CSCI 570: Homework set 5

Mounika Mukkamalla

1

a. After removing lower bound on the given graph, we get the following circulation graph:

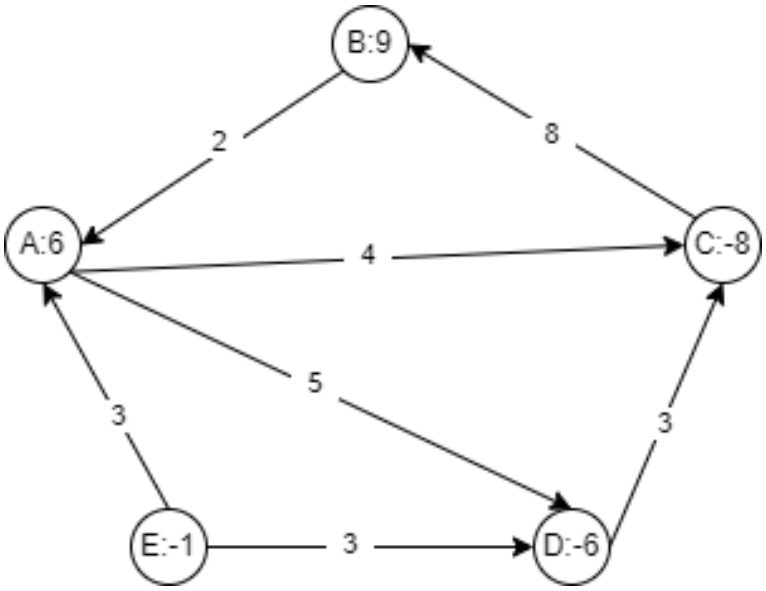

Figure 1: Updated circulation graph, G1

Demands on each vertex:
A: 6
B: 9
C: -8
D: -6
E: -1

b. By converting the circulation graph G1 to Network Flow:

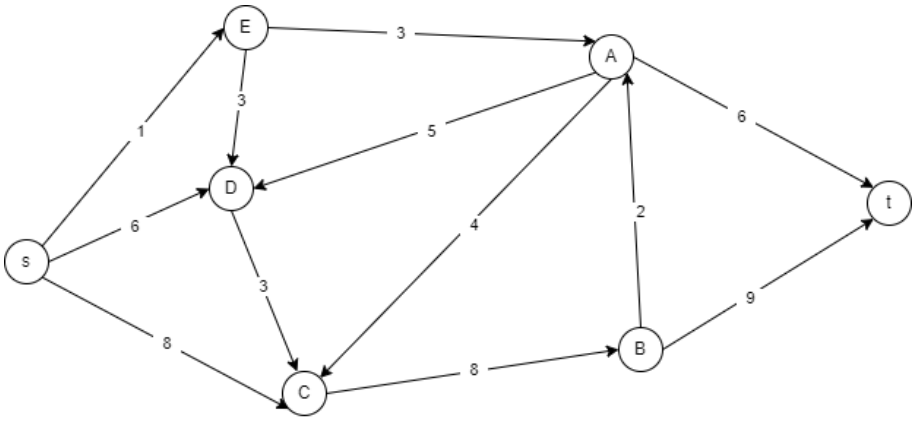

Figure 2: Updated circulation graph, G1

Max flow is **9**
And the paths for this flow are from:
$s->C->B->t$, with flow: 8
$s->E->A->t$, with flow: 1
Hence total flow = 8+1 = 9

c. Necessary condition for feasibility is $\sum_{v \in V} d(v) = 0$
$\sum_{v \in V} d(v) = d(A) + d(B) + d(C) + d(D) + d(E) = 6 + 9 - 8 - 6 - 1 = 0$
Necessary condition is satisfied.
Sufficient condition is: There exists a feasible solution iff max flow = sum of positive demands of all the vertices.
That is $\sum_{d(v)>0} d(v) = D$.
In circulation graph, G1, vertices A and B have positive d(v).
Hence $\sum_{d(v)>0} d(v) = d(A) + d(B) = 6 + 9 = 15$ - which is not equal to max flow, 9
Hence there doesnot exists a feasibe cirulation for the given graph.

2

Q2.

3

a) Let s, b, p represent the amounts of steel, brass and pewter respectively.
Given company currently has 8 hours of time.
And to manufacture one ton of steel it takes 3 hrs, hence to manufacture s tons of steel, it takes 3*s hrs.
Similarly, to manufacture b tons of brass, it takes 1*b hrs of time.
And to manufacture p tons of pewter, it takes 2*p hrs of time.
Hence, the constraint becomes, $3 * s + b + 2 * p <= 8$
And company also has 20 tons of special substance(ss).
To manufacture one ton of steel, 3 units of ss is needed, hence to manufacture s tons of steel, we need 3*s units of ss.
Similarly, we need 10*b units of ss to manufacture b tons of brass
and we need 5*p units of ss to manufacture p units of pewter.
Hence the linear equation becomes, $3 * s + 10 * b + 5 * s <= 20$
Linear problem:

$$max(s + b + p)$$

subject to:

$$3 * s + b + 2 * p <= 8$$

$$3 * s + 10 * b + 5 * s <= 20$$

$$s >= 0$$

$$b >= 0$$

$$p >= 0$$

b) supply of special substance is maximized: $max(3 * s + 10 * b + 5 * s)$
An additional constraint is being added on sum of amounts of steel and pewter, which is :

$s + p >= 2$ So the linear problem is formulated as:

$$max(3 * s + 10 * b + 5 * s)$$

subject to:

$$-s - p <= -2$$
$$3 * s + b + 2 * p <= 8$$
$$3 * s + 10 * b + 5 * s <= 20$$
$$s >= 0$$
$$b >= 0$$
$$p >= 0$$

4

Q4. Let the amount of cement received by city C from A be, C_A and amount of cement received by city C from B be C_B
Similarly, for city D, from A, D_A, and from B, D_B
Given the city C needs atleast 50 tons of cement, Hence total cement at C = $C_A + C_B >= 50$
Given city D needs atleast 60 tons of cement, Hence total cement at D = $D_A + D_B >= 60$
Given A has 70 tons of cement, hence amount of cement exported by A should be less than 70
$\implies C_A + D_A <= 70$
And given B has 80 tons of cement, hence amount of cement exported by B should be less than 80
$\implies C_B + D_B <= 80$
We need to minimise the cost of the all the export from cities A, B to cities C and D.
We know it costs 1 unit for one ton of export from A to C, Hence cost of exporting C_A tons from A to C = $1 * C_A$
Similarly, cost of exporting D_A tons from A to D is $2 * D_A$
cost of exporting C_B from B to C is $3 * C_B$
cost of exporting D_B from B to D is $4 * D_B$
Overall cost = $C_A + 2 * D_A + 3 * C_B + 4 * D_B$
Hence the linear program becomes:

$$min(C_A + 2 * D_A + 3 * C_B + 4 * D_B)$$

subject to:

$$-C_A - D_A >= -70$$
$$-C_B - D_B >= -80$$
$$C_A + C_B >= 50$$
$$D_A + D_B >= 60$$
$$C_A, C_B, D_A, D_B >= 0$$

5

Dual program of given linear program is:

$$min(-y_1 + 5y_2 + y_3)$$

subject to:

$$y_1 >= 1$$
$$-y_1 + y_2 >= -3$$
$$-3y_1 + 3y_2 + y_3 >= 4$$
$$y_1 >= 0, y_2 >= 0, y_3 >= 0$$

6

Let G be the given graph and V be the set of vertices and E be the set of edges .
Let x_v represent if vertex v is in the vertex cover or not. x_v is 0 if v is in the vertex cover and is 1 if v is not in the vertex cover.
We need to minimise the number of vertices in the vertex cover with a constraint:
atleast one vertex in each edge is included in the vertex cover.
Therefore, for an edge between (u, v) either u or v must be in the vertex cover.
$\implies$ x_u or x_v is 1 or both are 1.
$\implies$ for every edge, e between (u, v), $x_u + x_v >= 1$
Hence we can formulate vertex cover in Integer Linear Programming as:

$$min(\sum_{v \in V} x_v)$$

subject to

$$\forall (u, v) \in E, x_u + x_v >= 1$$
$$\forall u \in V, x_u \in \{0, 1\}$$

7

Given there exists an algorithm to check if there exists a satisfiable solution to a 3-SAT instance in polynomial time. Let this function be $check\_satisfiability()$
Assume our 3-SAT instance contains n variables, $X = \{x_1, x_2, ..., x_n\}$
Algorithm:
0. Check $check\_satisfiability()$ initially return false if $check\_satisfiability()$ return false.
1. Assign $x_1 = 0$ once and $x_1 = 1$ once and $check\_satisfiability()$ for both cases.
2. fix the value of x_1 for which there exists a satisfiable solution, i.e., for which $check\_satisfiability()$ returns true. if there exists solution for both then assign either of the values to x_1.
3. Now assign $x_2 = 0$ once and $x_2 = 1$ once and run $check\_satisfiability()$ for both cases with x_1 fixed from previous step.
4. Now Fix value of x_2 for which $check\_satisfiability()$ returns true. if $check\_satisfiability()$ returns true for both cases, assign either value to x_2

5. Continue the process for all the variables until you reach the end of the set X, x_n
6. Since in each step, we are fixing values of x_i, at the end we are left with values for all the variables x_i in X for which there exists a satisfiable solution.
7. return x_i for $1 <= i <= n$

Time complexity: The above algorithm traversal through the set X. each time, we call $check\_satisfiability()$ function which is polynomial and we call n times and hence overall complexity is polynomial.

8

We need to prove 5-coloring problem is NP-complete.
For any problem to be proved as np-complete, we need to prove the problem is np and also np-hard.
Case1: To prove 5-coloring is np.
Let G‘(V, E) be the given graph.
To prove 5-coloring is np, we need to prove that for a given potential solution to graph G‘, we can verify if this solution is right in polynomial time.
Proof:
1. Traverse all the vertices of the graph, G‘.
2. for each vertex, v, check if adjacent vertices to v are of different color in comparison with v or not.
3. If for all vertices, their adjacent vertices are of different color with respect to that vertex, and total number of different colors is less than or equal to 5 then we say that the graph is 5 -colorable.
The above algorithm is just graph traversal, and graph traversal takes linear time in terms of vertices and edges.
Hence 5-coloring problem is np.

Case2: To prove 5-coloring is np-hard.
To prove a problem is np-hard, we need to reduce a np-hard/np-complete problem to our problem.
Let’s consider 3-coloring problem, we need to prove 3-coloring is polynomial time reducible to 5-coloring.
That is 3-coloring $\leq p$ 5-coloring
Construction:
Let G be the given graph.
1. Construct a graph G‘ with all the vertices and edges same as G.
2. Add 2 additional vertices v_1 and v_2 to graph G‘.
3. Add edges from all the vertices in graph G‘ to v_1 and v_2.
4. Add an edge between v_1 and v_2
So, the graph becomes:

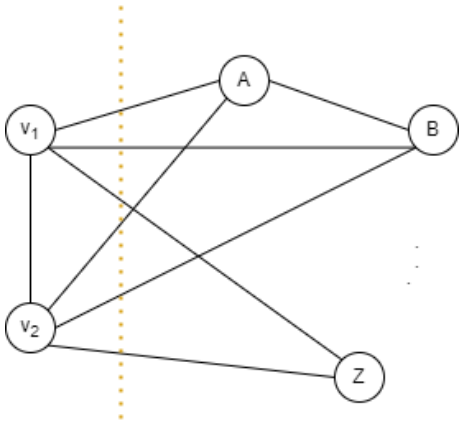

Figure 3: 5 coloring construction graph, G‘

The orange line above shows the new edges added from vertices v_1 and v_2 to the remaining vertices and

the right side of orange line is a clone of original graph G.
Claim: G is 3-colorable if and only if G‘ 5-colorable.
Proof a) If G is 3-colorable then G‘ is 5-colorable.
By construction, if G is 3-colorable, all the vertices other than v_1 and v_2 are 3-colorable.
In graph G‘, since v_1 is connected to all the vertices, a new color, c_4 needs to be assigned to v_1
And similarly, since v_2 is connected to all the vertices, a new color, c_5 needs to be assigned to v_2
Hence due to addition of 2 new vertices, v_1 and v_2, 2 new colors are added, and since remaining vertices in G‘ other than v_1 and v_2 can be colored using 3 colors, overall graph G‘ can be colored in 3+2, 5 colors.
Hence graph G‘ is 5-colorable.

Proof b) If G‘ is 5-colorable then we can construct a 3-colorable graph from it.
We are given G‘, we need to construct a 3 colorable graph.
Due to our construction, we definitely know that there exists atleast 2 vertices in the graph which are connected to all the other vertices.
Hence, remove any 2 vertices in the graph G‘ which are connected to all the vertices(other than self).
since these 2 vertices are connected to all the vertices, according to 5-colorable problem definition, these 2 vertices need to have different colors compared to all the other vertices in G‘.
That means these 2 vertices are uniquely colored with 2 different colors and when removed from the graph G‘, the left out graph has vertices which are colored with upto 3 different colors.
Hence we proved that we can construct a 3-colorable graph from 5-colorable graph.

Therefore, we proved out claim in both ways.
Hence, 5-colorable is atleast as hard as 3 colorable
And from 2 cases, we get to prove that 5-colorable problem is np-complete.

9

We need to prove that Longest path(LP) problem is np-complete.
Length of the longest path is assumed as number of edges here in my solution.
Case1: To prove Longest path problem is np.
Assume we are given a path, p1 as a potential solution to longest path problem. to prove LP is np, we need to verify if this is the solution to the LP problem.
1. Traverse the path and check if there are any repeated vertices. if there are repeated vertices in the traversal then return false.
2. if there are no repeated vertices and number of edges $>= k$ then return true.
This is just a traversal and hence takes polynomial amount of time.
Hence LP is np.
Case2: To prove LP is np-hard
Consider hamiltonian path problem to prove LP is np-hard.
need to prove, hamiltonian path $\leq p$ Longest Path
Construction:
Let G(V, E) be the given graph. Construct a new graph G‘ which is clone to G, i.e., with the same number of vertices as G and same edges as G.
Longest path problem is to decide if G‘ has simple path of length $>= k$
Claim: There exists a Hamiltonian path for graph G with v vertices if and only if there exists simple path in G‘ of length v-1 where v is the number of vertices.
Proof a) if there is a Hamiltonian path of v vertices, p1 for graph G then there exists simple path in G‘ of length v-1.
p1 is hamiltonian $\implies$ p1 visits all vertices of the graph G only once.
Hence p1 is simple path(since p1 has vertices which are visited only once). and since it visits all the vertices, v, number of edges will be v-1.
In clone graph G‘, we can get a clone path p1‘ which is same as p1 but in G‘.
Hence there exists a simple path p1‘ in G‘ for which length of the path is v-1.
$\implies$ there exists a simple path in G‘ of length v-1.

Proof b) If there exists simple path, p1‘ in G‘ of length v-1 then we have a Hamiltonian path with v vertices in G.
We need to construct a graph for the hamiltonian path:
Create a clone of G‘ with all same vertices and edges as G‘. and name this graph G(since this graph is same as our 1st hamiltonian graph).
we know that number of vertices in G‘ is v.
Since p1‘ is simple, all the vertices are non-repeated and there are v-1 edges, therefore number of vertices in p1‘ must be number of edges plus 1 which is v.
Hence all the vertices in G‘ are included in p1‘
With p1‘, we construct p1 in G with all the vertices and edges same as p1‘ except in G.
ie., by traversing p1‘, for each vertex we get find corresponding vertex in G. This results in p1 path for G.
Since p1‘ is simple, p1 is also simple(due to construction). and number of vertices in p1 is v and number of vertices in graph G is v.
Hence by definition, p1 is hamiltonian path for graph G.
Hence we proved b).

In both side proofs, we have obtained simple path with length v-1 - this can be considered as a subproblem for our longest path(simple path with length $>= k$)

Therefore, Longest path problem is atleast as hard as hamiltonian path problem.
And from both cases, Longest path problem is proved to be np-complete.

10

We need to prove that given scheduling problem is np complete.
Case1: To prove scheduling problem is np.
Let a set of courses, S, is given as a potential solution to the given problem.
Traverse through the set, S, for each course, we need to check if there exists any conflicting time slots with the remaining courses - which is a traversal through the remaining courses.
If there exists no conflict and size of set S is atleast k then S is a solution else it is not.
The above algorithm takes polynomial time.
Hence scheduling problem is np.
Case 2: To prove scheduling problem is np complete.
We can reduce from Independant set.
need to prove, Independant set $\leq p$ scheduling problem
Construction:
1. Assume G is the given graph with (V, E) as vertices and edges
2. For each vertex, v in G we construct a course.
3. For each edge between 2 vertices, u and v, create a conflict in the time intervals between corresponding courses.
4. Assign a disjoint set of intervals for each course satisfying the conflicts in step 3.
Claim: For a graph, G(V, E), S is the independent set of size k if and only if we can select non-conflicting k courses for scheduling problem.
Proof a) Given a graph, G(V, E) and if S is the independent set of size k, then we can select k non-conflicting courses for the scheduling problem.
Since S is an independent set, no 2 nodes in set S are connected.
Hence by construction, nodes represent our courses and edges represent conflicts
hence no 2 courses corresponding to the nodes in set S are connected and hence no 2 corresponding courses have conflicts.
$\implies$ All the corresponding courses are non-conflicting to each other.
Since we have k nodes in set S, we have k courses which are all non-conflicting to each other.

Proof b) If we have k non conflicting courses then we have a graph with independent set of size k.

Constructing a new graph from courses:
1. Consider each course as a vertex in the new graph.
2. Add a new vertex, u to the graph.
3. Add edges between this new vertex, u with all the other vertices
There exists no edge between 2 non-conflicting courses(according to our construction). Hence the vertices representing these courses would be in the independent set of the new graph since no vertex is connected to any other vertex(representing the non-conflicting course).
Therefore, all the vertices representing the non-conflicting courses are a valid elements in the independent set.
Therefore all k non conflicting courses are valid elements in independent set.
We can't have vertex u in the independent set along with these vertices because it is connected to all the other vertices and would contradict the independent set defintion.
Therefore, we can obtain an independent set of size k from k non-conflicting courses.

Hence proved both ways.
Obtaining k non-conflicting courses is a sub problem for obtaining atleast k non-conflicting courses.
And hence our scheduling problem is atleast as hard as Independent set problem
Hence from above 2 cases, we proved that our scheduling problem is np-complete.