Problem Set 6

Bayard Walsh

February 2024

1

INDEPENDENT-SET for k is $NP-$Complete, because INDEPENDENT-SET is in NP and we have a polynomial time reduction from the k clique problem, which is $NP-$HARD.

INDEPENDENT-SET $\in NP$

Witness
To show INDEPENDENT-SET $\in NP$, we have the witness x is a subset of nodes in G. Given x, we can verify if those nodes are an independent set for G in polynomial time by checking every edge in G and seeing if any two nodes $\in x$ have an edge between them, which will run in $|E|$ time. Next we check if $|x| \geq k$. If both are true we have that G contains a valid INDEPENDENT-SET of size k. Therefore if we have this witness x we can verify its correctness in polynomial time, meaning INDEPENDENT-SET $\in NP$.

INDEPENDENT-SET $\in NP - HARD$

We will do a reduction from k CLIQUE problem. Given some undirected graph G and integer k

$f(\langle G, k\rangle) =$
let G' be the complement graph of G *(this can be computed by copying the nodes from G to G', adding every possible edge between all nodes to G', then removing every edge $\in G$ from G')*

YES to YES
Assume there is a witness x showing $G \in k$-CLIQUE. Using x, we can construct a schedule y showing that $G' \in$ INDEPENDENT-SET. A clique is a set of nodes in a graph where every node in the clique is connected to each other. An independent set is a set of nodes where every node in the set is not connected to each other. Therefore if every potential edge not in G is added to G', and every edge in G is removed from G' (G' is the complement of G), then the set of nodes that are a clique for G, say x must be an independent set for G'. Consider any

two nodes in x; in G there must be an edge between every $u, v \in x$, by definition of a k-clique. Therefore in G', all of these edges will be removed as G' is the compliment of G. Note also that no new edges between any $u, v \in x$ can be added when converting from G to G' because every possible edge between any two nodes in x, u, v must already exist for x to be a clique. Our reduction will convert G its complement, G', through adding every potential edge and then removing every edge from G' that's $\in G$. Therefore if we can solve k-clique for G with witness x we can solve INDEPENDENT-SET k for G', as clique set x for G will be an independent set for G'.

NO to NO
We will solve the contra positive; given an independent set y for G' where $|y| \geq k$, we can find a clique x for G where $|x| \geq k$. As shown before, because of the property of switching all edges from a graph to get its compliment and because an independent set cannot have any edges between any two nodes in the set, the independent set of a graph will be a clique for its compliment. An edge will be added between every two nodes in y, making the independent set a clique for G. Therefore we can use the same set y as a k-clique for G. Therefore given witness y to solve independent-set for G', we can find witness x to solve k-clique for G, which is the same set of nodes.

Computable
The reduction from Clique to INDEPENDENT-SET involves querying through every potential edge $\in G$ and switching it from G to G', so all potential edges between every vertex pair will be switched once which is polynomial relative to the size of G.

2

DNF-SAT $\in P$, because the following algorithm solves DNF-SAT in polynomial time.

Algorithm:
Given $\phi = \{t_1, t_2, \cdots t_n\}$, where each t_i is some term
For every $t_i \in \phi$
for every $x \in t_i$ *(for every variable in the term)*

if $x = x_i$ and variable x_i is already set to false or if $x = \bar{x}_i$ and variable x_i is already set to true, reset all variables and continue to t_{i+1}

set variable x_i equal to true if $x = x_i$ and false if $x = \bar{x}_i$

if every $x \in t_i$ is checked and no contradiction is found, return **True**

if we check all $t_i \in \phi$ and there is a contradiction in every term return **False**

Yes to Yes
If every term in ϕ evaluates to False, then the OR of every term is False, and if at least one term in ϕ evaluates to True then the OR of every term is True.
Assume that some valid assignment of variables exists such that ϕ is satisfiable by a DNF formula. Therefore at least one of the terms in ϕ must evaluate to True under some assignment of variables, say t_i under assignment $x_1, x_2, \cdots x_n$. If it is possible that t_i evaluates to True then there must be some assignment of variables such that t_i is True, and our algorithm will hard code the corresponding True or False variables for every literal in t_i so that t_i evaluates to True. This mean ϕ will also evaluate to True. As our algorithm checks every t_i for a valid assignment of variables, if some valid assignment of variables exists for any t_i such that ϕ is satisfiable by a DNF formula, the algorithm will return True.

No to No
Assume that no valid assignment of variables exists such that ϕ is satisfiable by a DNF formula. Therefore ϕ must evaluate to False for every term and for every variable combination, because if one term could evaluate to True under some set of variables then we would have a satisfiable assignment of variables of ϕ. If we can assign arbitrary values to every variable in a term, then a DNF term will only evaluate to False if there is a contradictory assignment in that term (say $\bar{x}_4 \wedge x_4$, which will be False for any x_4 value). If there is no contradictory assignment in a term we can always hardcode the variables to exactly follow the literal so the term evaluates to True. Therefore if no valid assignment of variables exists such that ϕ is satisfiable by a DNF formula, then every term must have a contradictory assignment. Note that the algorithm will skip any term where a contradictory assignment is found, reset all variables, and check the next term for a valid assignment. Therefore if no valid assignment of variables exists such that ϕ is satisfiable by a DNF formula, the algorithm will skip every term and eventually return False.

Computable
In the algorithm we query through every literal in every term in ϕ and save assignments to variables within a given term (reset them after the term if a contradiction is found), and check if a variable has already been set for a given x_i, so the algorithm is computable.

Runtime
For every every term in ϕ, we will either terminate early if we find a valid assignment for a single term and return True or we will terminate after checking every term and return False. Therefore the runtime is polynomial as we check every literal in every term once in ϕ.

3

3.1 a

Our final coloring will be $k(1) = F, k(2) = T, k(3) = \$, k(4) = F, k(5) = T, k(6) = \$, k(7) = F, k(8) = F, k(9) = F$, which is proved below.

First we have that $k(7) = F, k(8) = F, k(9) = F$. Next, consider nodes $5, 6$. 5 is connected to 7 and 6 is connected to 8. Therefore both $5, 6$ are connected to some F node, meaning that neither can be F for a valid 3-coloring. Note that $5, 6$ are also connected to each other, meaning that $k(5) \neq k(6)$. As there are only 3 possible colors for a 3-coloring, one of $k(5)$,$k(6)$ **must equal** \$ and the other **must equal** T. Therefore let $k(5) = T$, and $k(6) = \$$ *(these assignments could be swapped and the proof wouldn't change, as we are focused on showing that $k(1) = F$).* Note that 4 is connected to both $5, 6$. By three coloring property, as $k(5) = T$, and $k(6) = \$$, $k(4) = F$.

Now consider nodes $2, 3$. 2 is connected to 4 and 3 is connected to 9. Therefore both $2, 3$ are connected to some F node, meaning that neither can be F for a valid 3-coloring. Note that $2, 3$ are also connected to each other, meaning that $k(2) \neq k(3)$. As there are only 3 possible colors for a 3-coloring, one of $k(2)$,$k(3)$ **must equal** \$ and the other must equal T. Therefore let $k(2) = T$, and $k(3) = \$$ *(these assignments could be swapped and the proof wouldn't change, as we are focused on showing that $k(1) = F$).*

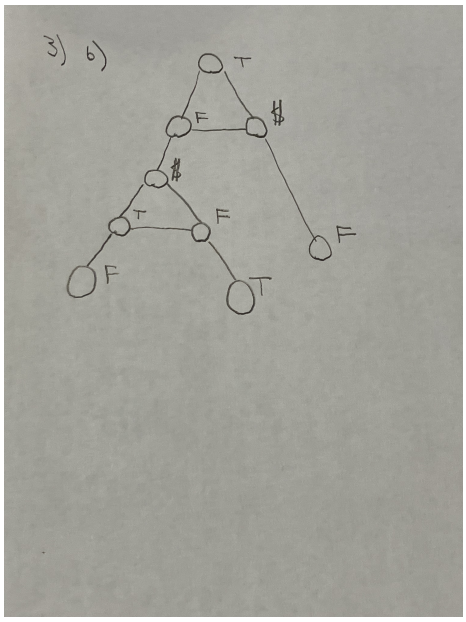
Finally we have that 1 is connected to both 2 and 3, and $k(2) = T$, and $k(3) = \$$. Therefore for a valid 3-coloring $k(1)$ **must equal** F. Therefore for $k(7) = k(8) = k(9) = F$, $k(1) = F$

3.2 b

Let $k(1) = T, k(2) = F, k(3) = \$, k(4) = \$, k(5) = T, k(6) = F, k(7) = F, k(8) = T, k(9) = F$.

As we have $k(7) = F, k(8) = T, k(9) = F$, $k_7, k_8, k_9 \in \{T, F\}$, however not all k_7, k_8, k_9 are F.

As there is no edge in G such that $k(u) = k(v)$, we have that the k assignment above is a valid 3 coloring for G. Therefore some valid 3 coloring must exist for the conditions described.



3.3

3-Coloring $\in NP$

The witness for this problem will be a set of nodes and their corresponding colors, which can be stored as a list of tuples labeling every node to some color, which will be of polynomial size. Given this set we can verify if this a valid 3 coloring in polynomial time by checking every edge in G and seeing if any two nodes attached by a given edge are the same color. Therefore given a polynomial size witness we can verify if G is $\in$ 3 coloring in polynomial time.

3-Coloring $\in NP$ **Hard**

Reduction from 3SAT problem

Given a ϕ instance of 3SAT we will construct a graph G with the following approach (see drawing for visual representation). We will have the $3-$Colors labeled as True (T), False (F), and \$ to reduce the 3SAT problem to the 3-Coloring coloring problem.

The first part of the reduction is creating nodes in the graph that assign consistent truth values to each variable, to make certain that for any x_i variable in 3SAT there is only one True or False assignment. This is done through adding nodes that represent every variable and its compliments $x_i, \bar{x}_i \in x_1, x_2 \cdots x_n$. For each node, connect x_i to $\bar{x}_i$ and then connect those two nodes to a third

node \$. This ensures that both x_i and $\bar{x}_i$ cannot be \$, which is important because \$ is not a valid encoding for x_i or $\bar{x}_i$ in 3SAT. Furthermore x_i must be the opposite truth value of $\bar{x}_i$ if a valid 3-Coloring of G exists, because each x_i is also connected to each $\bar{x}_i$. This follows the behavior of the compliment for a given variable.

Note that $((A \vee B) \vee C) = (A \vee B \vee C)$. Therefore, we will chain together two OR-gadgets, to get a triple OR to simulate a given clause for 3SAT. This is done by using an OR-gadget on A, B and then applying an OR-gadget to the output of A OR B with C. This double OR-gadget block is a graph that will output T if any input A, B, C are T. This block is shown in the problem statement for 3 its behavior is proved in 3A and 3B, that if all inputs to this gadget are F then the output will always be F, and if there exists some input that is not F, then there always exists some output which will be T. Therefore the double OR gadget follows the required behavior for a clause on 3 variables in 3SAT. We will create one of these blocks from every clause in ϕ. Instead of using A, B, C, for each clause we will connect the variable nodes set up previously to corresponding variables in that clause (to respective $x_i, \bar{x}_i$ variables).

Next, to emulate the $\wedge$ behavior between every clause in the 3SAT problem, we connect the output of every double OR-gadget to two other nodes, a F node and a \$ node. This ensures that if any double OR-gadget/ clause block doesn't output T, then the graph will not be a valid 3 coloring. Through this structure we simulate the conjunction of clauses in the 3SAT problem.

Computable

For each clause $\in \phi$ we add a constant amount of nodes (equal to the size of the double OR block) to our graph G. We also add a constant amount of nodes for each variable in ϕ. We also add a F and \$ node to keep the variable assignment consistent and check if the OR blocks evaluate to True in our graph. Therefore the size of graph grows at a constant rate with respect to the size of ϕ, so our graph can be computed in polynomial time.

YES TO YES

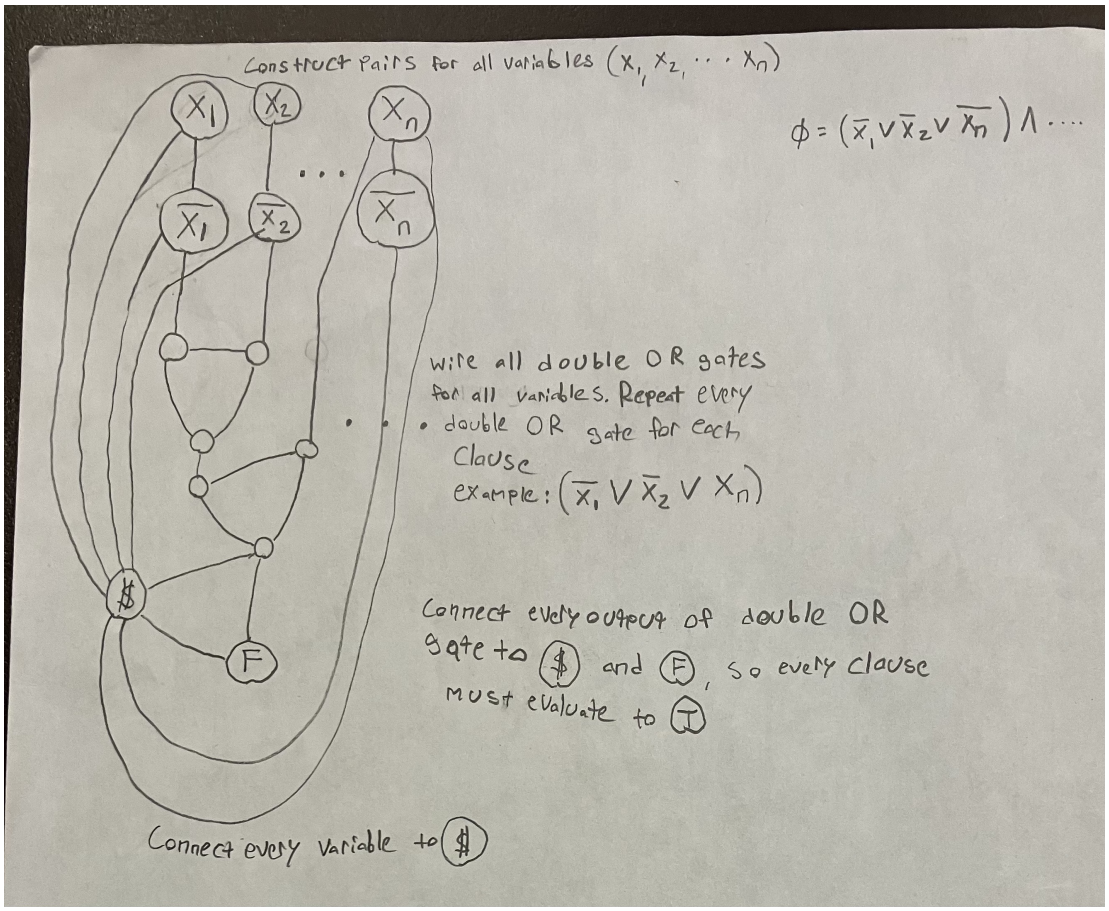
Assume that we have some witness x such that $w \in$ 3SAT. Therefore x will be some assignment of variables so that $w \in$ 3SAT. We can convert x to be a labeling of nodes in G for 3-coloring by converting each variable node to True depending if that variable is set to True in x, or False if that variable is set to False in x. If x is an assignment of variables such that 3SAT is True, then every clause / double OR block will evaluate to True, (as proved in 3A,3B this gadget follows the 3 OR behavior of a clause). Therefore each of the output nodes after these gadgets are colored T which follows the behavior needed for a valid 3 coloring for G. Therefore starting with coloring variable nodes according to the variable assignments given by x and coloring the graph as shown in $3a, 3b$

will result in a valid coloring G. Therefore we can save this color assignment as y, showing that given y, we can verify that $G \in$ 3-color.

NO TO NO

For the No case, we will solve the contra positive. Given a 3SAT problem w, and constructing G from w as described above, assume that there's some witness y such that using y we verify that G has a valid 3-Coloring in polynomial time. Our witness y will be a list of tuples which contains the assignments of colors to nodes in G. From y, we want to create a witness x such that we can verify that w in 3SAT in polynomial time. Our witness for 3SAT will be an assignment of truth values to variables in the problem; once we have an assignment for each variable we can query through w in polynomial time and see if the term assignments given by x causes w to evaluate to True. In order to get the variable assignments, we will check the nodes in our graph designated to $x_1, x_2 \cdots x_i$, and set each variable to True or False depending on the color labeling of each x_i. If y is a valid 3-coloring for G, then every node designated to $x_1, x_2 \cdots x_i$ must be labeled True or False because each of those nodes is connected to \$. Furthermore each variable node is also connected to the respective input of a double OR block corresponding to a given clause in w as described above; all of the outputs of these double OR blocks are then connected to False and \$ labeled nodes - therefore each of these must evaluate to T or else a 3-color labeling would not be possible. As our OR gadgets mirror the behavior of a given clause (again see $3a, 3b$), we know that the labels given to the inputs $x_1, x_2 \cdots x_i$ must result in w being satisfiable, or else our labeling would not provide a valid 3-coloring for our constructed G. Therefore our labeling of nodes $x_1, x_2 \cdots x_i \in Y$ that cause G to be 3 colorable must cause every clause in w to evaluate to True, meaning that we can take those same labels as truth value assignments to variables in x as our witness for 3SAT w to be satisfiable.

As we have that 3-Coloring $\in NP$ and 3-Coloring is NP Hard we have that 3-Coloring is NP Complete



4

MIN-MISSED-DEADLINES $\in NP-$**COMPLETE**

MIN-MISSED-DEADLINES $\in NP$

Our witness x will be a given schedule of nodes for task system $\langle G, Deadline, k\rangle$ assigned to times such that the schedule x has a cost at most k. x will be a list of 2 tuples, so it will have polynomial length relative to the size of G. Given x we can verify if there exists a schedule with cost at most k for $\langle G, Deadline, k\rangle$ by first checking that x satisfies the two conditions given in the problem statement; x assigns tasks that are injective and x assigns tasks in an order that follows the incoming edges in the directed graph. These can be checked by looking for duplicate assignments and by stepping through the graph and seeing if any tasks are assigned before their incoming edge. In the second part of the verification, we can query through each node v in G and count the amount of tasks

where StartTime(v) $\geq$ Deadline(v), and if this count is at most k, then we can verify that $G \in$ MIN-MISSED-DEADLINES. As we check each edge once, we can verify a purported witness in polynomial time.

MIN-MISSED-DEADLINES $\in NP$ Hard

We will do a reduction from k CLIQUE problem. Given some undirected graph G and integer k

$f(\langle G, k\rangle) = \langle G', Deadline, k'\rangle$

Let G' be a directed graph of task nodes where each vertex $\in G$ is $\in G'$ (as a vertex node) and each edge $(u, v) \in G$ is $\in G'$ (as an edge node) with two incoming edges from u, v

$Deadline(v \in V) = k + 1$ for each vertex node in G'
$Deadline(e \in E) = \binom{k}{2} + k + 1$ for each edge node in G'

$k' = |V| - k + |E| - \binom{k}{2}$

Polynomial Computable
The reduction takes G and expands it by 1 node and 2 directed edges for each $|E| \in G$ and adds every $|V| \in G$ to G'. The Deadline function and k' are easily calculable integers. Therefore the reduction is computable in polynomial time.

Yes to Yes
Assume there is a witness x showing $G \in k$-CLIQUE. Using x, we can construct a schedule y showing that $G' \in$ MIN-MISSED-DEADLINES. Our witness x will be a labeling of nodes $\in G$ such that all nodes in x are connected $\in G$ and $|x| \geq k$. In a $k-$clique, there are $\binom{k}{2}$ edges and k nodes in the sub graph of the clique. Therefore, we can create schedule y by scheduling all the vertex nodes in x followed by all the edge nodes in x. This will be a valid schedule for k', as we can schedule every vertex node and edge node in the clique without being penalized because there are k vertex nodes and $\binom{k}{2}$ edge nodes in the clique, and we have $k' = |V| - k + |E| - \binom{k}{2}$. In our schedule, we have the vertex nodes of the $k-$clique as $1, 2 \cdots, k$ (these are all valid as they are scheduled before $k + 1$) and the edge nodes of the $k-$clique as $k + 1, k + 2 \cdots, k + \binom{k}{2}$ (these are all valid as they are scheduled before $k + \binom{k}{2} + 1$). Note we need to $+1$ to our formula because StartTime(v) = Deadline(v) that is counted as a late scheduling. Therefore, our schedule will be penalized (schedule a task after its deadline) for all edges vertices and node vertices outside the k-clique x, which will be $|V| - k + |E| - \binom{k}{2}$. As $k' = |V| - k + |E| - \binom{k}{2}$, then y is in k'. Therefore we have created a witness (the schedule y) that shows a valid schedule exists for $\langle G', Deadline, k'\rangle$.

No to No

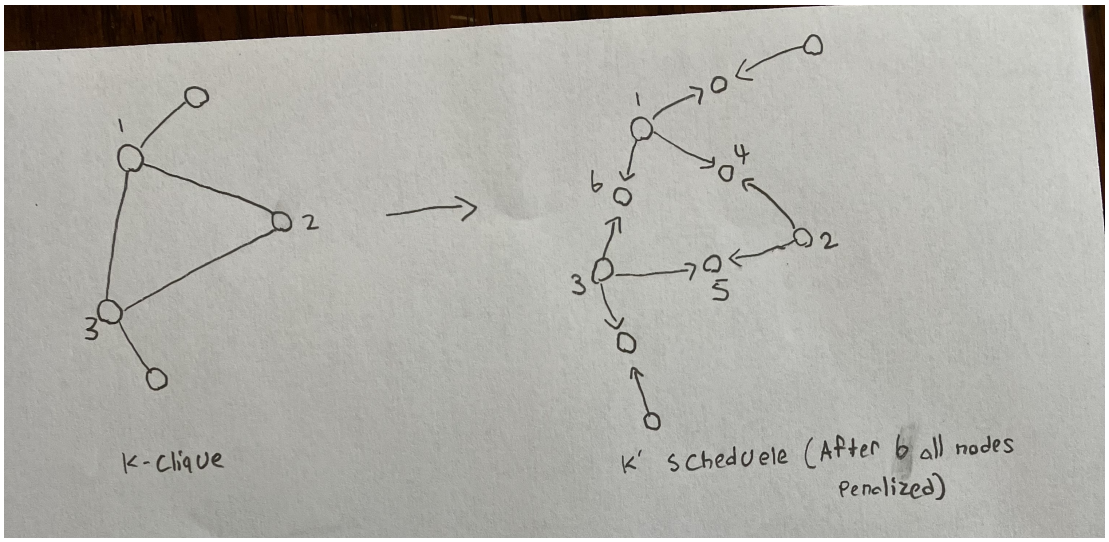
Assume there is a witness, a schedule y showing that our constructed task system $G' \in$ MIN-MISSED-DEADLINES for cost k. First, note that after scheduling $\binom{k}{2} + k$ nodes, every node scheduled will result in a penalty in G', as each vertex node is labeled k and each edge node is labeled $\binom{k}{2} + k$; so after $\binom{k}{2} + k$ any schedule will be penalized for all remaining nodes. With the strategy of selecting every node in a k clique followed by every edge in a k clique, we will schedule exactly $\binom{k}{2} + k$ tasks without penalty, which will put our schedule with a cost of exactly k'.

Consider if we can achieve the same score with a different scheduling. First, consider if we can achieve the same score by not scheduling k vertex nodes to start. Therefore we schedule at least 1 edge node in the first k steps. Then every vertex node becomes penalized after k, so we can only to schedule edge nodes without penalty. However,in the way we have constructed G' we can only schedule edge nodes where both the task nodes have already been scheduled. However with $k-1$ total vertex nodes scheduled we can only have less than $\binom{k}{2}$ total edges between $k-1$ nodes, as there are $\binom{k}{2}$ total edges between k nodes in a k-clique. Therefore we would run out of potential edges to schedule, meaning that if we don't schedule k vertex nodes to start we will achieve a schedule that is not in k', so we must schedule k node vertices first.

Next, consider if we can achieve the same score as a k-clique by scheduling some set of k vertex nodes to start such that k vertices are not a k clique. As shown before, we must pick k vertex nodes to get a schedule in k', however assume these k vertices are not a clique. If they are not a clique then there cannot be $\binom{k}{2}$ edges between the vertex nodes; therefore after k steps when we are penalized for every node, we have less than $\binom{k}{2}$ edges to schedule. Therefore our schedule would have a score greater than $|V| - k + |E| - \binom{k}{2}$, meaning it would not be in k'. Therefore we must pick k vertex nodes followed by $\binom{k}{2}$ edge nodes such that k vertex nodes form a k-clique for any valid k' schedule in G'. To transform G' to G and find a valid k-clique, we can remove all edge nodes $\in G'$ and save the k vertex nodes selected first as a valid $k-$clique for G. Therefore given a valid k' schedule for G' we can find a $k-$clique for G.

As we have that MIN-MISSED-DEADLINES is $\in NP$ and MIN-MISSED-DEADLINES is NP Hard we have that MIN-MISSED-DEADLINES is NP Complete

Drawing of k-clique to k' schedule


1
2
3
K-Clique
1
4
6
2
3
5
K' Schedule (After 6 all nodes penalized)