CS 170 HW 11

Due on 2019-15-04, at 9:59 pm

1 Study Group

List the names and SIDs of the members in your study group.

2 Bipartite Vertex Cover

A *vertex cover* of an undirected graph $G = (V, E)$ is a subset of vertices which touches every edge - that is, a subset $S \subset V$ such that for each edge $(u, v) \in E$, one or both of u or v are in S. Design an efficient algorithm to find the minimum sized vertex cover in a *bipartite* graph. (*Hint:* Use the max-flow-min-cut theorem on the network obtained for the maximum bipartite matching problem.)

Solution: Let L, R be a decomposition of the vertex set of the graph, G, such that for every edge in the graph, one of its end points is in L and the other is in R. We now construct the max-flow instance corresponding to the maximum bipartite matching problem for this graph, that is, we construct a new graph, G' with vertex set $V' = V \cup \{s\} \cup \{t\}$ and edge set $E' = E \cup \{(s, u) : u \in L\} \cup \{(v, t) : v \in R\}$. As before, each edge in G' is given capacity 1 and a solution the resulting max-flow problem can be used to construct a maximum bipartite matching for the graph G. We will now show how to construct a vertex cover for G from G'.

Let the max-flow-min-cut value for G' be k. Consider the minimum cut in G' with one partition containing s and the other containing t. Let P and Q be the two partitions of V' corresponding to the minimum cut of G' such that P contains s and Q contains t. Let $L_P = L \cap P$ and L_Q, R_P, R_Q defined analogously. Let $L'_P = \{u : u \in L_P \text{ and } u \text{ has a neighbour in } R_Q\}$. Consider the set of vertices $S = L_Q \cup R_P \cup L'_P$. Notice first of all that S is a vertex cover for G. We will now show that the size of S is at most k, the size of the cut. This follows from the following fact:

$$\begin{aligned} k &\geq |\{\text{Edges from } s \text{ to } L_Q\}| + |\{\text{Edges from } t \text{ to } R_P\}| + |\{\text{Edges between } L_P \text{ and } R_Q\}| \\ &\geq |L_Q| + |R_P| + |L'_P| = |S| \end{aligned}$$

To show that the set of vertices is the minimum vertex cover, note that k also corresponds to the size of the largest bipartite matching in the graph, G. For each edge in the matching, note that we have to pick at least one of the end points as part a vertex cover. We may conclude the size of any vertex cover is at least k. Therefore, the vertex cover, S is of minimum size.

3 Direct Bipartite Matching

Let $G = (L \cup R, E)$ be a bipartite graph such that for every edge, one of the endpoints is in L and the other is in R. Let M be a matching of G. A vertex, v, is said to be covered by M if one of its edges is in the matching, M. An *alternating path* is a path of odd length that starts and ends with a non-covered vertex, and whose edges alternate between M and $E \setminus M$.

(a) Prove that a matching, M, is a maximum matching if and only if there does not exist an alternating path with respect to it.

(b) Design an algorithm to find such an alternating path if it exists in time $O(|V|+|E|)$ time using a variant of breath first search.

(c) Give a direct $O(|V| \cdot |E|)$ time algorithm for finding a maximum matching in a bipartite graph.

Solution:

(a) We will first prove that if M is maximum, there does not exist an alternating path with respect to it. Suppose this were not true, let M be a matching and $p = u_1 - u_2 - \cdots - u_{2k}$ be an alternating path with $2k-1$ vertices and u_1 and u_{2k} are not covered. Note that we can increase the size of the matching by removing all the edges of the form (u_{2i}, u_{2i+1}) in the path p from the matching M and add the edges (u_{2i-1}, u_{2i}) to get a strictly larger matching.

For the other direction, suppose that M is a matching such that M has no alternating path with respect to it. For the sake of contradiction, suppose that M' is a matching of strictly larger size than M. Consider the subgraph of G, H, formed by the edges $T = M \cup M'$. Note that each vertex in H has degree at most 2 and furthermore, each vertex with degree 2 has one edge in M and the other edge in M'. Therefore, each connected component of H is either a line-graph, that is a subgraph of the form of the form $u_1 - u_2 - \cdots - u_k$ or a cycle of the form $u_1 - u_2 - \cdots - u_k - u_1$. Note that each cycle has an equal number of edges from M and M'. Since M' has strictly more edges than M, this means that there exists a line $u_1 - u_2 - \cdots - u_k$ which contains more edges from M' than M. This can only happen when k is of the form $2l$ and the line has $2l-1$ edges where l of the edges are from M' and $l-1$ from M. Therefore, the first and last edges in this line are from M'. This implies that the line $u_1 - u_2 - \cdots - u_k$ forms an alternating path for M contradicting our assumption that M is maximum.

(b) An algorithm to compute an alternating path is explained below:

Correctness:

We first prove that in the event that the algorithm returns a path, p, it indeed corresponds to an alternating path. Let $p = (u_1, \ldots, u_k)$ denote the path returned by the algorithm. Since, the algorithm explored the vertex u_1, u_1 belongs to L. Also, note that for every pair $(v, matched)$ inserted into the queue, $matched$ is $True$ if and only if $v \in R$. Therefore, we can conclude from that vertices of the form u_{2i-1} belong to L and u_{2i} belong to R. Furthermore, u_k belongs to R because the value of matched when u_k is popped from the queue is $True$. Furthermore, note that edges of the form (u_{2i-1}, u_{2i}) belong to $E \setminus M$ and those of the form (u_{2i}, u_{2i+1}) belong to M. We conclude that the the path p is an alternating path as the edges in the path alternate between those in M and not in M and the first and last edges do not belong to M. Therefore, p is an alternating path.

Next, we show that if $p = (u_1, \ldots, u_k)$ is an alternating path in G with respect to M, our algorithm will correctly return an alternating path. Suppose, without loss of generality that $u_1 \in L$ and $u_k \in R$. First, note that by an inductive argument similar to the

Algorithm 1 Find Alternating Path

Input: Bipartite Graph $G = (L, R, E)$, Partial Matching M
Output: Alternating path M if it exists
for $v \in L \cup R$ **do**
    $Explored[v] = False$
    $Parent[v] = null$
$Queue \leftarrow \phi$
$Found \leftarrow False$
$Start \leftarrow null$
for $v \in L$ **do**
    if $\neg Explored[v]$ **then**
        $Queue \leftarrow (v, False)$
        $Explored[v] \leftarrow True$
        while $\neg Queue$ **do**
            $(u, matched) \leftarrow Queue.pop()$
            if $matched$ **then**
                if $(u, t) \in M \wedge \neg Explored[t]$ **then**
                    $Queue.push((t, False))$
                    $Parent[t] \leftarrow u$
                    $Explored[t] \leftarrow True$
                else if $\nexists t : (u, t) \in M$ **then**
                    $Found \leftarrow True$
                    $Start \leftarrow u$
            else
                for $(u, t) \in (E \setminus M) \wedge \neg Explored[t]$ **do**
                    $Queue.push((t, True))$
                    $Parent[t] \leftarrow u$
                    $Explored[t] \leftarrow True$
if $Found$ **then**
    $path \leftarrow \phi$
    $curnode \leftarrow Start$
    while $Parent[curnode] \neq null$ **do**
        $path.append(Parent[curnode])$
    Return: $path$
else
    Return: $False$

correctness of BFS, u_k is explored in the algorithm. From the previous inductive argument from the previous, the value of *matched* when u_k was explored is $True$. Therefore, the variable $Found$ is set to $True$ when u_k is explored and the algorithm returns a path which is guaranteed to be an alternating path from the previous argument.

Runtime:

The runtime of the algorithm is $O(|V| + |E|)$ as each vertex and edge are explored at most twice in the running of the algorithm.

(c) A direct algorithm for maximum bipartite matching is described below:

Algorithm 2 Direct Bipartite Matching

Input: Graph $G = (L, R, E)$
Output: Matching M
$M \leftarrow \phi$
$p \leftarrow$ Find Alternating Path(G, M)
while $p \neq \phi$ **do**
  Let $p = (u_1, \ldots, u_k)$
  $M = M \setminus \{(u_{2l}, u_{2l+1})\}$ for $l \in \{1, \ldots, k/2 - 1\}$
  $M = M \cup \{(u_{2l-1}, u_{2l})\}$ for $l \in \{1, \ldots, k/2\}$
  $p \leftarrow$ Find Alternating Path(G, M)
Return: M

Correctness:

The correctness of the algorithm follows from the fact that when the algorithm terminates, it terminates where there is no alternating path for the matching M that is returned. Furthermore, the set of edges removed and added to the partial matching M imply that M remains a matching after this operation. Therefore, the set of edges returned, M, is a maximum matching.

Runtime Analysis:

Note that in each iteration, the number of edges in the matching increases by 1. Therefore, the algorithm Find Alternating Path is run at most $O(|V|)$ times. This implies that the total runtime of the algorithm is at most $O(|V| \cdot |E|)$ from our runtime bounds from the previous part.

4 Zero-Sum Battle

Two Pokemon trainers are about to engage in battle! Each trainer has 3 Pokemon, each of a single, unique type. They each must choose which Pokemon to send out first. Of course each trainer's advantage in battle depends not only on their own Pokemon, but on which Pokemon their opponent sends out.

The table below indicates the competitive advantage (payoff) Trainer A would gain (and Trainer B would lose). For example, if Trainer B chooses the fire Pokemon and Trainer A chooses the rock Pokemon, Trainer A would have payoff 2.

		Trainer B: ice	water	fire
Trainer A:	dragon	-10	3	3
	steel	4	-1	-3
	rock	6	-9	2

Feel free to use an online LP solver to solve your LPs in this problem.
Here is an example of a Python LP Solver and its Tutorial.

1. Write an LP to find the optimal strategy for Trainer A. What is the optimal strategy and expected payoff?

2. Now do the same for Trainer B. What is the optimal strategy and expected payoff?

Solution:

1. d = probability that A picks the dragon type
   s = probability that A picks the steel type
   r = probability that A picks the rock type

$$\begin{aligned} \max \quad & z & \\ -10d + 4s + 6r &\geq z & \text{(B chooses ice)} \\ 3d - s - 9r &\geq z & \text{(B chooses water)} \\ 3d - 3s + 2r &\geq z & \text{(B chooses fire)} \\ d + s + r &= 1 & \\ d, s, r &\geq 0 & \end{aligned}$$

   The optimal strategy is $d = 0.3346,\ s = 0.5630,\ r = 0.1024$ for an optimal payoff of -0.48.

2. i = probability that B picks the ice type
   w = probability that B picks the water type
   f = probability that B picks the fire type

$$\begin{aligned} \min \quad & z & \\ -10i + 3w + 3f &\leq z & \text{(A chooses dragon)} \\ 4i - w - 3f &\leq z & \text{(A chooses steel)} \\ 6i - 9w + 2f &\leq z & \text{(A chooses rock)} \\ i + w + f &= 1 & \\ i, g, f &\geq 0 & \end{aligned}$$

   B's optimal strategy is $i = 0.2677,\ w = 0.3228,\ f = 0.4094$. The value for this is -0.48, which is the payoff for A. The payoff for B is 0.48, since the game is zero-sum.

   (Note for grading: Equivalent LPs are of course fine. It is fine for part (b) to maximize B's payoff instead of minimizing A's. For the strategies, fractions or decimals close to the solutions are fine, as long as the LP is correct.)

5 Domination

In this problem, we explore a concept called *dominated strategies*. Consider a zero-sum game with the following payoff matrix for the row player:

		Column: A	B	C
Row:	D	1	2	-3
	E	3	2	-2
	F	-1	-2	2

(a) If the row player plays optimally, can you find the probability that they pick D without directly solving for the optimal strategy? (Hint: Notice that the payoff for E is always greater than the payoff for D. When this happens, we say that E *dominates* D, i.e. D is a *dominated strategy*).

(b) Given the answer to part a, if the both players play optimally, what is the probability that the column player picks A?

(c) Given the answers to part a and b, what are both players' optimal strategies? (You might be able to figure this out without writing or solving any LP).

Solution:

(a) 0. Regardless of what option the column player chooses, the row player always gets a higher payoff picking E than D, so any strategy that involves a non-zero probability of picking E can be improved by instead picking D.

(b) 0. We know that the row player is never going to pick D, i.e. will always pick either E or F. But in this case, picking B is always better for the column player than picking A (A is only better if the row player picks D). That is, conditioned on the row player playing optimally, B dominates A.

(c) Based on the previous two parts, we only have to consider the probabilities the row player picks E or F and the column player picks B or C. Looking at the 2-by-2 submatrix corresponding to these options, it follows that the optimal strategy for the row player is to pick E and F with probability $1/2$, and similarly the column player should pick B, C with probability $1/2$.