Solutions for Homework Assignment #8

Answer to Question 1.

a. We first describe how to construct from the input parameters a flow network $\mathcal{F}$ whose maximum flow allows us to find the desired assignment (if one exists). Let $n_j = |\{i : \ j \in W_i\}|$ — that is, n_j is the number of neighbours available to garden on weekend j and who must therefore be driven to the garden that weekend. Thus, $\lceil n_j/5 \rceil$ is the number of cars needed on weekend j.

- The graph of the flow network has $p + q + 2$ nodes: a source node s, a sink node t, nodes $x_1, \ldots, x_p$ for the p neighbours (and hence drivers), and nodes $y_1, \ldots, y_q$ for the q weekends.
- The edges of the graph and their capacities are as follows:
  - For every neighbour i, $1 \leq i \leq p$, there is an edge (s, x_i) with capacity k_i. This represents the maximum number of times i is willing to drive.
  - For every weekend j, $1 \leq j \leq q$, there is an edge (y_j, t) with capacity $\lceil n_j/5 \rceil$. This represents the number of cars needed to drive the neighbours available to garden on weekend j.
  - For every neighbour i and weekend j, $1 \leq i \leq p$ and $1 \leq j \leq q$, such that $j \in W_i$, there is an edge (x_i, y_j) with capacity 1. This represents that neighbour i is available on weekend j.

Note: There are several alternatives to this flow network. For example, it could be directed in the reverse way (from weekends to neighbours, instead of from neighbours to weekends as above). Or, instead of reasoning about the number of cars needed, we could reason about the number of neighbours that need to be driven. In this case the capacity of the edge (s, x_i) would be $5k_i$, the capacity of the edge (y_j, t) would be n_j, and the capacity of the edge (x_i, y_j) would be 5.

Algorithm: If an assignment that satisfies the constraints exists, the algorithm below returns an array $A[1..p]$ where $A[i]$ is the set of weeks in which neighbour i is assigned to drive; otherwise, the algorithm returns the string "impossible".

```
    OptAssignment(p, q, W[1..p], k[1..p])
1   for j := 1 to q do n_j := |{i : j ∈ W[i]}|
2   construct the flow network F described above
3   f := MaxFlow(F)
4   for i := 1 to p do A[i] := {j : f(x_i, y_j) = 1}
5   if Σ_{i=1}^{p} f(x_i, t) = Σ_{j=1}^{q} ⌈n_j/5⌉ then return A
6   else return "impossible"
```

Correctness: Suppose that there is an assignment that satisfies the constraints. That is, neighbour i is assigned to drive on $\ell_i \leq k_i$ weekends $i_1, \ldots, i_{\ell_i}$, on each of which he or she is available to garden, so that there are $\lceil n_j/5 \rceil$ neighbours assigned to drive on weekend j. Then there is a flow f in $\mathcal{F}$ that assigns ℓ_i units of flow on edge (s, x_i), which is then distributed to the edges (x_i, y_{i_t}), $1 \leq t \leq \ell_i$, one unit of flow to each, so that $\lceil n_j/t \rceil$ units of flow arrive into node y_j and are then delivered to node t on edge (y_j, t). Thus, $f(x_i, y_j) = 1$ if and only if neighbour i is assigned to drive on weekend j. Since this flow saturates all edges into t, it is a maximum flow (by the max-flow-min-cut theorem) and its value is $\mathcal{V}(f) = \sum_{j=1}^{q} \lceil n_j/5 \rceil$. So the algorithm computes an assignment that satisfies the constraints in line 4 and returns it in line 5.

If, on the other hand, there is no assignment that satisfies the constraints, there is no flow that saturates all the edges into t. So, in this case, the value of the maximum flow is less than $\sum_{j=1}^{q} \lceil n_j/5 \rceil$, and the algorithm returns "impossible" in line 6.

Running-time analysis: The graph has $O(p+q)$ nodes and $O(pq)$ edges. So we have:

- Line 1 takes $O(pq)$ time
- Line 2 takes $O(p+q+pq) = O(pq)$ time
- Line 3 takes $O\big((pq)^2(p+q)\big) = O\big(p^2q^2(p+q)\big)$, assuming we use the fewest-edges-path rule to choose an augmenting path
- Line 4 in lines 3-5 takes $O(pq)$ time
- Lines 5-6 take $O(1)$ time

So the overall running time of the algorithm is $O\big(p^2q^2(p+q)\big)$.

b. Variables: x_{ij} and y_{ij} for $1 \le i \le p$ and $1 \le j \le q$. We will have $x_{ij} = 1$ iff neighbour i is assigned to garden on weekend j, and $y_{ij} = 1$ iff neighbour i is assigned to drive on weekend j.

Objective function: Maximize $\sum_{i=1}^{p}\sum_{j=1}^{q} x_{ij}$, i.e., maximize the number of neighbours assigned to garden over the q weekends.

Constraints:

- $x_{ij}, y_{ij} \in \{0,1\}$, for all $1 \le i \le p$ and $1 \le j \le q$ (all variables must be assigned values 0 or 1)
- $\sum_{i=1}^{p} x_{ij} \le 10$, for all $1 \le j \le q$ (constraint (1))
- $x_{ij} = 0$, for all $1 \le i \le p$ and $1 \le j \le q$ such that $j \notin W_i$ (constraint (2))
- $x_{ij} - y_{ij} \ge 0$, for all $1 \le i \le p$ and $1 \le j \le q$, (a neighbour assigned to drive on a weekend also gardens that weekend)
- $\sum_{i=1}^{p} y_{ij} - \sum_{i=1}^{q} x_{ij}/5 \ge 0$, for all $1 \le j \le q$ (constraint (3))
- $\sum_{j=1}^{q} y_{ij} \le k_i$, for all $1 \le i \le p$ (constraint (4))

Answer to Question 2.

a. Variables: x_e for each $e \in E$. The intention is that

$$x_e = \begin{cases} 1, & \text{if } e \text{ is in a maximum matching of } G \\ 0, & \text{otherwise} \end{cases}$$

Objective function: Maximize $\sum_{e \in E} x_e$.

Constraints:

- $x_e \in \{0,1\}$. (Nonlinear constraint indicating the possible values of the variables.)
- $x_e + x_{e'} \le 1$, for all $e, e' \in E$ such that $e \ne e'$ and $e \cap e' \ne \varnothing$. (If two distinct edges share an endpoint, at most one of them is in the matching.)

b. To prove that V' is a vertex cover suppose, for contradiction, that it is not. Then there is some edge $\{u,v\} \in E$ such that both u and v are not V', i.e., are not endpoints of any edge in E'. Then $E' \cup \big\{\{u,v\}\big\}$ is a matching, contradicting that E' is a maximal matching.

To prove that $|V'| \le 2|V^*|$, we first recall that the size of any matching is less than the size of any vertex cover. (We proved this fact using a the Pigeonhople Principle when discussing bipartite graph matching, and we noted that it is true for all graphs, not just bipartite ones.) Thus $|E'| \le |V^*|$. Since E' is a matching, $|V'| = 2|E'|$. Therefore $|V'| \le 2|V^*|$, as wanted.

To prove that $|E'| \ge |E^*|/2$ suppose, for contradiction, that $|E'| < |E^*|/2$, Since $|V'| = 2|E'|$, it follows that $|V'| < |E^*|$. Let f be any function $f : E^* \to V'$ such that $f(e)$ is an endpoint of e. Note that such functions exist because every edge in E^* has at least one endpoint in V', since (as we proved earlier) V' is a vertex cover. Since $|E^*| > |V'|$, by the pigeonhole principle, there exist distinct $e, e' \in E^*$ such that $f(e) = f(e')$ — that is, there exist distinct edges in E^* that share an endpoint, contradicting that E^* is a matching.

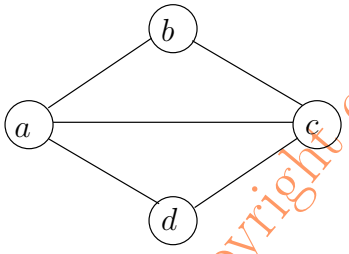
c. ApproxVertexCover&Matching(G)

```
1  E' := V' := ∅
2  for each edge {u, v} do
3      if u ∉ V' and v ∉ V' then E' := E' ∪ {{u, v}}; V' := V' ∪ {u, v}
4  return (V', E')
```

Clearly, at the end of the for-loop E' contains a maximal matching and V' contains the endpoints of the edges in E'. Thus, by part (b), V' is a vertex cover with at most twice as many nodes as a minimum vertex cover, and E' is a matching with at least half as many edges as a maximum matching.

d. The approximation bounds are tight.

In the graph shown below $E' = \{\{a, c\}\}$ is a maximal matching, and $E^* = \{\{a, b\}, \{c, d\}\}$ is a maximum (and therefore also maximal) matching. If the algorithm considers the edge $\{a, c\}$ first, it will return a maximal matching whose size is half the size of the maximum matching. If the algorithm considers the edge $\{a, b\}$ first, it will return a maximal matching $E' = E^*$ with four endpoints $\{a, b, c, d\}$, that is a vertex cover with twice as many nodes as the minimum vertex cover $\{a, c\}$.



THAT'S IT WITH HOMEWORK SOLUTIONS, FOLKS!