INFO 6205 – Program Structures and Algorithms Assignment 4

Student Name: Yuetong Guo

Professor: Nik Bear Brown

**Q1 (20 Points)**

**Given a directed graph G = (V,E), a cycle-cover is a set of vertex-disjoint cycles so that each vertex v € *V* belongs to a cycle. On other words, a cycle cover of a graph G is a set of cycles which are sub-graphs of G and contain all vertices of G. If the cycles of the cover have no vertices in common, the cover is a called vertex-disjoint cycle cover or simply a disjoint cycle cover.**

**The *vertex-disjoint cycle-cover problem* asks whether a given directed graph has a vertex-disjoint cycle cover.**

**A. (10 points) Is the vertex-disjoint cycle-cover problem in P? If so, prove it.**

**Ans:**

The Vertex-Disjoint Cycle Cover problem asks whether a given directed graph has a vertex-disjoint cycle cover. Unfortunately, this problem is known to be NP-complete even for simple graph classes such as bipartite graphs. Therefore, it is unlikely to have a polynomial-time algorithm to solve it in general.

There are some special cases where the Vertex-Disjoint Cycle Cover problem can be solved in polynomial time, such as for directed acyclic graphs (DAGs). In a DAG, a vertex-disjoint cycle cover can be obtained by finding a topological ordering and then forming cycles by connecting vertices in order, so that each vertex is part of exactly one cycle. However, in the general case, we would need to resort to approximation algorithms or other heuristics to find a good solution.

On the other hand, the Vertex Cover problem is a well-known NP-complete problem, which asks for a minimum-sized set of vertices that covers all edges of the graph. There are approximate polynomial-time algorithms to solve the problem though, such as a naive approach that considers all subsets of vertices one by one and checks if they cover all edges of the graph.

It's important to note that these two problems are different, even though their names sound similar. The Vertex-Disjoint Cycle Cover problem is specifically concerned with finding a set of vertex-disjoint cycles that cover all vertices of the graph, while the Vertex Cover problem is concerned with finding a set of vertices that cover all edges of the graph.

For this specific question, The Vertex-Disjoint Cycle Cover problem is in P for directed acyclic graphs (DAGs), as a solution can be found in polynomial time using a modification of the depth-first search algorithm.

The algorithm works as follows:

1. Perform a topological sort of the graph to obtain a linear ordering of the vertices.

2. For each vertex v in the order, perform a depth-first search from v until either a cycle is found or all reachable vertices have been visited. If a cycle is found, add it to the cycle cover and mark all the vertices in the cycle as visited.

3. Repeat step 2 for any remaining unvisited vertices in the order.

4. At the end of the algorithm, the cycle cover contains a set of vertex-disjoint cycles that cover all the vertices of the graph. The running time of the algorithm is O(|V|+|E|).

However, for general directed graphs, the Vertex-Disjoint Cycle Cover problem is NP-complete. This can be shown by a reduction from the Hamiltonian Cycle problem, which is known to be NP-complete. Therefore, the Vertex-Disjoint Cycle Cover problem is in P for DAGs, but not in P in general.

**B. (5 points) Suppose we require each cycle to have at most three edges. We call this the *3-cycle-cover problem.* Is the 3-cycle-cover problem in NP? If so, prove it.**

**Ans:**

The Cycle Cover Problem involves determining whether a given directed graph G = (V, E) contains a collection {C1, C2, · · · , Ck}, where each Ci is a simple directed cycle in G and every vertex v in G is contained in exactly one cycle Ci.

The 3-Cycle Cover Problem is similar to the Cycle Cover Problem, but with the added restriction that each cycle must have at most three edges, i.e., |Ci| ≤ 3 for all i.

Examples of “Yes” and “No” instances for both problems are shown below, with cycle covers displayed in bold:

(a) A graph with no cycle covers for Cycle Cover and 3-Cycle Cover;

(b) A graph with a cycle cover for Cycle Cover, but no cycle cover for 3-Cycle Cover;

(c) A graph with cycle covers for both Cycle Cover and 3-Cycle Cover.

Chart, radar chart

Description automatically generatedTop of FormBottom of Form

To prove that 3-Cycle Cover is in NP, we need to show that given a proposed solution (i.e., a collection of cycles), we can verify in polynomial time whether the solution is valid (i.e., if it covers all vertices of the graph and each cycle has length at most three).

To verify a solution, we simply need to check that each cycle is indeed a cycle with no repeated vertices, and that every vertex in the graph is covered by exactly one cycle. This can be done in linear time by iterating through each cycle and each vertex and checking that the cycle contains the vertex and that no other cycle contains it. Additionally, we can check that each cycle has length at most three by counting the number of edges in the cycle. Therefore, 3-Cycle Cover is in NP.

**C. (10 points) Is the *3-cycle-cover problem* in NP-complete? If so, prove it.**

**Ans:**

In B, we already know that 3-cycle-cover problem is NP. So we just need to prove it’s NP hard here.

To prove that 3-Cycle Cover is NP-hard, we will reduce from 3-SAT. Given an instance of 3-SAT with n variables and m clauses, we can construct a directed graph G as follows: (¬ == not)

* For each variable x\_i, create two vertices v\_i and v'\_i.
* For each literal x\_i or its negation ¬x\_i in the 3-SAT instance, create a clause gadget consisting of three vertices a, b, and c, and add edges to form a triangle. If the literal is x\_i, add a directed edge from v\_i to a and from ¬v'\_i to c. If the literal is ¬x\_i, add a directed edge from ¬v\_i to b and from v'\_i to c.
* Graphical user interface

  Description automatically generated with medium confidenceFor each pair of conflicting literals, such as x\_i and ¬x\_j, create a conflict connection gadget consisting of four vertices x, y, z, and w, and add edges as shown in the figure in the prompt.

Now, we claim that the 3-Cycle Cover problem on G has a solution if and only if the 3-SAT instance is satisfiable.

Suppose first that the 3-SAT instance is satisfiable. Then there exists an assignment of truth values to the variables that satisfies all clauses. For each true variable x\_i, we can select the cycle (a, x, c) in the clause gadget for the literal x\_i, and the cycle (b, x) in the conflict connection gadget for each conflicting literal ¬x\_j, to form a 3-cycle cover of G. This covers all vertices in the clause gadgets and conflict connection gadgets, so it covers all vertices in G.

Conversely, suppose that G has a 3-cycle cover. Then every vertex in G is covered by at least one cycle of length at most 3. Consider each variable x\_i. If both v\_i and v'\_i are covered by a cycle, then set x\_i to true; otherwise, set x\_i to false. By construction, every clause gadget is covered by a cycle of length 3 that includes one of its true literals, so all clauses are satisfied. Moreover, every conflict connection gadget is covered by at most one cycle, so there are no conflicting assignments for the variables. Therefore, the 3-SAT instance is satisfiable.

Since we have shown that the 3-Cycle Cover problem is both in NP and NP-hard (by reducing 3-SAT to it), it follows that the problem is NP-complete.

**Q2 (20 Points)**

**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*)*, . . ., *(sk*, *tk)*. The problem is to decide whether there exist node- disjoint paths *P*1, *P*2, …, *Pk* so that *Pi* goes from *si* to *ti*.**

**Show that *Directed Disjoint Paths* is NP-complete.**

**Ans:**

The Directed Disjoint Paths Problem is defined as finding k node-disjoint paths between k pairs of nodes in a directed graph. It is known as the k disjoint shortest paths problem when each path Pi is a shortest path from si to ti.

The undirected version of the 2 disjoint shortest paths problem can be solved in polynomial time. However, the directed version of the 2 disjoint shortest paths problem is known to be NP-hard if the length of each edge can be zero. This implies that the directed k disjoint shortest paths problem is also NP-hard, even when k = 2.

Therefore, the Directed Disjoint Paths Problem is NP-complete.

However, if we don’t rely on the hardness of any other problem such as the disjoint shortest paths problem, we could directly prove Disjoint Paths Problem is NP-hard follow below logic.  
  
To show that Directed Disjoint Paths is NP-complete, we will reduce the well-known NP-complete problem of the 3-SAT problem to it. Given an instance of 3-SAT with n variables and m clauses, we construct a directed graph G with 2n+2m nodes and 3n+3m edges as follows:

For each variable xi, we add two nodes si and ti to G, and two directed edges (si,ti) and (ti,si) with length 1.

For each clause Cj with literals ℓ1, ℓ2, and ℓ3, we add three nodes aj, bj, and cj to G, and the following edges:

(sj,aj) and (aj,tj) if ℓ1 = xi or ¬xi, with length 0 if ℓ1 = xi and length 1 if ℓ1 = ¬xi.

(sj,bj) and (bj,tj) if ℓ2 = xi or ¬xi, with length 0 if ℓ2 = xi and length 1 if ℓ2 = ¬xi.

(sj,cj) and (cj,tj) if ℓ3 = xi or ¬xi, with length 0 if ℓ3 = xi and length 1 if ℓ3 = ¬xi.

(ai,bj), (ai,cj), (bj,cj), (bj,ai), (cj,ai), and (cj,bj) with length 1.

We claim that G has a set of k disjoint paths from (s1,t1) to (sk,tk) if and only if the 3-SAT instance is satisfiable.

Suppose first that the 3-SAT instance is satisfiable. Then we can choose one literal for each variable that is true in the satisfying assignment and route the corresponding path through the corresponding clauses in G, skipping those that contain the negated literal. Since the chosen literals satisfy each clause, each clause gadget in G can be covered by three pairwise disjoint paths from si to ti. Moreover, since the chosen literals are pairwise compatible, the conflict gadgets in G can be covered by two pairwise disjoint paths from si to ti.

Conversely, suppose that G has a set of k pairwise disjoint paths from (s1,t1) to (sk,tk). Then we can choose literals for each variable such that the corresponding paths in G route through the corresponding clauses. Specifically, if the path for xi goes through the clause gadgets for clauses Cj1, Cj2, and Cj3, then we set xi to true if the paths through aj, bj, and cj are in that order, and to false if they are in the reverse order. It is easy to verify that the chosen literals satisfy each clause, and therefore the 3-SAT instance is satisfiable.

Since the 3-SAT problem is NP-complete, and we have shown a polynomial-time reduction from it to Directed Disjoint Paths, it follows that Directed Disjoint Paths is also NP-complete.

**Q3 (20 Points)**

**You are organizing a game hack-a-thon and want to make sure there is at least one instructor who is skilled at each of the n skills required to build a game (e.g. programming, art, animation, modeling, artificial intelligence, analytics, etc.) You have received job applications from m potential instructors. For each of n skills, there is some subset of potential instructors qualified to teach it. The question is: For a given number k ≤ m, is possible to hire at most k instructors that can teach all of the n skills. We’ll call this the *Cheapest Teacher Set*.**

**Show that *Cheapest Teacher Set* is NP-complete.**

**Ans:**

1. To show that the Cheapest Teacher Set problem is in NP, we need to show that given a proposed solution (i.e., a set of at most k teachers), we can verify in polynomial time whether the solution is valid (i.e., if it includes at least one teacher qualified for each of the n skills).

To verify a solution, we simply need to check that there is at least one teacher in the solution who is qualified for each of the n skills. This can be done in linear time by checking each of the n skills and each of the teachers in the proposed solution. Therefore, the Cheapest Teacher Set problem is in NP.

2. To prove that the Cheapest Teacher Set problem is NP-complete, we will reduce the Set Cover problem to it. In the Set Cover problem, we are given a set U of n elements and a collection C of m subsets of U, and the question is whether there exists a subset C' of at most k sets in C such that the union of the sets in C' is equal to U.

To construct an instance of the Cheapest Teacher Set problem from a given instance of the Set Cover problem, we create n skills, one for each element in U. We also create m teachers, one for each set in C. Each teacher is qualified to teach the skills corresponding to the elements of the set they represent. We set k equal to the given parameter for the Set Cover problem.

3. We will show that the Cheapest Teacher Set instance has a solution if and only if the Set Cover instance has a solution. Suppose there is a solution to the Set Cover instance consisting of at most k sets whose union is equal to U. We can choose the teachers who correspond to those sets, giving us a set of at most k teachers who are qualified to teach every skill. Therefore, the Cheapest Teacher Set instance has a solution.

Conversely, suppose there is a solution to the Cheapest Teacher Set instance consisting of at most k teachers who are qualified to teach every skill. For each skill, we can choose the teacher who is qualified to teach that skill. The set of chosen teachers corresponds to a subset of at most k sets in C whose union is equal to U. Therefore, the Set Cover instance has a solution.

Since we have shown that the Cheapest Teacher Set problem is both in NP and NP-hard (by reducing the Set Cover problem to it), it follows that the problem is NP-complete.

**Q4 (20 Points)**

**Suppose you’re helping to organize a summer sports camp, and the following problem comes up. The camp is supposed to have at least one counselor who’s skilled at each of the *n* sports covered by the camp (baseball, volleyball, and so on). They have received job applications from *m* potential counselors. For each of the *n* sports, there is some subset of the *m* applicants qualified in that sport. The question is: For a given number *k < m*, is it possible to hire at most *k* of the counselors and have at least one counselor qualified in each of the *n* sports? We’ll call this the *Efficient Recruiting Problem*.**

**Show that Efficient Recruiting is NP-complete.**

**Ans:**

1. To show that the Efficient Recruiting Problem is in NP, we need to show that given a proposed solution (i.e., a set of at most k counselors), we can verify in polynomial time whether the solution is valid (i.e., if it includes at least one counselor qualified for each of the n sports).

To verify a solution, we simply need to check that there is at least one counselor in the solution who is qualified for each of the n sports. This can be done in linear time by checking each of the n sports and each of the counselors in the proposed solution. Therefore, the Efficient Recruiting Problem is in NP.

2. To prove that the Efficient Recruiting Problem is NP-complete, we will reduce the Set Cover problem to it. In the Set Cover problem, we are given a set U of n elements and a collection C of m subsets of U, and the question is whether there exists a subset C' of at most k sets in C such that the union of the sets in C' is equal to U.

To construct an instance of the Efficient Recruiting Problem from a given instance of the Set Cover problem, we create n sports, one for each element in U. We also create m counselors, one for each set in C. Each counselor is qualified to coach the sports corresponding to the elements of the set they represent. We set k equal to the given parameter for the Set Cover problem.

3. We will show that the Efficient Recruiting instance has a solution if and only if the Set Cover instance has a solution. Suppose there is a solution to the Set Cover instance consisting of at most k sets whose union is equal to U. We can choose the counselors who correspond to those sets, giving us a set of at most k counselors who are qualified to coach every sport.

Therefore, the Efficient Recruiting instance has a solution. Conversely, suppose there is a solution to the Efficient Recruiting instance consisting of at most k counselors who are qualified to coach every sport. For each sport, we can choose the counselor who is qualified to coach that sport. The set of chosen counselors corresponds to a subset of at most k sets in C whose union is equal to U. Therefore, the Set Cover instance has a solution.

Since we have shown that the Efficient Recruiting Problem is both in NP and NP-hard (by reducing the Set Cover problem to it), it follows that the problem is NP-complete.

**Q5 (20 Points)**

**Suppose you live with n − 1 other people, at a popular off-campus cooperative apartment, the *Ice-Cream and Rainbows Collective*. Over the next *n* nights, each of you is supposed to cook dinner for the co-op exactly once, so that someone cooks on each of the nights.**

**Of course, everyone has scheduling conflicts with some of the nights (e.g., algorithms exams, Miley concerts, etc.), so deciding who should cook on which night becomes a tricky task. For concreteness, let’s label the people, *P* ∈ {p1, . . . , pn}, the nights, *N* ∈ {n1, . . . , nn} and for person pi, there’s a set of nights Si ⊂ {n1, . . . , nn} when they are not able to cook. A person cannot leave Si empty.**

**If a person isn’t doesn’t get scheduled to cook in any of the n nights they must pay $200 to hire a cook.**

**A. Express this problem as a maximum flow problem that schedules the maximum number of**

**matches between the people and the nights.**

**Ans:**

This problem can be formulated as a bipartite matching problem in a bipartite graph, where one set of vertices represents the people (P) and the other set represents the nights (N). To construct the bipartite graph, add an edge (pi, nj) if person pi can cook dinner on night nj.

Once you have constructed the bipartite graph, convert it into a flow network by directing all edges from "people" to "nights," adding a source vertex s with edges from s to each person, and a sink vertex t with edges from each night to t. Set the capacity of each edge to be equal to 1. Then, find a maximum flow in the network using a maximum flow algorithm, such as the Ford-Fulkerson algorithm.

A feasible dinner schedule exists if and only if the maximum flow value equals n, which means that all n people are matched to distinct nights. In this case, examine the edges in the maximum flow that join people to nights to determine the optimal schedule. If the maximum flow value is less than n, there isn't a feasible schedule where all n people cook exactly once, and one or more people will need to pay $200 to hire a cook.

Constructing the bipartite graph takes time O(n^2), as there are at most n^2 edges in the bipartite graph, with each of the n people potentially joined to n nights. Adding the additional edges to create the directed flow network takes time O(n). The overall complexity of the solution is dominated by the maximum flow algorithm, which has a complexity of O(n^3) in this case, using the Edmonds-Karp algorithm as an example. Therefore, finding a feasible dinner schedule or concluding that there isn't one can be done in time O(n^3).

Diagram

Description automatically generatedThe example graph would be:

**B. Can all n people always be matched with one of the n nights? Prove that it can or cannot.**

**Ans:**

We cannot guarantee that all n people can always be matched with one of the n nights, as it depends on the specific constraints (Si) for each person.

To prove that it is not always possible to match all n people with n nights, consider the following counterexample:

Suppose we have 3 people (n = 3): P = {p1, p2, p3} and 3 nights: N = {n1, n2, n3}. The constraints for each person are as follows:

* p1 can't cook on nights: S1 = {n1}
* p2 can't cook on nights: S2 = {n2}
* p3 can't cook on nights: S3 = {n3}

In this situation, we can construct the bipartite graph as follows:

1. p1 can be matched with nights: {n2, n3}
2. p2 can be matched with nights: {n1, n3}
3. p3 can be matched with nights: {n1, n2}

When we apply the maximum flow algorithm to the resulting flow network, we find the following matching:

* p1 cooks on night n2
* p2 cooks on night n1
* There is no available night for p3 to cook

In this case, the maximum flow value is 2, which is less than n (3). Consequently, not all 3 people can be matched with one of the 3 nights, and p3 would need to pay $200 to hire a cook.

This counterexample demonstrates that it is not always possible to match all n people with one of the n nights due to individual constraints.