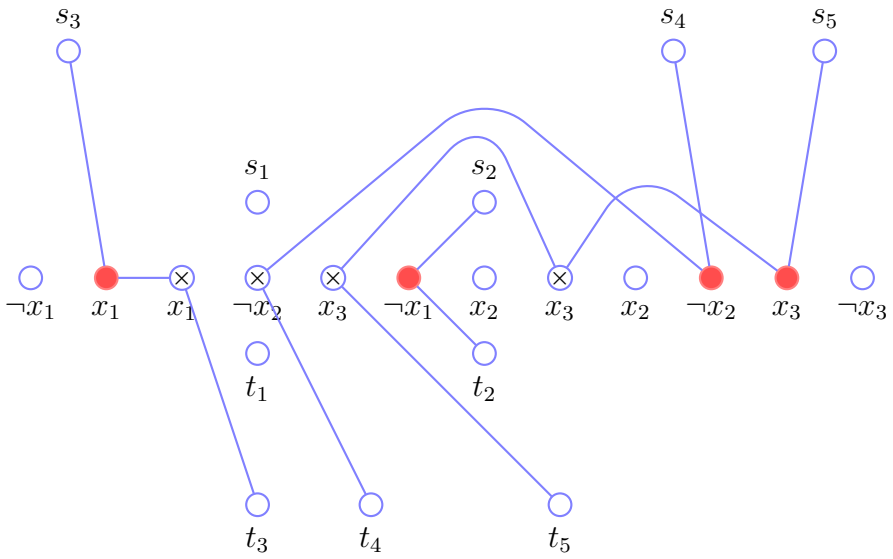


Figure 7: When $\{x_1 = false, x_2 = true, x_3 = false\}$, s_1 can't reach t_1, failed.

Similarly, if we already have k node-disjoint paths in graph G, we just do the inverse process, the $3SAT$ can be satisfied.

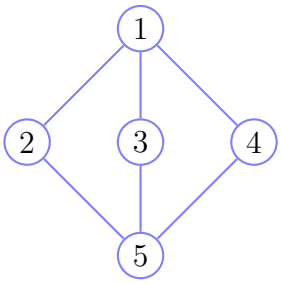
Directed Disjoint Paths Problem is in **NP**-complete.

6 Longest Common Subsequence Problem

First, Longest Common Subsequence is in **NP**. Given an integer k and a set $R = \{S_1, S_2, ..., S_p\}$ of sequences, whose $length(S_i) = n$. We can verify if $|LCS(R)| \geq k$ in polynomial time. Just check whether each subsequence s of S_1 is also the subsequence of $S_2, ..., S_p$, it takes $O(C_n^k * n * (p-1))$.

Second, we prove $Vertex\ Cover \leq_p LCS$. Given $G =< E, V >$ where $V = \{1, 2, ..., n\}$ and $E = e_{ij},\ i, j \in V$. For each edge e_{ij}, we construct a sequence $S_{ij} = 1, 2, ..., i-1, i+1, ...n, 1, 2, ..., j-1, j+1, ...n$, $R = \{S_{ij} | e_{ij} \in E\}$. If there are k vertexes can cover all e_{ij}, then $|LCS(R)| \geq n-k$ holds. What's more, $V-T$ can be the $LCS(R)$ where T is the vertexes that can cover all edges in G.

Given a graph G like Figure 8, we can list all the S_{ij} on the right side.



$$
\begin{aligned}
V &= 1,2,3,4,5 \\
S_{12} &= 2,3,4,5,1,3,4,5 \\
S_{13} &= 2,3,4,5,1,2,4,5 \\
S_{14} &= 2,3,4,5,1,2,3,5 \\
S_{25} &= 1,3,4,5,1,2,3,4 \\
S_{35} &= 1,2,4,5,1,2,3,4 \\
S_{45} &= 1,2,3,5,1,2,3,4
\end{aligned}
$$

Figure 8: Graph G for vertex cover.

There are three vertexes that can cover all the edges, i.e. $k = 3$ and $T = \{2, 3, 4\}$, so $|LCS(R)| \geq n - k = 2$ holds. What's more, $LCS = \{1, 5\}$.

If vertexes cover is $T = \{2, 3, 4\}$, we prove $LCS = \{1, 5\}$. As vertexes cover is $T = \{2, 3, 4\}$, for $\forall e_{ij}$, either $i \in T$ or $j \in T$. If $i \in T$, set $S_{ij} = 1, 2, ..., i-1, i+1, ...n$; else if $j \in T$, set $S_{ij} = 1, 2, ..., j-1, j+1, ...n$. So $LCS(R) = V - T$. For example, for e_{12}, $2 \in T$, we set $S_{12} = 1, 3, 4, 5$, so $2 \notin LCS$. Finally $LCS = \{1, 5\}$.

If $LCS = \{1, 5\}$, we prove $T = \{2, 3, 4\}$ is vertexes cover. If there is one edge that T can't cover, say e_{ij}. so $i \notin T$ and $j \notin T$. $S_{ij} = 1, 2, ..., i-1, i+1, ...n, 1, 2, ..., j-1, j+1, ...n$, $LCS(S_{ij}, V) \supsetneqq LCS(R)$, contradiction! For example, suppose edge e_{15} exists, e_{15} can't be covered by T, $S_{15} = 2, 3, 4, 5, 1, 2, 3, 4$, $V = 1, 2, 3, 4, 5$. $LCS(S_{15}, V) \supsetneqq LCS(R) = \{1, 5\}$, so e_{15} doesn't exist, T is vertexes cover.

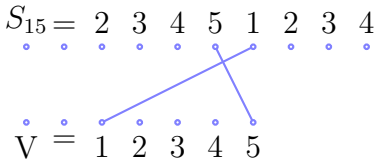


Figure 9: There is one intersection, so $LCS(S_{15}, V) \supsetneqq LCS(R) = \{1, 5\}$, e_{15} doesn't exist, $T = \{2, 3, 4\}$ is vertexes cover.

So, the *yes/no* **LCS** problem is in **NP**-complete.