CSCI 670: Advanced Analysis of Algorithms
4^{th} Assignment

Fall 2021

Suggested reading: Chapter 7 of Algorithm Design (Jon Kleinberg and Eva Tardos).

Problem 1. Given a bipartite graph, design a polynomial-time algorithm for finding a minimum vertex cover (i.e., a set $S \subseteq V(G)$ with smallest possible size such that each edge has at least one endpoint in S). Note that for general graphs finding minimum vertex cover is NP-Complete.

Solution 1. Let us denote our bipartite graph with $G = (X, Y, E)$. We will prove that the size of the maximum matching of G is equal to the size of the minimum vertex cover, and give an algorithm to construct a minimum vertex cover from a maximum matching.

Let M be a set of edges corresponding to a maximum matching. As each edge can be covered by one of its endpoints and the endpoints of edges of M are disjoint, any vertex cover needs at least $|M|$ edges. Therefore, if we construct a vertex cover of size $|M|$ it will be a minimum vertex cover. We will do this by selecting exactly one endpoint from each edge of M

Let $L = X \backslash V(M)$ (unmatched nodes of X) and $R = Y \backslash V(M)$ (unmatched nodes of Y). Also, let L' be the set of that are reachable from L using alternating paths, and R' be the set of nodes that are reachable from R using alternating paths (i.e., paths where $e_1 \notin M, e_2 \in M, e_3 \notin M, e_4 \in M$, and so on). We take all vertices of M on the left side that are in R', all vertices of M on the right side that are in L' and the left endpoints of edges of M that are not reachable from both L and R using alternating paths. This can be written as $C = (X \cap V(M) \cap R') \cup (Y \cap V(M) \cap L') \cup ((V(M) \cap X) \backslash (L' \cup R'))$. This way we select at least one endpoint from each edge of M. In fact, we never select both endpoints of an edge of M. To see this, consider an edge $(u, v) \in M$. If we take both u and v, it means u is reachable from R and v is reachable from L, which in turn means there is an alternating path connecting an unmatched node of X to an unmatched node of Y, which contradicts with the fact that M is a maximum matching. This proves that $|C| = |M|$.

Now, let us see that C is a vertex cover. Edges of M are covered because we select at least one endpoint of each edge of M. Edges (u, v) with $u \in L$ and $v \in V(M)$ are covered because $v \in C$. Similarly, edges (u, v) with $u \in V(M)$ and $v \in R$ are covered because $u \in C$. As M is a maximum matching there are no edges (u, v) with $u \in L$ and $v \in R$. Therefore, it remains to see that all edges (u, v) with $(u, v) \notin M, u \in V(M), v \in V(M)$ are covered. If $u \in R'$, then $u \in C$. Otherwise, let v' be the pair of u in M. If $v' \in L'$, then we get that $v \in L'$, implying that $v \in C$. Finally, if $v' \notin L'$, then $(u, v') \in M$ is not reachable from both L and R using alternating paths. Therefore, by the definition of C (see the 3 rd union term of C), we would select u.

Problem 2. Problem 7.27 from the textbook.

Solution 2. First, we will use network flow to give polynomial time algorithm for finding a fair driving schedule. Then, we will prove that such a schedule always exists. (b) We create a graph $G = (V, E)$, where we have one vertex for each person $p_i (i = 1.k)$, one vertex for

each day $q_j (j = 1..d)$, source s, and sink t. For each person p_i we add a directed edge from s to p_i with capacity $\lceil \Delta_i \rceil$. If $p_i \in S_j$, we add a directed edge from p_i to q_j with capacity ∞. Finally, for each day q_j we add a directed edge from q_j to t with capacity 1. Then we use Edmonds-Karp algorithm to find an integer maximum flow in our graph. If the value of flow is d (which we will show soon), then by looking at which edges of form (p_i, q_j) are used in our flow, we can find a fair driving schedule (assigning person p_i to drive on day j if the edge (p_i, q_j) has non-zero flow). Conversely, if there is a fair driving schedule we can build a flow of value d. That flow is a composition of d paths, one for each day. For day j, the path is $s \to p_{i_j} \to q_j \to t$, where p_{i_j} is the person assigned to drive on day j.

(a) Now we want to prove that for any sequence of sets $S_1, S_2, \ldots, S_d$, there exists a fair driving schedule. We already proved that we only need to show that there exists a flow in G of size d. We will show that the capacity of any $s - t$ cut in G is at least d. This will prove that $\min - \text{cut}(G) \geq d$. Combining it with the fact that min-cut (G) = max-flow (G) and max-flow $(G) \leq d$, we will get that max-flow $(G) = d$. Let $(S, \bar{S})$ be $s - t$ cut in G. W.l.o.g let's assume that $p_1, p_2, \ldots, p_r \in \bar{S}$ and $p_{r+1}, p_{r+2}, \ldots, p_k \in S$. If the cut cuts any edge that has infinite capacity we are done. Otherwise, it cuts edges $(s, p_i)\,(i = 1..r)$, and edges (q_j, t) if there is at least one driver $p_i (i = r + 1..n)$ is present in day j. Therefore,

$$c(S, \bar{S}) = \sum_{i=1}^{r} \lceil \Delta_i \rceil + a$$

where a is the number of days where at least one from $p_{r+1}, p_{r+2}, \ldots, p_k$ is present. We want to show that $c(S, \bar{S}) = \sum_{i=1}^{r} [\Delta_i] + a \geq d$. It is equivalent to show that $\sum_{i=1}^{r} \lceil \Delta_i \rceil \geq b$, where b is the number of days where no one from $p_{r+1}, p_{r+2}, \ldots, p_k$ is present (or equivalently the number of days for which $S_j \subseteq \{1, 2, \ldots, r\}$). Let those days be $j_1, j_2, \ldots, j_b$

$$\begin{aligned}
\sum_{i=1}^{r} \lceil \Delta_i \rceil &\geq \sum_{i=1}^{r} \Delta_i \\
&= \sum_{i=1}^{r} \sum_{j : p_i \in S_j} \frac{1}{|S_j|} \\
&\geq \sum_{i=1}^{r} \sum_{j : p_i \in S_j, j \in \{j_1, \ldots, j_b\}} \frac{1}{|S_j|} \\
&= \sum_{t=1}^{b} \sum_{i : 1 \leq i \leq r, p_i \in S_{j_t}} \frac{1}{|S_j|} \quad \text{(because } S_{j_t} \subseteq \{1, \ldots, r\}\text{)} \\
&= \sum_{t=1}^{b} 1 \\
&= b
\end{aligned}$$

Problem 3. Problem 7.31 from the textbook.

Solution 3. First, checking whether a box can be placed in another box can be done in constant time. Since we want to minimize the number of visible boxes, we can maximize the number of boxes that are placed inside another one. As no two boxes can be placed directly inside another one, for each box there is one or zero box(es) directly inside it. This leads us to reduce our problem to finding a maximum matching. Let n be the number of boxes and let boxes be indexed from 1 to n. We build a bipartite graph $G = (X, Y, E)$, where $X = \{x_1, x_2, \ldots, x_n\}, Y = \{y_1, y_2, \ldots, y_n\}$, and $E = \{(x_i, y_j) \mid$ if box i can be placed inside box $j\}$

Proposition. Let a be the minimum number of visible nodes and m be the size of maximum matching in G. We claim that $a = n - m$.

Proof. Suppose we are given a matching of size m. Whenever that matching contains an edge (x_i, y_j) we place box i directly inside box j. Notice, for every box we will place at most one box inside it. So, our arrangement is valid. The number of visible nodes in this arrangement will be the number of nodes of X that are unmatched. Therefore, $a \leq n - m$. Conversely, suppose we are given an arrangement with a visible boxes. Let $M = \{(x_i, y_j) \mid \text{if box } i \text{ is inside box } j\}$. As no two boxes can be directly inside another box, M is a matching. If there are n boxes and a of them are visible, $|M| = n - a$. Therefore, $m \geq n - a$. Combining with $a \leq n - m$, we get $a = n - m$.

Finding maximum matching in a bipartite graph can be done in polynomial time using Kuhn's or Edmonds-Karp algorithm. Finding the actual box arrangement can be done as described in the proof of the proposition.

Problem 4. Problem 7.39 from the textbook.

Solution 4. (a) Let R_i denote the sum of numbers in i-th row and C_j denote the sum of numbers in j-th column. Let $T = \sum_i R_i = \sum_j C_j$. First, we want to keep only the values that are fractional. If the value of cell (i, j) is α_{ij} and is integer, we subtract α_{ij} from R_i and C_j. We end up having an equivalent problem of finding a desired rounding (but now we don't take into consideration numbers that are already integers). We construct a graph $G = (V, E)$, where we have one vertex for each row r_i, one vertex for each column c_j, sink s, and source t. For each row r_i, we add a directed edge from s to r_i with capacity R_i. For each column c_j, we add a directed edge from c_j to t with capacity C_j. Also, for each pair of r_i and c_j, add an edge from r_i to c_j with capacity 1 if the number of cell (i, j) is fractional. We want to prove that the desired rounding exists if and only if there is an integer flow [1] of value T. Obviously, that will be a maximum flow. First, let's assume there is an integer flow of size T. If the edge (r_i, c_j) is used in the flow, we round the number of cell (i, j) to 1, otherwise we round it to 0. In row i there will be R_i numbers that will be rounded to 1 and in column j there will be C_j numbers that will be rounded to 1 (because of the flow conservation constraint). This proves that we have a desired rounding. Conversely, if we are given a desired rounding, we can specify a flow in G that has value of T. That flow is a composition of T paths, each corresponding to a number that was rounded to 1. If cell (i, j) was rounded to 1, we add the following path into our flow: $s \to r_i \to c_j \to t$. Since row and column sums are respected, we won't violate capacity constraints.

(b) Suppose we add an integer δ to the value of cell (i_0, j_0). We want to prove that a desired rounding exists in the new table if and only if a desired rounding exits in the original table. We can prove only one direction. Take any desired rounding of the original table. Adding δ to the value of rounded value of cell (i_0, j_0), keeping rounded values of other cells the same, we get a desired rounding for the new table. Now suppose the value of cell (i, j) is α_{ij}. For each cell (i, j) we subtract $\lfloor \alpha \rfloor$ from the value of that cell and we get a new table where each value is between 0 and 1 (endpoints included). Now we can use the solution of part (a) to solve the problem. Note, in part (a) we assumed that values 0 and 1 are possible. Therefore, our reduction is correct even if some α_{ij} are integers.

(c) It is enough to prove that the desired rounding exists for the case of part (a). In the graph we constructed in the part (a) there is a fractional flow with total value of T (it will also be a maximum flow). That flow is easy to describe $f_{s,r_i} = R_i, f_{c_j,t} = C_j, f_{r_i,c_j} = \alpha_{ij}$ if $\alpha_{i,j} \notin \mathbb{Z}$, otherwise $f_{r_i,c_j} = 0$. As all capacities in G are integers, existence of a fractional maximum flow implies existence of integral maximum flow, which in turn implies existence of desired rounding.

Problem 5. Suppose that you have two companies, each trying to sell their own copy of n different products. For each product $i = 1, \ldots, n$, we know the following: 1. the price $p_i \geq 0$ that company 1 charges and the price $p'_i \geq 0$ that company 2 charges, 2. the quality $q_i \geq 0$

of company 1's version, and the quality $q'_i \geq 0$ of company 2 's version. A user wants to buy exactly one copy of each of the n products. The goal is to maximize total quality minus total price.

In principle, the user can combine the products of the two companies. But there's a catch: the products of different companies may not be perfectly compatible. For each pair (i, j), we have a penalty term $\pi_{i,j} \geq 0$, the loss in utility from combining company 1's version of product i with company 2 's version of product j. Notice that it is possible that $\pi_{i,j} \neq \pi_{j,i}$

We assume that products are always compatible internally, i.e., there's no penalty for combining two products from the same company. All pairwise penalty terms are subtracted from the a priori utility. Give a polynomial-time algorithm for computing which components to purchase from which of the two companies to maximize the total utility (including penalties) minus the total price.

Solution 5. Let $T(i) \in \{A, B\}$, denote which company are we buying the product i from. The objective is then the following:

$$\begin{aligned} O(T) &= \sum_{i:T(i)=A} (q_i - p_i) + \sum_{i:T(i)=B} (q'_i - p'_i) - \sum_{i,j:T(i)=A,T(j)=B} \pi_{ij} \\ &= \sum_{i=1}^{n} (q_i + q'_i) - \sum_{i:T(i)=A} (p_i + q'_i) - \sum_{i:T(i)=B} (p'_i + q_i) - \sum_{i,j:T(i)=A,T(j)=B} \pi_{ij} \end{aligned}$$

Maximizing $O(T)$ is equivalent to minimizing

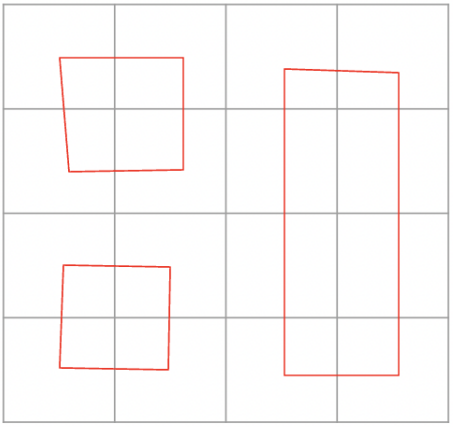
$$\tilde{O}(T) = \sum_{i:T(i)=A} (p_i + q'_i) + \sum_{i:T(i)=B} (p'_i + q_i) + \sum_{i,j:T(i)=A,T(j)=B} \pi_{ij}$$

Defining $b_i = p_i + q'_i \geq 0, a_i = p'_i + q_i \geq 0$ we get:

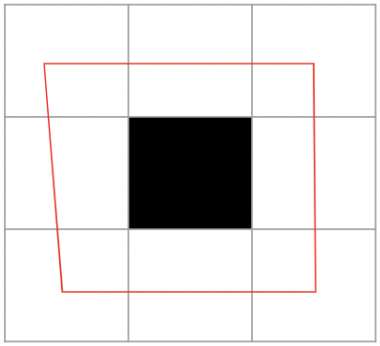
$$\tilde{O}(T) = \sum_{i:T(i)=A} b_i + \sum_{i:T(i)=B} a_i + \sum_{i,j:T(i)=A,T(j)=B} \pi_{ij}$$

We build a graph where we have a node u_i for product i (n nodes in total), source s and sink t. For each product i, we add an edge from s to u_i with capacity a_i and add edge from u_i to t with capacity b_i. For each pair $(i, j)(i = 1..n, j = 1..n, i \neq j)$, we add an edge from u_i to u_j with capacity π_{ij}. Given any $s - t$ cut $(S, \bar{S})$ we define $T(i) = A$ if $i \in S$ and $T(i) = B$ otherwise. The capacity of the cut $(S, \bar{S})$ is equal to the cost $\tilde{O}(T)$. Conversely, given $T(i)$, we construct an $s - t$ cut $(S, \bar{S}), S = \{s\} \cup \{i \mid T(i) = A\}$. Again, the capacity of the cut is equal to $\tilde{O}(T)$. This proves that any optimal $T(i)$ corresponds to a minimum s - t cut. Min-cut can be computed in polynomial time using Edmonds-Karp algorithm.

Problem 6. Consider an $N \times N$ grid, we say that two cells are adjacent if they share an edge. So, each cell has at most 4 neighbors. A cycle on a grid is a sequence of cells $c_1, \ldots c_k$ such that c_i and c_{i+1} is adjacent where assume that $c_0 = c_k$. We say that a grid can be decomposed into cycles if disjoint union of cycles covers covers all cells. An example of cycle decomposition of a 4×4 grid can be seen below.



One can prove that any $N \times N$ grid can be decomposed into cycles if and only if N is even (how?). In this problem we assume that a subset of cells are already occupied. The question is can we decompose remaining cells into cycles? The following figure is an example for cycle decomposition of 3×3 grid when a cell is occupied.



Assume that grid is denoted by $0-1$ square matrix A of size N^2 such that $A_{i,j} = 1$ if and only if cell i, j is occupied. Desing an algorithm to whether the given configuration of grid can be decomposed into cycles or not? If it is possible, can we find cycle decomposition as well?

Solution 6. We can consider the grid as a graph where unoccupied cells are vertices and two cells are connected by an edge if they are share a line segment in the grid. Let $G = (V, E)$ be the graph constructed by using the grid. We say $v \in e$ if an edge incident to a vertex v, and $u \sim v$ if $(u, v) \in E$.

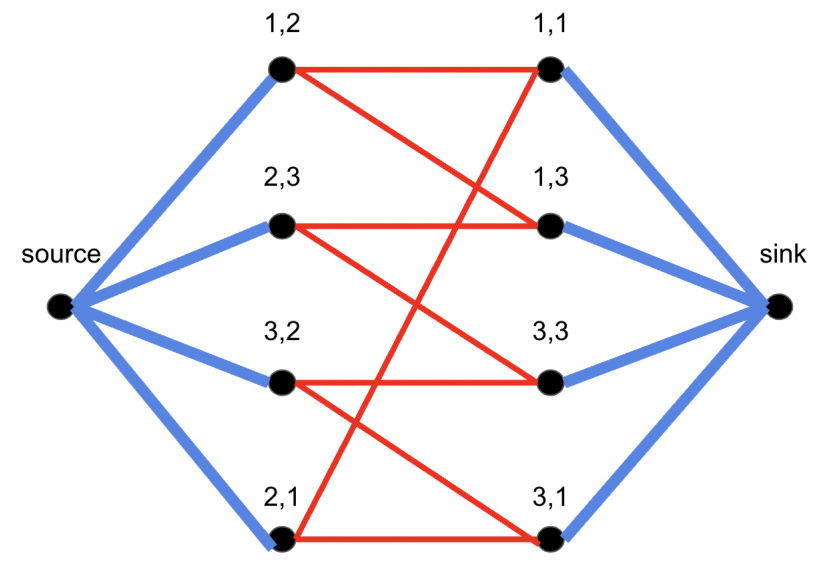
Lemma 1. Let E' be a subset of edges E such that each vertex of V is adjacent to exactly 2 edges of E'. Then E' is a cycle decomposition of G.

How to prove the lemma? By the lemma, it is enough to find such a subset of edges E'. Notice that lemma works for any graphs.

Now, we will use the fact that graph is constructed based on a grid. Let's call cells (vertices) i, j are odd vertices if $i + j \mod 2 \equiv 1$ and even vertices if $i + j \mod 2 \equiv 0$. Observe that edges only occur between vertices of odd vertices and even verticess, call them V_O and V_E respectively.

Now, we can formulate the cycle decomposition problem as a flow problem as follows. Let s be the source and t be the sink. We define a new graph $G' = (V', E')$ where $V' = V \cup \{s, t\}$,

$E' = E \cup E_O \cup E_E$ where $E_O = \{(s, v) \mid v \in V_O\}$ and $E_E = \{(u, t) \mid u \in V_E\}$. The capacity of each edge is defined as follows $c_e = 2$ if $e \in E_O \cup E_E$ and $c_e = 1$ if $e \in E$. The following figure domonstrates the network defined for the 3×3 grid given in the problem statement. Here the red edges have capacity 1, and blue edges have capacity 2.



Consider the maximum flow on this network. Since edge capacitied are integer, WLOG we can assume that solution would be integral. If the maximum flow uses 2-unit (blue) edges of each vertex with full capacity, then we guarantee that used 1-unit (red) edges consists of a cycle decomposition by Lemma 1. So, it can be computed in $poly(N)$ time with Ford-Fulkerson algorithm.