Exercises 5

Good problems

DPV8.7

[similar to Assignment 5 q2]
We model this as a graph problem G, where each clause is a vertex, each variable is a vertex, and there is an edge between a clause and a variable if the variable is in that clause. This graph is bipartite, since edges only go between clauses and variables. Since each variable appears at most 3 times, every variable vertex has degree at most 3. Since every clause has exactly 3 literals, every clause vertex has degree 3. Let A be the set of clause vertices and B be the set of variable vertices.

We show that this graph has a matching of size |A|. By a modified Hall's Theorem, there is a matching of size |A| if and only if for every subset S in A, $|N(S)| \geq |S|$, where N(S) is the neighbour set of S in B.

Since each clause has exactly 3 literals and each variable appears at most 3 times, $|A| \leq |B|$. Since every vertex in A has 3 edges, S has 3|S| edges. Since every vertex in B has at most 3 edges, we must have that $|N(S)| \geq |S|$. By Ha''s Theorem, we must have that there is a matching of G of size |A|.

This means that for every clause, it is assigned to one unique variable in that clause. Then we can set each of the variables in the matching to true to satisfy the formula.

Finding a maximum matching in a bipartite graph takes O(|V||E|) time.

KT8.41

A) We can create a new undirected graph G'. For each vertex v in G, we create two vertices v_{in} and v_{out} in G'. For each edge (u, v) in G, we add an edge u_{out}, v_{in} in G'. G' is a bipartite graph, since edges only go between vertices in the 'out' partition and vertices in the 'in' partition. There is a cycle cover in G if and only if there is a perfect matching in G'. This is because if each node is part of exactly one cycle, each node has exactly one incoming edge and one outgoing edge. In G', this means every v_{in} and v_{out} intersect one edge each, ie a perfect matching. Maximum bipartite matching takes O(|V||E|) time with the Ford Fulkerson algorithm.

B) We first prove that this problem is in NP. Given a set of node-disjoint cycles, it takes linear time to iterate all of them and check that they are vertex disjoint and cover all vertices, as well as each have length at most 3.

We prove that this problem is a reduction of 3-dimensional matching (3DM). Given an instance of 3DM with disjoint sets X, Y, Z each of size n, and set T (subset of X x Y x Z), we construct an instance of our problem as follows. Our graph G has a vertex for each element in X U Y U Z. For each triple (x,y,z) in T, we add edges (x,y), (y,z), (z,x). For each element w covered by multiple triples, we create a new vertex corresponding to w for each triple. This means every vertex currently has exactly indegree 1 and outdegree 1, if it is part of at least one triple, otherwise degree 0. For each element w covered by multiple vertices, let W denote the set of vertices corresponding to that element. We add a set A of additional nodes of size |W| - 1, and add an edge from every vertex in W to every vertex in A, and an edge from every vertex in A to every edge in W.

If there is a solution to this instance of 3DM, we have that there is a subset t of T such that

every element in X U Y U Z is in exactly one triple. Each triple in T corresponds to a 3-cycle in G. Since t covers every element in X U Y U Z exactly once, this means this set of 3-cycles covers each vertex in G exactly once: ie it is vertex-disjoint. For each element w covered by multiple vertices, we have |W| - 1 uncovered vertices, as well as its corresponding set of vertices A, of size |W| - 1. Hence we add |W| - 1 disjoint 2-cycles for each of these elements w, covering all vertices. Hence we have a cycle-cover where each cycle has at most 3 edges.

If there is a solution to this instance of our problem, we have that there is a set of node-disjoint cycles of at most 3 edges. The vertices in the sets A can only be covered by 2-cycles, since no 3-cycle passes through them. The set of 2-cycles covering vertices in each set A leaves one node uncovered from each set A. Now there is one vertex per element in X U Y U Z. These must be connected by 3-cycles, since each of these vertices is part of a 3-cycle by construction. This is a 3D matching: each element is in exactly one triple.

It takes polynomial time to construct G, since we iterate every element in X U Y U Z once, and every triple once. Since 3DM is NP-complete, our problem must also be NP-complete.

CLRS26.2-11

Our algorithm works as follows. We replace every edge (u, v) in G with two directed edges (u, v) and (v, u), each with capacity 1 to form G'. We pick an arbitrary vertex s. For this vertex s, we iterate through every other vertex t. In each iteration, we model this as a max-flow problem, with source vertex s and sink vertex t. We then use the Ford Fulkerson algorithm to find the max-flow in G' from s to t. The minimum value of all these max-flows is the edge connectivity of G.

Runtime: It takes O(|E|) time to update the graph edges. We run max-flow at most |V| times, on a graph with |V| vertices and 2|E| edges. Thus our algorithm takes O($|V|^2$|E|) time.

Correctness: We prove that the minimum value of all these max-flows is the edge connectivity of G. Consider one flow network with some vertex sink vertex t. The value of the max flow in G' from s to t is the same as the value of the minimum cut separating s and t. Since each edge has capacity 1, the value of the cut is the same as the number of edges in G crossing the cut. Hence the max flow is the number of edges we need to remove so that s and t are in different components. We consider all possible values of t and take the minimum value, since for the graph to be disconnected, s must be in a different component from at least one vertex t. Hence our algorithm is correct.

CLRS26-4

A) We run one iteration of the Ford-Fulkerson algorithm. If there is an augmenting path, we update the flow accordingly, otherwise the max flow cannot be increased.

Runtime: One iteration of Ford-Fulkerson takes O(|V| + |E|) time. Hence our algorithm takes O(|V| + |E|) time.

Correctness: Case 1: edge (u, v) does not cross a minimum cut. Then increasing its capacity does not change the capacity of any minimum cut, so there is no augmenting path and max flow does not change. Case 2: edge (u, v) crosses a minimum cut. Then increasing its capacity by 1 increases the capacity of that minimum cut by 1. If there is an augmenting path, the max flow increases by 1. If there is no augmenting path, there is some other minimum cut (u, v) does not cross. Ford Fulkerson will find an augmenting path if there is one, so we just need one iteration for this.

B) If the flow through edge (u, v) was less than its old capacity, we do nothing. Otherwise, we proceed as follows. We find a path from s to t that contains (u, v) using BFS. Then we decrease

the flow of every edge on that path by 1. This decreases our total flow by 1. Then we run one iteration of Ford Fulkerson. If we find an augmenting path, we will increase the flow by 1, and our max flow does not change. Otherwise if there is no augmenting path, the max flow is decreased by 1.

Runtime: BFS takes O(|V| + |E|) time. One iteration of Ford-Fulkerson takes O(|V| + |E|) time. Hence our algorithm takes O(|V| + |E|) time.

Correctness: The idea is that edge (u, v) now has too little capacity to hold the amount of flow it originally was, so maintain this max flow, we have one unit of flow from u to v that we must redirect along a different path. If such a path exists, it will be found when we decrement the flow along the path and run Ford Fulkerson. Otherwise, there will be no augmenting path and we must decrease our max flow by 1.

DPV7.19

We first check that the flow leaving source vertex s is equal to the flow entering sink vertex t. We iterate through every other vertex and check that its incoming and outgoing flows are equal. We iterate through every edge and check that the flow through the edge is at most its capacity. Then we run one iteration of Ford Fulkerson on the graph. If there is an augmenting path, the flow was not maximal, since we have an augmenting path that will increase the max flow. If there does not exist an augmenting path, the flow is maximal.

Runtime: It takes O(|E|) time to iterate every edge, and O(|V|) time to iterate every vertex. One iteration of Ford-Fulkerson takes O(|V| + |E|) time. Hence our algorithm takes O(|V| + |E|) time.

KT7.12

If the minimum s-t cut has size $\leq$ k, then we can reduce the flow to 0 by disconnecting s and t. Otherwise, we can decrease the max flow by k. Our algorithm works as follows. We use the Ford-Fulkerson algorithm to find a minimum s-t cut. We delete k of the edges crossing this cut.

Runtime: It takes O(|V||E|) time to run the max-flow min-cut algorithm.

Correctness: Since every edge has capacity 1 and we removed k of these edges, this minimum cut now has capacity max-flow - k. Every other minimum cut has capacity at least max-flow - k. Hence we reduce the max-flow by k.

DPV8.10

A) [You can probably also reduce Hamiltonian path, by setting G to a path of length |V|, and H to the HP graph].
We prove this is a reduction of maximum clique. Given an instance of the max clique problem with graph G and clique size k, we construct an instance of our problem as follows. We set graph A to be the complete graph on k vertices, and B = G.

If there is a solution to this instance of max clique, there exists a clique of size k in G. Then the clique is a subgraph of G. A complete graph on k vertices is a clique of size k. Hence A is a subgraph of B.

If there is a solution to this instance of our problem, we have that the complete graph on k vertices is a subgraph of B = G. Then there is a clique of size k in G.

B) We prove this is a reduction of Hamiltonian path. Given an instance of HP with graph G,

there is a Hamiltonian path if and only if there is a simple path covering all vertices of G. Hence we can set g = |V|. A simple path of length g=|V| in G is a Hamiltonian path.

C) We prove this is a reduction of 3-SAT. Given an instance of 3-SAT with formula F with n variables and m clauses, we set the CNF formula to F and g = m. Then there is a solution to this instance of 3-SAT if and only if there is a truth assignment satisfying all the clauses, ie g=m clauses.

D) We prove this is a reduction of max clique. Given an instance of the max clique problem with graph G and clique size k, we construct an instance of our problem with graph G, a=k, b=n(n-1)/2. A clique of size k in G means the subset of vertices in the clique (a subset of a vertices) has b edges between them. If there is a subset of a=k vertices in G with b=n(n-1)/2 vertices between them, this means there is an edge between every pair of vertices which is a clique.

E) We prove this is a reduction of independent set. Given an instance of independent set with graph G and integer k, we construct an instance of our problem with graph G, a=k, b=0. There is an independent set of size k if and only if there is a set of vertices of size k=a with no edges between them.

F) We prove this is a reduction of vertex cover. Given an instance of vertex cover with graph G = (V, E) and integer j, we construct an instance of our problem as follows. U = E. For each vertex v in G, we create a subset consisting of the edges intersecting v. We set integer k = j.

If there is a solution to this instance of vertex cover, we have a set of at most k vertices which covers all edges. Since each vertex corresponds to a set in our problem, we have a set of at most k subsets which cover all edges, which is U.

If there is a solution to this instance of our problem, we have a set of at most k subsets whose union is U. Since each subset corresponds to a vertex, we have a set of k vertices that cover all edges.

G) We prove this is a reduction of Hamiltonian cycle. Given an instance of HC with graph G = (V, E), we construct an instance of our problem as follows. We set b = |V|, d_{ij} = 1 if (i, j) is an edge, otherwise 999, and r_{ij} = 2 for all (i, j).

If there is a Hamiltonian cycle in G, this cycle has a total edge cost of b, since there are |V| edges each with distance 1. There are two vertex-disjoint paths from every vertex to every other vertex since this is a cycle.

If there is a solution to this instance of our problem, we have that there is a graph satisfying a total edge cost of b, with two vertex-disjoint paths from every vertex to every other vertex. This is a cycle, and since the edge cost is |V| it must be a Hamiltonian cycle.

DPV8.11

We first prove that this problem is in NP. Given a dominating set, we can iterate every vertex and check those it is adjacent to, to ensure every vertex is in the set or adjacent to at least one member of D. This takes polynomial time.

We prove that this problem is a reduction of vertex cover. Given an instance of vertex cover with graph G and integer k, we construct an instance of our problem G' as follows. We add all vertices and edges of G to G'. For each edge (u, v) in G, we create a new vertex w_{uv} in G', and add edges (u, w_{uv}) and (w_{uv}, v) in G' (edge (u,v) is also in G'). Let I be the number of isolated

vertices in G. We set b = k + I.

If there is a solution to this instance of vertex cover, there is a vertex cover of size k in G. Let V' be this vertex cover. Let V_I be the set of isolated vertices in G' (which are the same as the isolated vertices in G). We show that $V' \cup V_I$ is a dominating set in G'. All the isolated vertices are covered, since they are in the dominating set. The vertices in V' cover all edges, so every vertex w_{uv} that we added is covered. Each of the non-isolated original vertices is incident to at least one edge in G, so it must either be in V'or adjacent to a vertex in V'. We have that $|V' \cup V_I| = |V'| + |V_I| = k + I = b$. Hence we have found a solution to our instance of dominating set.

If there is a solution to this instance of our problem, we have that there is a dominating set of size b in G'. All the isolated vertices must be in the dominating set. Let V' be the remaining k vertices. We modify V' as follows. If any of the vertices w_{uv} we added are in V', we replace it with u (could also choose v). Since w_{uv} is adjacent to only u and v, it dominates only itself, u, and v. By using u instead, we still dominate all 3 of these vertices. We do this for all w_{uv} in V'. Now V'uses only vertices in G, but is still a dominating set in G'. Suppose for sake of contradiction V' is not a vertex cover for G. Then there exists some edge uv that is not covered. But this means w_{uv} is not covered in the dominating set, since neither u or v are in V'. This is a contradiction, so V' must be a vertex cover for G.

It takes polynomial time to construct G', since we iterate every vertex and edge in G once. Since vertex cover is NP-complete, our problem must also be NP-complete.

KT8.9

We first prove that this problem is in NP. Given a set of at least k paths, it takes polynomial time to verify that they are vertex-disjoint.

We prove that this problem is a reduction of 3-dimensional matching (3DM). Given an instance of 3DM with disjoint sets X, Y, Z each of size n, and set T (subset of X x Y x Z), we construct an instance of our problem as follows. Our graph G has a vertex for each element in X U Y U Z. For each triple (x,y,z) in T, we add edges (x,y) and (y,z). We also add path (x,y,z) to our set of requests. We set k = n.

If there is a solution to this instance of 3DM, we have a subset t of T of n triples where every element of X U Y U Z is contained in exactly one triple. This means we have n triples that are disjoint from each other. Since each triple corresponds to a path, we have found n=k paths that are vertex-disjoint.

If there is a solution to this instance of our problem, we have a set of k=n paths in G that are vertex-disjoint. Each path is of length 2 and corresponds to a triple, by construction. Since they are vertex-disjoint, this means our triples must contain a different element from X, Y, and Z, forming a 3D matching of size n.

It takes polynomial time to construct G, since we iterate every element in X U Y U Z once, and every triple once. Since 3DM is NP-complete, our problem must also be NP-complete.

KT 8.14

We first prove that this problem is in NP. Given a set k jobs, we can check that none overlap in polynomial time.

We prove that this problem is a reduction of independent set. Given an instance of independent set with graph G = (V, E) and integer j, we construct an instance of our problem as follows. We label each edge with a unique number. An edge labeled with 1 thus corresponds to time interval 1-2, and an edge labeled with 2 corresponds to time interval 2-3, and so on. For each vertex v, we create a job with time intervals of all edges incident to it. We set k = j.

If there is a solution to this instance of independent set, we have an independent set of at least k in G. Since each vertex corresponds to a job, and each edge a distinct time interval, we have a set of k jobs each with a distinct set of time intervals.

If there is a solution to this instance of our problem, we have at least k jobs with distinct time intervals. Since each job corresponds to a vertex and its time intervals correspond to edges, we have a set of k vertices with no shared edges.

It takes polynomial time to construct our jobs, since we iterate every vertex and edge of G once. Since independent set is NP-complete, our problem must also be NP-complete.

KT 8.28

We first prove that this problem is in NP. Given an independent set, it takes polynomial time to verify that is size of at least k, and that there are no paths of length 1 or 2 between vertices in the set.

We prove that this problem is a reduction of independent set. Given an instance of independent set with graph G = (V, E) and integer j, we construct an instance G' of our problem as follows. We add all vertices of G to G'. For every edge (u, v) in G, we add a new vertex w_{uv}, and edges (u, w_{uv}) and (v, w_{uv}). Then we add edges between every pair of these new vertices. We set k = j.

If there is a solution to this instance of independent set, we have a set of vertices of size at least k that have no edges between them in G. Since all these vertices are at least distance 2 from each other in G, they must be at least distance 3 from each other in G' because of the new vertices we added. Thus this is a strongly independent set of size at least k in G'.

If there is a solution to this instance of our problem, we have a set of vertices of distance at least 3 from each other in G'. None of the vertices can be the ones we added, since all these vertices are within distance 2 of every vertex in the graph. Since all these vertices are at least distance 3 from each other in G', they must be at least distance 2 from each other in G. Then these nodes form an independent set of size at least k in G.

It takes polynomial time to construct G', since we iterate every vertex and edge in G once. Since independent set is NP-complete, our problem must also be NP-complete.

Interesting problems

CLRS34-4

A) Does there exist a schedule that completes all tasks with total profit at least k?

B) We first prove that this problem is in NP. Given a schedule, we can verify in polynomial time that none of the intervals overlap, every job is scheduled, and the profit is at least k.

We prove that this problem is a reduction of 0/1 knapsack. Given an instance of knapsack with capacity W, target value V, and items $1 \dots n$ with where item i has weight w_i and value v_i, we construct an instance of our problem as follows. For each item i, we create task a_i with profit v_i, time w_i, and deadline W. We set k = V.

If there is a solution to this instance of knapsack, we have a set of items with total value at least V. These items have total weight at most W. Then we can create a schedule where we schedule the tasks corresponding to these items consecutively starting from time 0. Since their times are equal to the item weights, we can schedule them with no gaps in the time interval [1, W]. The profit of these items is equal to the value of the items, so our profit is at least V. We schedule all other jobs consecutively starting at time W, but since this is after the deadline they make no profit. Hence we have a solution for our problem.

If there is a solution to this instance of our problem, we have a schedule for our tasks. Let S be the set of tasks completed by their deadline. The times of the tasks in S must sum to at most W, and their profits must sum to at least V. Then we have a set of their corresponding items whose weights sum to at most W and values total at least V.

It takes polynomial time to construct this instance, since we just iterate every item once. Since knapsack is NP-complete, our problem must also be NP-complete.

C) Use solution below.

D) We define our subproblems as follows. D(j, t) is the maximum profit we can obtain with tasks 1…j at time t. D(j, 0) = 0 for all j. We have the recurrence

D(j, t) = max{ D$(j-1, t-t_j) + p_j$ (if $t \leq d_j$), D$(j-1, t)$ }

At time t, we can either have used item j or not. Our result is D(n, T), where T is the sum of processing times.

Runtime: We have O(nT) subproblems, each of which takes O(1) time to compute. Hence our runtime is O(nT).

CLRS26-2

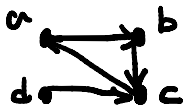
A) We define G' as suggested in the problem. x_0 is our source vertex, and y_0 is our sink vertex. We set all edge capacities to 1. We run max flow to find a maximum bipartite matching in G'. The edges selected in the matching constitute our path cover. We construct the actual paths as follows. We start a new path using a vertex i that has not yet been placed in a path. If x_i is unmatched, we cannot proceed any further so the path ends there. If x_i is matched to some y_j, we add j to the path (if j is the start of a different path, we combine the two). We continue until every vertex is part of a path.

Runtime: It takes O(|V| + |E|) time to construct G'. It takes O(|V||E|) time to run the max

flow. Hence our algorithm takes O(|V||E|) time.

Correctness: Every vertex is part of a path, since our algorithm continues until every vertex is part of a path. Each vertex appears in no more than one path, since the path edges are constructed from the matching, and in the matching every 'in' vertex (y) and every 'out' vertex (x) has at most one edge (ie at most one path edge enters each vertex and at most one path edge leaves each vertex). Suppose that the maximum flow algorithm yields a cover with k paths, and hence flow n−k, but a minimum path cover uses strictly fewer than k paths. Then it must use strictly more than n−k edges, so we can recover a flow which is larger than the one previously found, contradicting the fact that the previous flow was maximal. Thus, we find a minimum path cover.

B) No. The graph can be covered with a single path dcab, but our algorithm could output paths d, abc



Challenging problems

DPV8.23 A)