CSCC73 A8

Saad Makrod

December 2022

Q1 Max Flow/Linear Programming

Part A: Note that from the input we can compute the number n_j of neighbours available to garden on week j, $1 \leq j \leq q$. Consider the following flow network corresponding to the given problem:

$$V = \{s,t\} \cup \{w_i : 1 \leq i \leq q\} \cup \{p_i : 1 \leq i \leq p\}$$
$$E = \{(s,p_i) : 1 \leq i \leq p\} \cup \{(w_i,t) : 1 \leq i \leq q\} \cup \{(p_j,w_i) : 1 \leq j \leq p \text{ and } 1 \leq i \leq q \text{ and } i \in W_j\}$$
$$c(e) = \begin{cases} k_i & e = (s,p_i), 1 \leq i \leq p \\ 1 & e = (p_j,w_i), 1 \leq j \leq p, 1 \leq i \leq q, i \in W_j \\ \lceil \frac{n_i}{5} \rceil & e = (w_i,t), 1 \leq i \leq q \end{cases}$$
$$\mathcal{F} = (G = (V,E), s, t, c)$$

In the graph G above w_i represents a weekend i and p_j represents neighbour j. The edges (p_j, w_i) are 1 if and only if neighbour p_j is driving on week i. The capacities on the edges are defined as follows: if the edge is from the source to a neighbour node the capacity is the amount of times a neighbour is available to drive, if the edge is from a neighbour to a week this edge only exists if the neighbour is available that week and is 1 if the neighbour drives that week, lastly the capacity for the week to sink edges represent the minimum number of drivers needed to transport all neighbours available that week. Consider the following algorithm:

```
GARDEN-ASSIGNMENT($p$, $q$, $W_i$, $k_i$) # $1 \leq i \leq p$
    $\mathcal{F}$ = flow network constructed by procedure above
    $f$ = FF-MAXFLOW($\mathcal{F}$)
    $E' = \{e \in E : e = (w_i,t), 1 \leq i \leq q\}$
    if $V(f) = \sum_{e \in E'} c(e)$ is false return None
    $A = \emptyset$
    for $i = 1$ to $q$
        $A[i] = \{j : f(p_j,w_i) = 1, 1 \leq j \leq p\}$
    return $A$
```

ALGORITHM CORRECTNESS: We want a max flow that is able to determine if the following is possible:

1. Just enough drivers assigned each weekend to carry the neighbours available to garden that weekend
2. Each of the neighbours assigned to drive on a weekend is available to garden that weekend
3. No neighbour i is assigned to drive more than k_i weekends

We will argue that this assignment is possible if the value of the max flow is equivalent to $\sum_{e \in E'} c(e)$ where $E' = \{e \in E : e = (w_i,t), 1 \leq i \leq q\}$. In other words, the flow of all edges into t are saturated.

The proof of property 1 will be done by contradiction. Suppose that the flow of all edges into t are not saturated and property one is satisfied. Note that by definition for each weekend j, n_j is the number of neighbours available to garden on week, $1 \leq j \leq q$. Thus by definition the minimum number of cars needed each week is $\lceil \frac{n_j}{5} \rceil$. Thus if the flow of all edges into t are not saturated then it is impossible for all possible neighbours to be taken which is a contradiction.

For property 2, this holds by construction of the flow network. In the assignment returned by the algorithm a neighbour is assigned to drive if and only if the flow on edge (p_j, w_i) is 1 (connecting neighbour j and week i, $1 \leq j \leq p$ and $1 \leq i \leq q$). Furthermore this edge only exists if the neighbour is available that week since by definition the edges were created as follows: $\{(p_j, w_i) : 1 \leq j \leq p \textit{ and } 1 \leq i \leq q \textit{ and } i \in W_j\}$.

Lastly property 3 is satisfied since the flow from s to each neighbour node is the value k_i. Since flows follow conservation and capacity constraints it is guaranteed that a neighbour will never drive more than k_i times.

The the flow network satisfies all properties and this assignment is possible if the value of the max flow is equivalent to $\sum_{e \in E'}$ where $E' = \{e \in E : e = (w_i, t), 1 \leq i \leq q\}$.

Thus the algorithm will perform correctly since it checks if the value of the flow is equal to the sum of the capacities of all edges going into t. If A is returned that at each index i we will have a set of drivers that are neighbours which can drive in week i.

RUNNING TIME: We will refer to the number of nodes as n and the number of edges as m. Note that $n = p + q + 2$ so it linear with respect to the number of neighbours and weekends. Furthermore $m = p + q \sum_{i=1}^{p} |W_i|$ so it is linear with respect to the number of neighbours, weekends, and the availabilities of all neighbours. To create $\mathcal{F}$ wll take $\Theta(n + m)$ time and it takes $\Theta(mn)$ time to run FF-MAXFLOW. The creation of E' can also be done in $\Theta(m)$ since we only need to check the edges. Lastly the creation of A will take $\Theta(m)$ time since we need to inspect all edges from neighbour nodes to weekend nodes. Thus overall the algorithm runs in $\Theta(nm)$ time.

Part B: Note that I will convert W_i to a 2D array W where $W[i][j]$ is 1 if neighbour i is available to garden on weekend j where $1 \leq i \leq p$ and $1 \leq j \leq q$. This can be done in pq time and it makes it easier to express the linear constraints.

VARIABLES:

$$x_{ij} = \begin{cases} 1 & \textit{if neighbour } i \textit{ is assigned to weekend } j, 1 \leq i \leq p, 1 \leq j \leq q \\ 0 & \textit{otherwise} \end{cases}$$

$$y_{ij} = \begin{cases} 1 & \textit{if neighbour } i \textit{ is assigned to drive on weekend } j, 1 \leq i \leq p, 1 \leq j \leq q \\ 0 & \textit{otherwise} \end{cases}$$

OBJECTIVE FUNCTION: max $\sum_{i=1}^{p} \sum_{j=1}^{q} x_{ij}$

CONSTRAINTS:

1. $\sum_{i=1}^{p} x_{ij} \leq 10 \; \forall j, 1 \leq j \leq q$
2. $x_{ij} - W[i][j] \leq 0 \; \forall j, 1 \leq j \leq q$ and $\forall i, 1 \leq j \leq p$
3. $y_{ij} - x_{ij} \leq 0 \; \forall j, 1 \leq j \leq q$ and $\forall i, 1 \leq j \leq p$
4. $\frac{1}{5} \sum_{i=1}^{p} x_{ij} \leq \sum_{i=1}^{p} y_{ij} \; \forall j, 1 \leq j \leq q$
5. $\sum_{j=1}^{q} y_{ij} \leq k_i \; \forall i, 1 \leq i \leq p$
6. $x_{ij} \in \{0, 1\}$ and $y_{ij} \in \{0, 1\}$

JUSTIFICATION: A correct assignment will maximize the number of people available each week. By definition the variable x_{ij} is 1 if a neighbour i is assigned to week j and 0 otherwise. Thus it is correct to maximize the sum of x_{ij} since this leads to the most people gardening. The constraints capture the problem as follows:

Constraint 1 guarantees that the number of people assigned to garden each week is no more than 10. This is because x_{ij} can only be 0 or 1. Thus we only have to check if the sum for each j is less than 10.

Constrain 2 verifies that neighbour i is assigned to week j if and only if they are available that week. This is because in the case where $W[i][j]$ is 0 and x_{ij} is 1 the inequality is false.

Constraint 3 verifies that a person is assigned to drive in week j only if they are assigned to garden in week j. This is because in the case where y_{ij} is 1 and x_{ij} is 0 the inequality is false.

Constraint 4 ensures that enough people are assigned to drive to transport all people assigned to garden.

This is because each person has a 5 seater car so we verify that the the number of people going divided by 5 is less than or equal to the number of drivers.

Constraint 5 guarantees that the for all neighbours i they will not drive more than k_i times. This is because it sums the times each neighbour i will drive and asserts that is it $\leq k_i$.

Constraint 6 is the non-linear constraint of the 0/1 linear program that ensures that x_{ij} and y_{ij} are always either 0 or 1.

All these constraints together capture the entire problem so this linear program satisfies all requirements.

Q2 Matching of Graphs

Part A:

VARIABLES:

$$m_{uv} = \begin{cases} 1 & \textit{if } (u,v) \in E \textit{ is assigned to the matching} \\ 0 & \textit{otherwise} \end{cases}$$

OBJECTIVE FUNCTION: max $\sum_{(u,v)\in E} m_{uv}$

CONSTRAINTS:

1. $\sum_{u\in V} m_{uv} \leq 1 \; \forall v \in V$
2. $m_{uv} \in \{0, 1\}$

JUSTIFICATION: Since we want to capture a maximum matching as a 0/1 linear program we want to maximize the number of edges in the matching. The only constraint we have is that no node in an endpoint more than once which is satisfied by constraint one since is loops over all the edges of each node and guarantees that at most only one edge is used. The second constraint is the non linear constraint of a 0/1 linear program that guarantees m_{uv} is 0 or 1.

Part B: The proof will be done by contradiction. Suppose that $|E'| < |E^*|/2$. Since E' is a matching there must be x edges corresponding to E'. Similarly there are y edges corresponding to E^*. This means that E' and E^* contain $2x$ and $2y$ endpoints respectively. By the inequality $x < y/2$ and it follows that $2x < 2y/2 = y$. This means that E' has strictly less than half of the endpoints E^* has. However if this is the case then this implies that there exists an edge in E^* that is not in E' and also not incident on any endpoint in E'. Thus it is possible to add this edge to E' to create a matching of greater cardinality. However, this is a contradiction since E' is supposed to be maximal. Thus $|E'| \geq |E^*|/2$.

Furthermore, suppose that $|V'| > 2|V^*|$. Since V' is a vertex cover there are x nodes corresponding to V'. Similarly, there are y nodes corresponding to V^*. By the inequality $x > 2y$. Note that since V' comes from a matching, all the edges that are covered by the nodes in V' are disjoint (meaning they do not share an endpoint). This implies that to cover these edges you need $x/2$ nodes. However by the inequality $x > 2y \implies x/2 > y$. Since $x/2 > y$ it is impossible for V^* to have covered them, thus $|V^*|$ cannot be a vertex cover since it does not cover all edges. However this is a contradiction since we assumed that V^* is a minimum vertex cover. Thus $|V'| \leq |V^*|$.

□

Part C: Consider the following algorithms:

```
APPROX-MATCHING(G)
    $S = \emptyset$
    while $G$ has $\geq 2$ nodes
        $(u, v)$ = random edge in $G$
        remove $u, v$ from $G$ and all edges associated with $u$ or $v$
        $S = S \cup (u, v)$
    return $S$
APPROX-VC(G)
    $S = \emptyset$
    while $G$ has $\geq 2$ nodes
        $(u, v)$ = random edge in $G$
        remove $u, v$ from $G$ and all edges associated with $u$ or $v$
        $S = S \cup \{u, v\}$
    return $S$
```

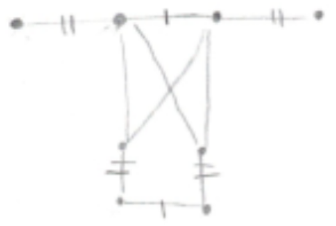
JUSTIFICATION: In both the algorithms above we greedily find a maximal matching of G. The algorithms find a matching because as we select an edge all edges which share an endpoint with it are deleted. It is maximal because we continue until no edges are left by the while loop condition. APPROX-MATCHING will return the set of edges associated with the maximal matching. By part b we know that the size of this matching must be at least half of the size of the optimal matching. Thus we have a $1/2$ approximation. Similarly APPROX-VC returns the nodes that cover the maximal matching. By part b we know that the size of this cover is at most double the size of an optimal minimum vertex cover. Thus this is a 2 approximation of vertex cover.

RUNNING TIME: The running time of both algorithms in $O(n + m)$. This is because at each iteration the algorithm selects an edge randomly and removes all edges which share an endpoint. This means that it looks at all edges and nodes at most once. Thus we have a linear running time of $O(n + m)$.

<u>Part D:</u> The approximation is tight.

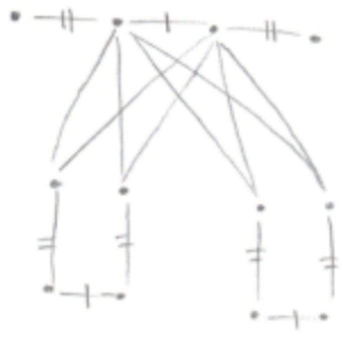
For APPROX-VC consider a complete graph with $2k + 1$ nodes. Then this graph has a maximal matching of k edges which will always be produced by the algorithm. This is because the graph is complete so can always find $n/2$ disjoint edges where n is the number of nodes. This means that APPROX-VC produces a cover of $2k$ nodes. However in a complete graph the minimum vertex cover will have k nodes. Thus the 2-approximation is tight.

For APPROX-MATCHING consider the example below:



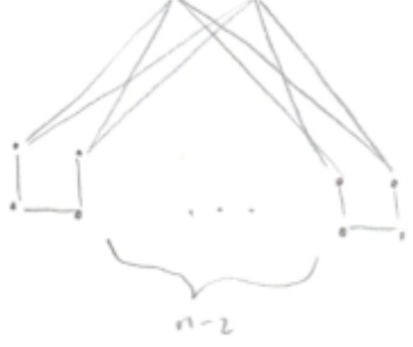
maximal : 2

maximum : 4



maximal : 3

maximum : 6



maximal : n+1

maximum : 2n+2

In the example above it is possible for APPROX-MATCHING to give a matching which is 1/2 the size of the optimal. Note that it cannot be guaranteed due to the random nature of the algorithms (at each iteration they select random edges). As seen in the image above a maximal matching of n edges can be found but a maximum matching will always have $2n$ edges.