Assignment 12 – More Intractability

Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: _____ Wisc id: _____

1. *Kleinberg, Jon. Algorithm Design (p. 512, q. 14)* We've seen the Interval Scheduling Problem several times now, in different variations. Here we'll consider a computationally much harder version we'll call *Multiple Interval Scheduling*. As before, you have a processor that is available to run jobs over some period of time.

   People submit jobs to run on the processor. The processor can only work on one job at any single point in time. Jobs in this model, however, are more complicated than we've seen before. Each job requires a *set* of intervals of time during which it needs to use the processor. For example, a single job could require the processor from 10am to 11am and again from 2pm to 3pm. If you accept the job, it ties up your processor during those two hours, but you could still accept jobs that need time between 11am and 2pm.

   You are given a set of n jobs, each specified by a set of time intervals. For a given number k, is it possible to accept at least k of the jobs so that no two accepted jobs overlap in time?

   Show that Multiple Interval Scheduling is NP-Complete.

Solution:

1. *Multiple Interval Scheduling is in NP.*

   Given a set of jobs $J_1, \ldots, J_k$, for each job J_i, iterate through the intervals in the job and flag the processor as occupied during each. If you try to flag the processor during an interval when it is already flagged, then the solution is invalid. If you iterate through all k jobs without trying to flag the same time twice, then the certificate is valid. This process takes at most time equal to the number of available time intervals, since at worst we flag each interval once.

2. *MIS is NP-Hard. (Independent Set $\leq_p$ MIS)*

   An arbitrary instance of Independent Set is defined by a graph G.

   - For each edge e_j in G, create a time interval t_j.
   - For each node v_i in G, create a job J_i which uses the intervals t_j which correspond to edges adjacent to v_i.

   ($\Rightarrow$) If there is a size k independent set, then we can select k nodes from G such that no edge is adjacent to more than one node in our set. Let J_i be in our candidate set of non-conflicting jobs if and only if v_i is in this independent set. By construction, each time interval t_j corresponds to an edge in G, so only one of its adjacent nodes in G could have been picked, and only the jobs corresponding to those nodes require t_j. Since we only picked one of those, our candidate set of jobs is non-conflicting.

   ($\Leftarrow$) If there is a size k set of non-conflicting jobs, then by the same logic as above the set of nodes $\{v_i\}$ corresponding to the jobs $\{J_i\}$ constitute an independent set in G.

   We have a polynomial time mapping from Independent Set to Multiple Interval Scheduling in which we have a yes instance of IS if and only if we have a yes instance of MIS. Since Independent Set is NP-complete, Multiple Interval Scheduling is NP-Hard.

2. *Kleinberg, Jon. Algorithm Design (p. 519, q. 28)* Consider this version of the Independent Set Problem. You are given an undirected graph G and an integer k. We will call a set of nodes I "strongly independent" if, for any two nodes $v, u \in I$, the edge (v, u) is not present in G, and neither is there a path of two edges from u to v, that is, there is no node w such that both (v, w) and (u, w) are present in G. The Strongly Independent Set problem is to decide whether G has a strongly independent set of size at least k.

   Show that the Strongly Independent Set Problem is NP-Complete.

Solution:

1. *Strongly Independent Set is in NP.*

   Given a set of nodes I, we can check whether any nodes $v, u \in I$ are too close together in polynomial time. For each $u \in I$, we can use BFS to see if any of the other nodes are distance 1 or 2 edges away. This requires $O(nm)$ time in total.

2. *SIS is NP-Hard. (Independent Set $\leq_p$ SIS)*

   An arbitrary instance of Independent Set is defined by a graph G. We will construct a graph G' as an instance of Strongly Independent Set.

   - For each node v_i in G, add v_i to G'.
   - For each edge $e_j = (u, v)$ in G, add a node w_j to G' and add edges (u, w_j) and (w_j, v).
   - Finally add an edge between each pair of added w nodes.

   ($\Rightarrow$) If there is a size k independent set, then the same set of nodes is a size k strongly independent set in G'. Because we have subdivided each edge in G into two edges in G', if two nodes were not adjacent in G (and both had at least one adjacent edge), the distance between them is now 3 ($u - w_i - w_j - v$).

   ($\Leftarrow$) If there is a size k strongly independent set in G', that doesn't use any of the added w nodes, then it follows by construction that they constitute an independent set in G. It remains to show that no nodes w can be part of a strongly independent set. Without loss of generality, consider a particular such node w^*. Because w^* is adjacent to all other w nodes, it can only be part of a strongly independent set if no other w node is, and additionally no other node adjacent to a w node is. In other words, w^* can only be part of a strongly independent set if it is the only node in the set (excluding nodes with no incident edges at all). Note that if there is a strongly independent set which includes w^*, there must exist a second one that includes no w nodes by replacing w^* with any non-w node that has at least one incident edge.

   We have a polynomial time mapping from Independent Set to Strongly Independent Set in which we have a yes instance of IS if and only if we have a yes instance of SIS. Since Independent Set is NP-complete, Strongly Independent Set is NP-Hard.

3. *Kleinberg, Jon. Algorithm Design (p. 527, q. 39)* The *Directed Disjoint Paths Problem* is defined as follows: We are given a directed graph G and k pairs of nodes $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$. The problem is to decide whether there exist node-disjoint paths $P_1, \ldots, P_k$ so that P_i goes from s_i to t_i.

   Show that Directed Disjoint Paths is NP-Complete.

Solution:

1. *Directed Disjoint Paths is in NP.*

   Given a set of paths $P_1, \ldots, P_k$, for each path P_i, iterate through the path and flag each node. If you encounter a node that is already flagged, then the solution is invalid. If you iterate through all k paths without trying to flag the same node twice, then the certificate is valid. This process takes at most time $O(|V|)$, since at worst we flag each node in G once.

2. *DDP is NP-Hard. (3-SAT $\leq_p$ DDP)*

   Suppose we are given a logical formula ϕ with variables $x_1, \ldots x_m$ and clauses $c_1, \ldots c_k$. We construct a graph G in the following way. For each variable x_i, we make two paths:

   - $s_i^v \to T_{i,1} \to T_{i,2} \to \ldots \to T_{i,k} \to t_i^v$, call it T path.
   - $s_i^v \to F_{i,1} \to F_{i,2} \to \ldots \to F_{i,k} \to t_i^v$, call it F path.

   For each clause c_j, we make three paths for each literal occuring inside c_j:

   - If $x_i \in c_j$, then $s_j^c \to T_{i,j} \to t_j^c$
   - If $\overline{x_i} \in c_j$, then $s_j^c \to F_{i,j} \to t_j^c$

   This creates an instance of DDP with the graph G and $m + k$ pair of nodes (s_i^v, t_i^v) and (s_j^c, t_j^c) for $1 \leq i \leq m$ and $1 \leq j \leq k$. ($\Rightarrow$) Suppose ϕ is satisfiable. If x_i is assigned true, then we choose the F path between (s_i^v, t_i^v). If x_i is assigned false, then we choose the T path between (s_i^v, t_i^v). For each clause c_j, there is a literal $\ell_i \in \{x_i, \overline{x_i}\}$ that is assigned to true since ϕ is satisfiable. If $x_i \in c_j$ is assigned to true, then we can choose $s_j^c \to T_{i,j} \to t_j^c$. If $\overline{x_i} \in c_j$ is assigned to true, then we can choose $s_j^c \to F_{i,j} \to t_j^c$. Note that all paths are disjoint. ($\Leftarrow$). Suppose there are disjoint paths between all (s, t) pairs. We interpret the path $s_j^c \to T_{i,j} \to t_j^c$ as the clause c_j satisfied by assigning true to x_i. Similarly, the path $s_j^c \to F_{i,j} \to t_j^c$ as the clause c_j satisfied by assigning false to x_i. Note that there is no conflict in the interpretation, since if there were two distinct clauses c_j and $c_{j'}$ that chose $T_{i,j}$ and $F_{i,j'}$, there cannot be a path between s_i^v and t_i^v.

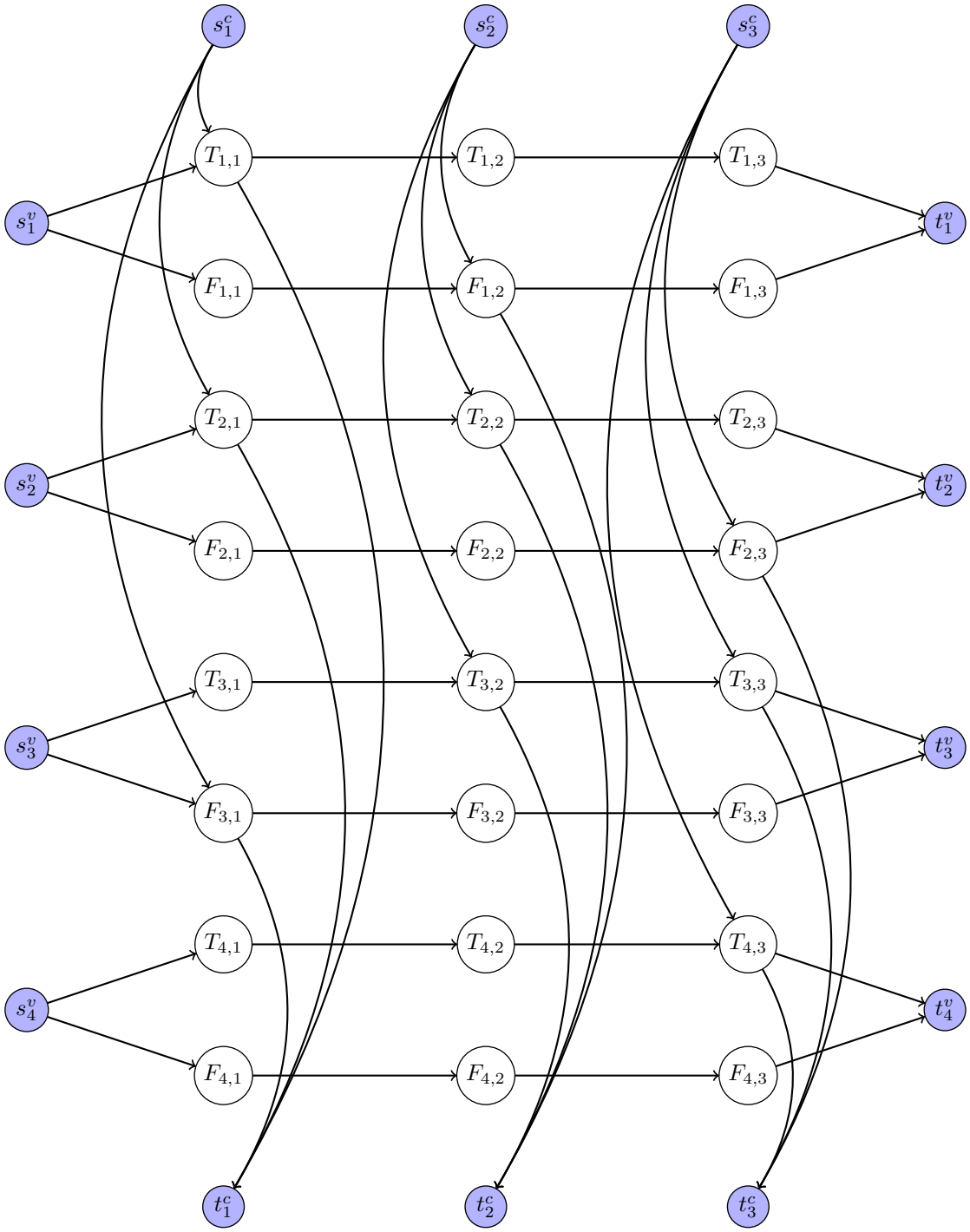
   We have a polynomial time mapping from 3-SAT to DDP in which we have a yes instance of 3-SAT if and only if we have a yes instance of DDP. Since 3-SAT is NP-complete, DDP is NP-Hard.

An example of the reduction is given in the next page.

Solution: Suppose we are given the following 3-SAT formula ϕ.

$$\phi(x_1, x_2, x_3, x_4) = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_3 \vee x_4)$$

We will construct the following graph as an input to DDP.



Solution: Alternate reduction: (proof omitted)

We will reduce Independent Set to DDP. Let $(G = (V, E), k)$ be an instance of independent set. Create k pairs $(s_1, t_1), \ldots (s_k, t_k)$. For each vertex $v \in V$, create nodes a_v and b_v, and edges (s_i, a_v) and (b_v, t_j) for all $i, j = 1, \ldots, k$. For each edge e, create a node x_e. Finally, for each vertex v with incident edge set $I(v) \subset E$, choose some order $e_1, \ldots, e_{|I(v)|}$ of $I(v)$ and create directed edges

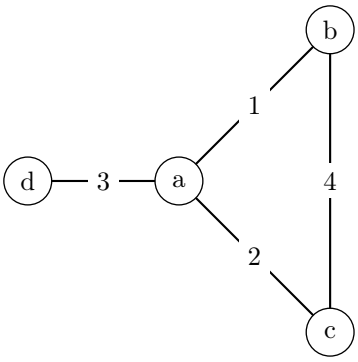
$$(a_v, x_{e_1}), (x_{e_1}, x_{e_2}), (x_{e_2}, x_{e_3}), \ldots, (x_{e_{|I(v)|-1}}, x_{e_{|I(v)|}}), (x_{e_{|I(v)|}}, b_v).$$

The idea is that choosing a path $(s_i, a_v, x_{e_1}, x_{e_2}, \ldots, x_{e_{|I(v)|-1}}, x_{e_{|I(v)|}}, b_v, t_i)$ corresponds to adding vertex v to the independent set; we cannot also add any vertex u adjacent to v because then the paths will intersect at the node x_e corresponding to the edge e between u and v.

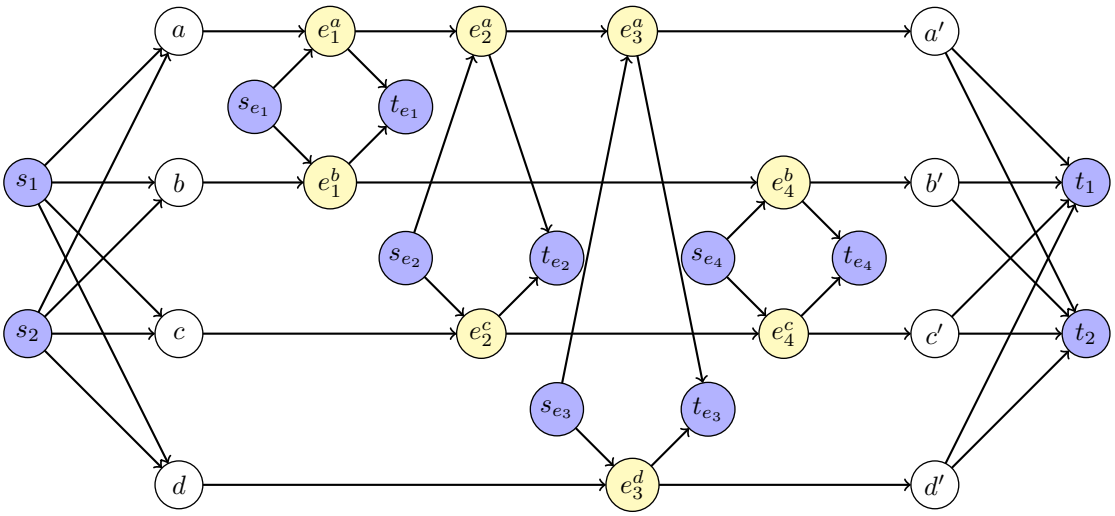
However, this reduction isn't quite correct because there's no guarantee that a path containing a_v will continue on to b_v – it could take the wrong edge at some x_e node. To fix this, replace each node x_{uv} for edge uv with two nodes x^u_{uv} and x^v_{uv}, where the edges along the u path go in and out of x^u_{uv} and the edges along the v path go in and out of x^v_{uv}. Then add a new source/target pair (s_{uv}, t_{uv}) and edges $(s_{uv}, x^u_{uv}), (s_{uv}, x^v_{uv})$ and $(x^u_{uv}, t_{uv}), (x^v_{uv}, t_{uv})$ (still forcing the vertex paths to use at most one of the nodes x^u_{uv} and x^v_{uv}).

An example of the reduction is given below.

Suppose we are given the following graph G with $k = 2$ as an input to Independent Set:



We will construct the following graph as an input to DDP.



4. *Kleinberg, Jon. Algorithm Design (p. 508, q. 9)* The *Path Selection Problem* may look initially similar to the *Directed Disjoint Paths Problem*, but pay attention to the differences between them! Consider the following situation: You are managing a communications network, modeled by a directed graph G. There are c users who are interested in making use of this network. User i issues a "request" to reserve a specific path P_i in G on which to transmit data.

   You are interested in accepting as many path requests as possible, but if you accept both P_i and P_j, the two paths cannot share any nodes.

   Thus, the Path Selection Problem asks, given a graph G and a set of requested paths $P_1, \ldots, P_c$ (each of which must be a path in G), and given a number k, is it possible to select at least k of the paths such that no two paths selected share any nodes?

   Show that Path Selection is also NP-Complete.

Solution:

1. *Path Selection is in NP.*

   Given a set of paths $P_1, \ldots, P_k$, for each path P_i, iterate through the path and flag each node. If you encounter a node that is already flagged, then the solution is invalid. If you iterate through all k paths without trying to flag the same node twice, then the certificate is valid. This process takes at most time $O(|V|)$, since at worst we flag each node in G once.

2. *PS is NP-Hard. (Independent Set $\leq_p$ PS)*

   An arbitrary instance of Independent Set is defined by a graph G. We will create a graph G' on which to run Path Selection.

   - For each edge in G, create a node in G'.
   - For each node v_i in G, create a path P_i that visits each node in G' which corresponds to an edge in G that is incident to v_i. In other words: let $e_{j_1}, e_{j_2}, \ldots, e_{j_{h(i)}}$ be the edges incident to a node v_i in G, then we create a path P_i with edges $e_{j_1} \to e_{j_2} \to \ldots e_{j_{h(i)}}$ (order doesn't matter). (see the figure on next page for an illustration).

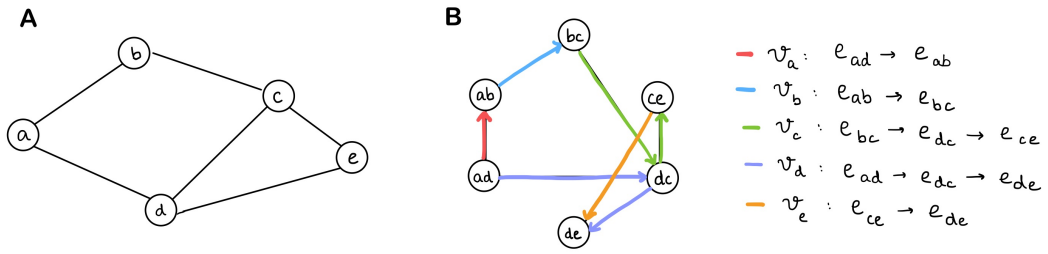
   G has an independent set of size k if and only if there are k paths that are node-disjoint in G'.

   ($\Rightarrow$) If there is a size k independent set, then we can select k nodes from G such that no edge is adjacent to more than one node in our set. Let P_i be in our candidate set of disjoint paths if and only if v_i is in this independent set. By construction, each node e_j in G' corresponds to an edge in G, so only one of its adjacent nodes in G could have been picked, and only the paths corresponding to those nodes include e_j in G'. Since we only picked one of those, our candidate set of paths is indeed disjoint.

   ($\Leftarrow$) If there is a size k set of node-disjoint paths, then by the same logic as above the set of nodes $\{v_i\}$ corresponding to the disjoint paths $\{P_i\}$ constitute an independent set in G.

We have a polynomial time mapping from Independent Set to Path Selection in which we have a yes instance of IS if and only if we have a yes instance of PS. Since Independent Set is NP-complete, Path Selection is NP-Hard.

Solution:



Path Selection Problem. **A** Example directed graph $G = (V, E)$. Observe that the maximum independent set for G has size 2 ($\{a, e\}, \{b, d\}, \{b, e\}$, or $\{a, c\}$). **B** We build a graph G'. For each edge in G, we create a node in G'. Then for every vertex v_i of G with adjacent edges $e_{j_1}, e_{j_2}, \ldots, e_{j_h(i)}$, we create a path P_i with edges $e_{j_1} \to e_{j_2} \to \ldots e_{j_h(i)}$ picked in some arbitrary order. Observe that G' has a maximum of 2 node-disjoint paths: (P_a:red, P_e:orange), (P_b:blue, P_d:purple), (P_b:blue, P_e:orange), and (P_a:red, P_c:green).