Problem 1

I made the assumption that the graph representation of this routing problem uses directed edges.

Part (a)

I devise an integer linear program capturing the routing problem. Let f_{ij}^k be the indicator variable for whether the path between the kth demand pair uses edge ij. Let $s(k)$ be the source vertex of the kth pair, and $t(k)$ be its target vertex. Let u_{ij} be the integer capacity of edge ij. The ILP is:

$$0 \le f_{ij}^k \le 1 \quad \forall k, ij$$

$$\sum_k f_{ij}^k \le u_{ij} \qquad \forall ij$$

$$\sum_j f_{ij}^k - f_{ji}^k = \left\{ \begin{array}{ll} 1 & \text{if } s(k) = i \\ -1 & \text{if } t(k) = i \\ 0 & \text{otherwise} \end{array} \right\} \qquad \forall k, i$$

Part (b)

I argue that the relaxion of this ILP can be seen as defining a collection of fractional paths between each demand pair, of total capacity 1, and that these paths can be read out of the solution in polynomial time.

For each demand pair k, there is a flow of 1 from $s(k)$ to $t(k)$. I can decompose this flow into a collection of fractional paths as follow. I create a "residual" graph G_k with the same vertices and edges as the problem graph, but where the initial capacity of edge ij is $r_{ij} = f_{ij}^k$. I repeat until no paths can be found in G_k:

- Find a path P from $s(k)$ to $t(k)$ in G_k.
- Saturate this path: for each edge ij, set $p_{ij} = \min_{ij' \in P} r_{ij'}$ if $ij \in P$ and set $p_{ij} = 0$ otherwise.
- Add the path P as defined by the p_{ij}'s to the collection of fractional paths.
- Update the capacities on the edges of G_k: $r_{ij} \leftarrow r_{ij} - p_{ij} (\ge 0)$.

Now, the residual graph G_k has the invariant:

$$\sum_j r_{ij} - r_{ji} = 0 \quad \forall i, \text{ s.t. } i \ne s(k) \text{ and } i \ne t(k)$$

$$\sum_j r_{s(k)j} - r_{js(k)} = -\sum_j r_{t(k)j} - r_{jt(k)}$$

Indeed, initially the invariant holds as the r_{ij}'s inherit it from the f_{ij}^k's which satisfy the ILP. Then, when a path of length l from $s(k)$ to $t(k)$, $p_{s(k)i_1} \rightarrow$

$p_{i_1 i_2} \rightarrow \ldots \rightarrow p_{i_l t(k)}$, is substracted from the capacities r_{ij}'s, the invariant is maintained. Indeed, for any vertex i in the path different from $s(k)$ and $t(k)$, the path value is substracted from some r_{ij} but also from some r_{ji} so that the net flow from i is still 0. Furthermore, the path value is substracted from $r_{s(k)i_1}$ but also from $r_{i_l t(k)}$ so that the net flow from $s(k)$ is still equal the opposite of the net flow from $t(k)$. When no path is found, it means that the net flow from $s(k)$ is 0, which means that all the fractional paths found sum to 1.

Therefore, this procedure indeed decomposes the flow of a demand pair into a collection of fractional paths of total capacity 1. This procedure runs in polynomial time. Indeed, each iteration removes an edge from the residual graph as it saturates the found path, so there can be at most m iterations (and m paths). Each path can be found in $O(m \log n)$ time using Dijkstra's algorithm with a binary heap and the other steps of an iteration take $O(m)$ time. Therefore, the procedure can find the collection of fractional paths of a demand pair in $O(m^2 \log n)$ time, which is polynomial in the size of the problem.

Part (c)

If each edge has capacity 1 and the path-finding problem has a feasible solution, I devise a randomized rounding scheme that gives an integer feasible solution where every edge carries $O(\log n)$ paths, in polynomial time.

By the procedure in part (b), I get a collection of fractional paths for each flow of a demand pair. Since the value of the flow of a demand pair is 1, the sum of the values of the fractional paths for a demand pair is 1. So I consider the values of the paths to be probabilities. For each flow, I pick one of the path out of its collection with a probability equal to its value. This is my rounding scheme.

I show that after trying my rounding scheme an expected constant number of times, I will find an integer solution in which each edge carries $\leq 1 + 2\log n = O(\log n)$ paths. Consider an arbitrary edge. Let X_k be the indicator variable denoting whether the path for the kth demand pair goes through this edge. I know that $\mathbb{E}\left[\sum_k X_k\right] = \sum_k \mathbb{P}\left[X_k = 1\right] \leq 1 = \mu$. Using the Chernoff bound for $\epsilon = 2\log n > 2e - 1$, $\mathbb{P}\left[\sum_k X_k \geq 1 + 2\log n\right] \geq 2^{-(1+2\log n)} = \frac{1}{2n^2}$. By the union bound, the probability that some edge carries $> 1 + 2\log n$ paths is $\leq \frac{1}{2}$.

Since each attempt is polynomial and after a constant number of attempts, I expect to find a solution in which every edge carries $O(\log n)$ paths, my $O(\log n)$-approximation algorithm is polynomial.

Part (d)

I generalize part (c) and argue that if a solution exists in which every edge carries only w paths, then in polynomial time a solution can be found in which edge of capactiy w carries only $w + O(\sqrt{w \log n})$ paths.

I use the same rounding scheme as before. Now, for an arbitrary edge, $\mathbb{E}\left[\sum_k X_k\right] = \sum_k \mathbb{P}\left[X_k = 1\right] \leq w = \mu$. Using the Chernoff bound for $\epsilon = 3\sqrt{\log n / w} < 2e - 1, \mathbb{P}\left[\sum_k X_k \geq (1+\epsilon)w\right] \leq e^{-(3\sqrt{\log n/w})^2 w/4} = e^{-9\log n/4} =$

$\frac{1}{n^{\frac{9}{4}}}$. By the union bound, the probability that some edge carries $> w(1+\epsilon)$ paths is $n^{2-\frac{9}{4}} = n^{-\frac{1}{4}} < 1$ for $n \geq 2$.

Thus, after an expected constant number of polynomial-time attempts, I'll find a solution in which each edge carries only $w + 3\sqrt{w \log n}$ paths. Assuming $w > \log n$, I have a 4-approximation.

Problem 2

Part (a)

If the answer to a set-basis instance is "yes", I can say that the number of sets in C is $\leq 2^k$, because there are 2^k subsets of B and each set $A \in C$ equals the union of some subset of B.

Part (b)

I show that if two elements x and y appear in precisely the same family of sets in C, removing y from all sets in C preserves the answer to the basis.

If the answer to the basis was "yes", it is clear that it is still "yes" after removing y from all sets in C. Indeed, we can use the same solution B after removing y from all sets in B.

If the answer to the basis was "no", then it is still "no" after removing y from C. If C had more than 2^k sets, it still has more than 2^k sets, so a solution B cannot exists. If C has less than 2^k sets, suppose, for sake of contradiction, that a solution B exists after the removing of y. Then, I can construct a solution to the original C by adding y to each set of B which has x. Contradiction.

Thus, the answer is preserved.

Part (c)

If C has more than 2^k sets, then I can immediately answer "no". So suppose C has $\leq 2^k$ sets. I can simplify the sets as in part (b) in polynomial time in the number of sets and the number of items. After simplification, each two items must disagree on at least one set to belong to, so there can be at most $2^{2^k} - 1$ items. So I have reduced the problem to a problem with at most 2^k sets and at most $2^{2^k} - 1$ items, so the problem is now a function of k. Thus, set basis is fixed-parameter tractable with respect to k.

Problem 3

Part (a)

I show that maximum satisfability is fixed-parameter tractable with respect to treewidth. I modify the algorithm presented in class for the decision version of SAT to keep track of the number of clauses satisfied (as opposed to just whether all clauses are satisfied) in the truth table for a set of merged clauses.

Initially, my algorithm has one truth table per clause. For each truth assignment in a truth table, the number of clauses satisfied is naturally 1 if the clause is satisfied with the given truth assignment and 0 otherwise. I eliminate variables using the elimination ordering on the graph where vertices are variables and edges represent shared clauses between variables. When I eliminate a variable, I merge all the clauses (or, recursively, set of clauses) in which it appears and create one truth table out of the smaller ones. I fill in each possible truth assignment (of the other variables) in the big table as follows: for the two possible truth assignments of the eliminated variable, I compute the sum of the number of satisfied clauses over all projections in the smaller tables, keeping the maximum sum and the corresponding best truth assignment of the eliminated variable for this truth assignment of the other variables. At the end, I'll have a few truth tables corresponding to independent sets of clauses. I can then simply choose the maximum number of clauses satisfied from each table and sum them to get the total maximum number of clauses satisfied. This algorithm works because it preserves the number of clauses satisfied as the variables are eliminated, in the same way that the original algorithm preserves the satisfiability of the formula. The order of its running time remains the same, so it's still polynomial for a fixed treewidth.

Part (b)

I show that vertex cover is fixed-parameter tractable with respect to the treewidth parameter by reducing vertex cover to maximum satisfiability.

Each vertex corresponds to a variable: the vertex is in the cover if its corresponding variable is true. Each edge corresponds to an or-clause of its endpoints. In order to minimize the number of variables that are true, for each variable x, I add the one-variable clause (not x).

Now, I can solve the corresponding maximum satisfiability problem. This works because note that if any edge is not covered by the returned solution, it means that there is another optimal solution in which the edge is covered, which corresponds to another optimal solution in which the corresponding clause is true. Indeed, simply set either variable of the or-clause to true: this won't change the number of the clauses satisfied because it will toggle the or-clause from false to true and toggle a variable negation clause from true to false. So if the solution returned by the problem is missing some edges, I can cover them by choosing any of its endpoints, indifferently.

So I'll have n variables and $n + m$ clauses in the corresponding maximum satisfiability problem. The treewidth graph of this maximum satisfiability problem is the same as the treewdith graph of the vertex cover problem, because the one-variable negation clauses don't add any edges (as they are not shared clauses). Therefore, by part (a), vertex cover is fixed-parameter tractable with respect to the treewidth parameter.

Problem 4

Part (a)

I give a 9-competitive deterministic algorithm for optimizing the total distance travelled up and downstream before I find a bridge.

Algorithm

I go upstream for 1 meter and back, then downstream for 2 meters and back, then upstream for 4 meters and back, then downstream for 8 meters and back, ..., then upstream for 2^k meters and back, then downstream for 2^{k+1} meters and back, and so on, until I find the bridge.

Analysis

Let d be the distance at which the bridge is located. Let $k = \lfloor \log_2 d \rfloor$. In the worst case, I will be going up and down until I go 2^{k+1} meters in one direction and back, and then a distance d in the other direction to finally find the bridge. In the worst case then, $d = 2^k + c$ where $0 < c < 2^k$ (if $c = 0$, I would find the bridge without needing to go the distance 2^{k+1} in the wrong direction). Notice that $k = \log_2(d - c)$. So the worst-case distance I'll have to travel before finding the bridge is:

$$\begin{aligned}
d' &= 2\left(\sum_{i=0}^{k+1} 2^i\right) + d \\
d' &= 2(2^{k+2} - 1) + d = 2(4 \cdot 2^k - 1) + d = 2(4(d-c) - 1) + d \\
d' &= 8d - 8c - 2 + d = 9d - 8c - 2 \\
d' &\leq 9d
\end{aligned}$$

Thus, the competitive ratio is 9.

Part (b)

I give a randomized 7-competitive algorithm for the problem.

Algorithm

The algorithm is similar to part (a), except that I flip a fair coin once to decide whether to go upstream or downstream initially.

Analysis

I have a probability $\frac{1}{2}$ of starting on the "worst" side and travelling the same distance d' as in part (a). I also have a probability $\frac{1}{2}$ of starting on the "best" side and travelling a smaller distance, saving on the $(k+1)$th round trip. Thus, using the randomized algorithm, the expected distance I'll travel before finding the bridge is:

$$\begin{aligned}
d'' &= \frac{1}{2}\left(2\left(\sum_{i=0}^{k+1} 2^i\right) + d\right) + \frac{1}{2}\left(2\left(\sum_{i=0}^{k} 2^i\right) + d\right) \\
d'' &= \left(\sum_{i=0}^{k+1} 2^i\right) + \left(\sum_{i=0}^{k} 2^i\right) + d \\
d'' &= 2\left(\sum_{i=0}^{k+1} 2^i\right) + d - 2^{k+1} \\
d'' &= d' - 2^{k+1} = d' - 2 \cdot 2^k = d' - 2(d-c) = d' - 2d - 2c \\
d'' &= 9d - 8c - 2 - 2d - 2c = 7d - 9c - 2 \\
d'' &\leq 7d
\end{aligned}$$

Thus, the expected competitive ratio is 7.

Problem 5

Part (a)

I show that any deterministic strategy for this problem is terrible from a competitive perspective. Indeed, for any deterministic strategy, I can contrive a setting which forces the strategy to choose the date of lowest rank, as follows. The first date has a score of 0. As long as the strategy decides to break up, the score of the ith date is $-i+1$. Once the strategy accepts a date, say the ith date, the remaining $k-i$ dates have scores set to $1, 2, \ldots, k-i$. This way, the deterministic strategy chooses the date with the worst score, i.e. the one with the lowest rank.

Part (b)

I devise a random strategy that gives me a constant probability of ending with the absolute best companion.

I divide the pool of k partners into two sets, A and B, each partner independently having probability $\frac{1}{2}$ of being in either set. I date each person in A and break up, remembering the score of the best partner seen in A. I proceed to dating the persons in B. If I find a date in B with a better score than the best score seen in A, I stay with them forever.

The probability that I end up with the absolute best companion, p_b, is $\geq$ than the probability that the second best person ended up in A and the best person ended up in B. Since, these two probabilities are independently $\frac{1}{2}$, $p_b \geq \frac{1}{4}$. So I have a constant probability of ending up with the absolute best companion.

Part (c)

I devise a randomized strategy in which the expected rank of my final choice is $O(\log k)$.

I divide the pool of k people into sets A and B like in part (b). I date all the people in A, ranking them. When I date the people in B, I decide to stay forever with the first person whose rank is better than the rank of the $(2\log k)$th person in A.

I consider 3 cases for k large enough.

1. Suppose there is no person in B that is better than the $(2\log k)$th best in A. This means that the best $2\log k$ persons are in A, so this case has probability ${\frac{1}{2}}^{2\log k} = \frac{1}{k^2}$. In this case, I might end up with the person of the lowest possible rank (k).

2. Suppose the $(2\log k)$th person in A has rank $> 12\log k$. This means that at most $2\log k$ of the best $12\log k$ people are in A. To upper-bound the probability of this case, I use the Chernoff bound. My indicator variable X_i denotes whether the ith best person belongs to A. Then, $\mathbb{P}\left[\sum_{i=1}^{12\log k} X_i \leq 2\log k \leq (1-\frac{2}{3})6\log k\right] \leq e^{-\frac{4}{3}\log k} \leq \frac{1}{k}$. In this case,

let's suppose I might end up with the person of the lowest possible rank (k) for sake of upper bounding the expectation.

3. If (1) and (2) don't hold, then I'll end up with a companion whose rank is $< 12 \log k$. I upper-bound the probability of this case by 1.

Putting it all together, the expected rank of my final choice, $\mathbb{E}\left[\text{rank}\right]$, is $\leq \frac{1}{k^2} \cdot k + \frac{1}{k} \cdot k + 1 \cdot 12 \log k = O(\log k)$.

Part (d)

People are in one of two dating phases: sample phase (set A) and commit phase (set B). During the sample phase, it might be hard to stick to the strategy and move on no matter what if you find someone you really like. The problem doesn't deal with the companions actually having strategies of their own, so I won't even go into the headaches and heartbreaks that can occur between partners in distinct dating phases.