Algorithm Design and Analysis (Fall 2022) Assignment 5

Xiangyuan Xue (521030910387)

1. (a) Consider a fractional allocation, where the portion of item j allocated to agent i is

$$a_{ij} = \frac{v_{ij}}{\sum\limits_{k=1}^{n} v_{kj}}$$

so for agent i, the value is

$$v_i(A_i) = \sum_{j=1}^{m} a_{ij} = \sum_{j=1}^{m} \frac{v_{ij}}{\sum\limits_{k=1}^{n} v_{kj}} \geq \frac{1}{n} \sum_{j=1}^{m} v_{ij} = \frac{1}{n} v_i([m])$$

Therefore, this allocation is qualified.

(b) In this flow network, u_i represents the agents and w_i represents the items. The edges from s to u_i declares the requirement of agent i. The edges from u_i to w_j means that item j can make contribution to agent i's value. The edges from w_j to t limits the contribution of item j.

From 1a we know that all the agents' requirement can be satisfied. So the capacity of edges from s to u_i will be fully reached. The value of maximum flow is

$$\max_f v(f) = \frac{1}{n} \sum_{i=1}^{n} v_i([m])$$

where f implies a fractional allocation such that each agent i receives a value of at least $\frac{1}{n} v_i([m])$.

(c) From 1b we know that the value of maximum flow is limited by the cut containing the edges from s to u_i. Since the inequality holds that

$$\lfloor \frac{1}{n} v_i([m]) \rfloor \leq \frac{1}{n} v_i([m])$$

the cut is still the bottleneck after modification. So the value of maximum flow is

$$\max_f v(f) = \sum_{i=1}^{n} \lfloor \frac{1}{n} v_i([m]) \rfloor$$

where f implies an allocation such that $v_i(A_i) = \lfloor \frac{1}{n} v_i([m]) \rfloor$.

(d) Suppose ∞ is an integer which is large enough. We claim that the capacity $c(e)$ of each edge e is an integer. By Integrality Theorem, the maximum flow in 1c can be made integral, namely for each edge e, the value of $f(e)$ is always an integer.

We have figured out the value of maximum flow in 1c, and this means that in the maximum flow, the capacity of edges from s to u_i will be fully reached. Since the value of $f(e)$ for each edge e is an integer, the maximum flow implies a proportional allocation by the value of flow on edges from u_i to w_j when items are indivisible.

Therefore, the proportional allocation always exists even when items are indivisible.

(e) An algorithm to compute a proportional allocation is as follow.

Algorithm 1 Proportional Allocation

```
Input: Preference matrix $\boldsymbol{V}^{n\times m} \in \{0,1\}^{n\times m}$
Output: Partition $(A_1, A_2, \dots, A_n)$ of $[m] = \{1, 2, \dots, m\}$
1: Construct a flow network where $c(s,u_i) = \lfloor \frac{1}{n} v_i([m]) \rfloor$, $c(u_i, w_j) = \infty$, $c(w_j, t) = 1$
2: Compute the maximum flow $f$ by Edmonds-Karp Algorithm
3: for $i = 1$ to $n$ do
4:     $A_i \leftarrow \{j : f(u_i, v_j) = 1\}$
5: return $(A_1, A_2, \dots, A_n)$
```

The correctness of this algorithm has been proved in 1d.

Consider its running time. In the flow network, there are $n+m+2$ vertices and no more than $nm+n+m$ edges. Its time complexity is dominated by Edmonds-Karp Algorithm, which takes $O\left((n+m+2)(nm+n+m)^2\right) = O\left(n^2m^2(n+m)\right)$ time. Therefore, its time complexity is $O\left(n^2m^2(n+m)\right)$.

2. (a) Since G contains a perfect match, there exists a matching with size n. Consider an arbitrary $S \subseteq A$, for any $a \in S$, there exists $b \in B$ such that $(a,b) \in E$. By the properties of matching, for any $a, a' \in S$, corresponding $b, b' \in B$ are different, namely $b \neq b'$. Thus, there exists at least $|S|$ vertices $b \in B$ in $N(S)$, namely $|N(S)| \geq |S|$.

Since S is arbitrary, we can conclude that $|N(S)| \geq |S|$ for any $S \subseteq A$.

(b) For the minimum s-t cut, we will not choose the edges (a,b) where $a \in A$ and $b \in B$, because their capacity is ∞. Thus, the minimum s-t cut crosses two types of edges, (s,a) where $a \in A$ and (b,t) where $b \in B$. At the same time, G is separated into two parts, denoted by $\{s\} \cup A \cup B$ and $A' \cup B' \cup \{t\}$, where $A = \{a_1, a_2, \dots, a_p\}$, $B = \{b_1, b_2, \dots, b_q\}$, $A' = \{a_{p+1}, a_{p+2}, \dots, a_n\}$, $B' = \{b_{q+1}, b_{q+2}, \dots, b_n\}$.

Since $|N(S)| \geq |S|$, we replace S with A' and have $|N(A')| \geq |A'|$. Since $N(A') \subseteq B'$, we have $|N(A')| \leq |B'|$. Thus

$$|A| + |B'| \geq |A| + |N(A')| \geq |A| + |A'| = n$$

where $|A| + |B'|$ is the value of the minimum s-t cut, which is no larger than the value of a specific cut crossing (s,a) where $a \in A$, namely $|A| + |B'| \leq n$.

Therefore, the minimum s-t cut has value $|A| + |B'| = n$.

(c) Consider the new graph H as a flow network and we have following explanation.

- The capacity of edges (s, a) where $a \in A$ is 1, which limits that each $a \in A$ can be matched only once.
- The capacity of edges (b, t) where $b \in B$ is 1, which limits that each $b \in B$ can be matched only once.
- By flow conservation, a positive flow on edges (a, b) where $a \in A$ and $b \in B$ represents a single matching. Thus, a flow with value $v(f)$ corresponds to a matching with $v(f)$ edges.

Therefore, the maximum flow in H corresponds to the maximum matching in G. G contains a perfect matching if and only if the maximum matching in G contains n edges, namely the value of maximum flow in H is n.

According to 2b, if $|N(S)| \geq |S|$ holds for all $S \subseteq A$, the minimum s-t cut has value n. By Max-Flow-Min-Cut Theorem, the value of maximum flow in H is also n, which indicates that G contains a perfect matching.

3. (a) In this linear program, $x_e \geq 0$ indicates the fractional matching on the edge e. The maximum matching problem limits that each edge e is chosen only once. For the fractional version, this limitation is described as for each $v \in V$, it holds that $\sum_{e=(u,v)} x_e \leq 1$. The object that we want to maximize is the total fractional matching, namely $\sum_{e \in E} x_e$. Therefore, this linear program describes the fractional version of the maximum matching problem.

(b) The dual of this linear program is written as follow.

$$\begin{aligned} \min \quad & \sum_{v \in V} y_v \\ \text{s.t.} \quad & \forall (u, v) \in E : y_u + y_v \geq 1 \\ & \forall v \in V : y_v \geq 0 \end{aligned}$$

In this dual program, $y_v \geq 0$ indicates the fractional cover on the vertex v. The minimum vertex cover problem requires that each vertex v is covered at least once. For the fractional version, this requirement is described as for each $(u, v) \in E$, it holds that $y_u + y_v \geq 1$. The object that we want to minimize is the total fractional cover, namely $\sum_{v \in V} y_v$. Therefore, this dual program describes the fractional version of the minimum vertex cover problem.

(c) We prove this proposition by induction on the order of the determinant.

- For a 1×1 submatrix $\boldsymbol{D}$ of $\boldsymbol{A}$, the value of its only element is from $\{1, 0, -1\}$, so its determinant $|\boldsymbol{D}| \in \{1, 0, -1\}$.
- Suppose that for any $k \times k$ submatrix $\boldsymbol{D}$ of $\boldsymbol{A}$, it holds that $|\boldsymbol{D}| \in \{1, 0, -1\}$.
- Consider a $(k+1) \times (k+1)$ submatrix $\boldsymbol{D}$ of $\boldsymbol{A}$. In a bipartite graph, each edge is incident with exactly two vertices, which indicates that there are no more than two 1s for any column of $\boldsymbol{D}$.

- If there exists a column that is all-zero, it is obvious that $|\boldsymbol{D}| = 0$.
- If there exists a column containing only one 1, we can remove the specific row and column, and $|\boldsymbol{D}|$ will be equal to the determinant of the remaining $k \times k$ submatrix, which means that $|\boldsymbol{D}| \in \{1, 0, -1\}$.
- Otherwise, each column contains two 1s. By the properties of bipartite graph, the rows of $\boldsymbol{D}$ can be partitioned into two parts $(\boldsymbol{r}_1, \boldsymbol{r}_2, \dots, \boldsymbol{r}_s)^T$ and $(\boldsymbol{r}_{s+1}, \dots, \boldsymbol{r}_k, \boldsymbol{r}_{k+1})^T$, representing two sides of the bipartite graph. Since each edge is incident with both sides, it holds that

$$\sum_{i=1}^{s} \boldsymbol{r}_i - \sum_{i=s+1}^{k+1} \boldsymbol{r}_i = \boldsymbol{0}$$

which shows that the rows of $\boldsymbol{D}$ is linearly dependent, so $|\boldsymbol{D}| = 0$.

Thus, it still holds that $|\boldsymbol{D}| \in \{1, 0, -1\}$.

Therefore, for any submatrix $\boldsymbol{D}$ of $\boldsymbol{A}$, its determinant $|\boldsymbol{D}| \in \{1, 0, -1\}$. In other words, the incident matrix of bipartite graph $\boldsymbol{A}$ is totally unimodular.

(d) With the incident matrix $\boldsymbol{A}$, we can rewrite the linear program describing the fractional version of the maximum matching problem in matrix form.

$$\begin{aligned} \max \quad & \boldsymbol{1}^T \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{1} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned}$$

Since $\boldsymbol{A}$ is totally unimodular, the solution of this linear program is integral. Note that if $x_e^* \geq 2$, the corresponding vertices will be selected more than once, which violates the constraints. Therefore, for each $e \in E$, it holds that $x_e^* \in \{0, 1\}$ and the optimal solution is exactly the size of the maximum matching.

Rewrite the dual program describing the fractional version of the minimum vertex cover problem in matrix form.

$$\begin{aligned} \min \quad & \boldsymbol{1}^T \boldsymbol{y} \\ \text{s.t.} \quad & \boldsymbol{A}^T\boldsymbol{y} \geq \boldsymbol{1} \\ & \boldsymbol{y} \geq \boldsymbol{0} \end{aligned}$$

Since $\boldsymbol{A}$ is totally unimodular, the solution of this dual program is integral. Note that if $y_v^* \geq 2$, we can decrease y_v to 1 while satisfying the constraints, which violates the optimality. Therefore, for each $v \in V$, it holds that $y_v^* \in \{0, 1\}$ and the optimal solution is exactly the size of the minimum vertex cover.

By LP Duality Theorem, the linear program has the same optimal solution with the dual program. In other words, the size of the maximum matching equals to the size of the minimum vertex cover, which is exactly König-Egerváry Theorem.

(e) Consider the following graph that is not bipartite.

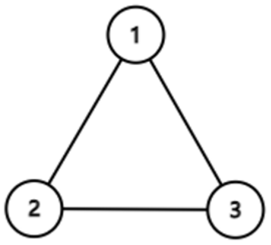

Figure 1: Counterexample for 3e

Note that in this graph, the size of the maximum matching is 1, but the size of the minimum vertex cover is 2, which indicates that the claim in König-Egerváry Theorem fails on this non-bipartite graph.

4. (a) In this linear program, $x_u \geq 0$ indicates the fractional cover on the vertex u. The minimum vertex cover problem requires that each vertex u is covered at least once. For the fractional version, this requirement is described as for each $(u, v) \in E$, it holds that $x_u + x_v \geq 1$. The object that we want to minimize is the total fractional cover, namely $\sum_{u \in V} x_u$. Therefore, this dual program describes the fractional version of the minimum vertex cover problem.

   (b) Consider an arbitrary vertex cover $C \subset V$, which corresponds to a solution $\{x_u\}_{u \in V}$ to the linear program where

$$x_u = \begin{cases} 1, & u \in C \\ 0, & u \notin C \end{cases}$$

   The size of C can be written as

$$|C| = \sum_{u \in V} x_u$$

   Since C is a vertex cover of G, for any edge $(u, v) \in E$, at least one of u and v is in C. In other words, at least one of x_u and x_v is 1. Thus, it holds that $x_u + x_v \geq 1$, which promises that the solution $\{x_u\}_{u \in V}$ is feasible.

   Note that $\{x_u^*\}_{u \in V}$ is an optimal solution to the linear program. We have the inequality that

$$\sum_{u \in V} x_u \geq \sum_{u \in V} x_u^*$$

   which holds for any feasible solution $\{x_u\}_{u \in V}$.

   Therefore, we conclude that $\sum_{u \in V} x_u^*$ is a lower bound to the size of any vertex cover.

   (c) The constraints of the linear program requires that for any edge $(u, v) \in E$, it holds that $x_u^* + x_v^* \geq 1$. Then we claim that

$$\max\{x_u^*, x_v^*\} \geq 0.5$$

Otherwise, we will have $x_u^* + x_v^* < 0.5 + 0.5 = 1$, which violates the constraints. So at least one of x_u^* and x_v^* is no less than 0.5. Then by definition, at least one of u and v will be in $S(G)$, which promises that $S(G)$ is a valid vertex cover.

Note that only those $u \in V$ such that $x_u^* \geq 0.5$ will be selected. So an upper bound to $|S(G)|$ is given as

$$|S(G)| = \sum_{u \in V} 1_{x_u^* \geq 0.5} \leq \sum_{\substack{u \in V \\ x_u^* \geq 0.5}} 2x_u^* \leq 2 \sum_{u \in V} x_u^*$$

From 4b we already have a lower bound that

$$\sum_{u \in V} x_u^* \leq \mathrm{OPT}(G)$$

Therefore, we have $|S(G)| \leq 2 \cdot \mathrm{OPT}(G)$, indicating that it is a 2-approximation algorithm.

(d) Let $\boldsymbol{A}$ denote the incident matrix of G. Since G is a bipartite graph, $\boldsymbol{A}$ will be totally unimodular, which has been proved in 3c. Since all the values on the right hand side of this linear program are integers, there exists an integral optimal solution to this linear program.

Note that if $x_u^* \geq 2$, we can decrease x_u^* to 1 while satisfying the constraints, which violates the optimality. Therefore, for each $u \in V$, it holds that $x_u^* \in \{0, 1\}$ and the linear program directly gives a minimum vertex cover.

5. (a) Consider an arbitrary iteration of Dinic's Algorithm. We construct the level graph $\overline{G}^f$ of the residual network G^f.

Note that the blocking flow f brings at least one critical edge for each s-t path on $\overline{G}^f$. When we update f by adding the blocking flow, these critical edges will disappear on G^f, which means each s-t path on $\overline{G}^f$ will be cut off and thus, s and t will be disconnected.

According to the definition of the level graph $\overline{G}^f$, we can claim that the connection between some level l and $l+1$ will be cut off on G^f. When we construct a new level graph $\overline{G}^{f'}$, at least one level should be added to make t reachable from s. Then the level of the sink t will be increased by at least 1. Since the level of t on $\overline{G}^f$ is just the distance from s to t on G^f, we have

$$dist'(t) \geq dist(t) + 1$$

Therefore, the distance from s to t on G^f is increased by at least 1.

(b) We can compute a blocking flow on a level graph by following algorithm.

Algorithm 2 Blocking Flow Generation

```
Input: Level graph $\overline{G}^f$
Output: Blocking flow $f$
1: initialize $f$ as an empty flow and $G^f = \overline{G}^f$
2: while there exists an $s$-$t$ path $p$ on $G^f$ do
3:     find the edge $e \in p$ with minimum capacity $c$
4:     update $f$ by pushing a flow with value $c$ along $p$
5:     update $G^f$
6:     remove $e$ from $G^f$
7: return $f$
```

Note that the s-t path denoted by $s \to u_1 \to u_2 \to \cdots \to u_n \to t$ has the property that $dist(u_{k+1}) - dist(u_k) = 1$ in $\overline{G}^f$. The searching will be implemented by DFS.

To prove the correctness, we should notice that whenever an edge e becomes critical, it will be removed from G^f. When this algorithm terminates, no s-t path p can be found in G^f, namely each s-t path in $\overline{G}^f$ contains at least one critical edge.

Consider its running time. An arbitrary s-t path contains at most $|V|$ vertices, so finding path, updating flow, updating G^f and removing e will take $O(|V|)$ time. In each iteration, at least one edge becomes critical and is removed. So the number of iteration is no larger than $|E|$. Therefore, its time complexity is $O(|V| \cdot |E|)$.[1]

(c) Dinic's Algorithm is described in following pseudo-codes.

Algorithm 3 Dinic's Algorithm

```
Input: Flow Network $G$
Output: Blocking flow $f$
1: initialize $f$ as an empty flow and $G^f = G$
2: while there exists an $s$-$t$ path $p$ on $G^f$ do
3:     construct the level graph $\overline{G}^f$ of $G^f$
4:     find a blocking flow on $\overline{G}^f$
5:     update $f$ by pushing the blocking flow
6:     update $G^f$
7: return $f$
```

By 5b we know that finding a blocking flow takes $O(|V| \cdot |E|)$ time. Constructing the level graph and updating f and G^f will take $O(|E|)$ time. Thus, the operations inside the iteration are $O(|V| \cdot |E|)$.

By 5a we know that after each iteration, the distance from s to t on G^f is increased by at least 1. Since the distance from s to t is no larger than $|V|$, the number of iteration is also no larger than $|V|$.

Therefore, the overall time complexity for Dinic's Algorithm is $O(|V|^2 \cdot |E|)$.

(d) For a bipartite graph $G = (A, B, E)$, we construct a flow network H with vertices $C = A \cup B \cup \{s, t\}$. There is an edge from s to each $a \in A$ with capacity 1. There is an edge from each $b \in B$ to t with capacity 1. There is an edge from $a \in A$ to

$b \in B$ with capacity 1 if $(a, b) \in E$. The value of maximum flow in H is the size of maximum matching in G.

First, we consider the running time for a single iteration in Dinic's Algorithm. Since the capacity of each edge is 1, for an arbitrary s-t path p on G^f, any edge $e \in p$ will be critical and removed immediately after updating f. Thus, each edge will be chosen only once in this iteration, so the time complexity is $O(|E|)$.

Then we estimate the number of iteration in Dinic's Algorithm. Suppose that we have finished $\left\lfloor \sqrt{|V|} \right\rfloor$ iterations and there are k iterations remaining. By 5a we know that the distance from s to t has been increased by at least $\left\lfloor \sqrt{|V|} \right\rfloor$, so any s-t path will contain more than $\left\lfloor \sqrt{|V|} \right\rfloor$ vertices in future. Since the capacity of each edge is 1, different s-t paths found in different iterations will not intersect each other. So they totally contain more than $k \left\lfloor \sqrt{|V|} \right\rfloor$ vertices, requiring that

$$k \left\lfloor \sqrt{|V|} \right\rfloor < |V|$$

so k is upper bounded by the inequality that

$$k < \frac{|V|}{\left\lfloor \sqrt{|V|} \right\rfloor} < 2 \left\lceil \sqrt{|V|} \right\rceil$$

Thus, the number of iteration is

$$\left\lfloor \sqrt{|V|} \right\rfloor + k = O(\sqrt{|V|}) + 2O(\sqrt{|V|}) = O(\sqrt{|V|})$$

Therefore, the total time complexity is $O(|E| \cdot \sqrt{|V|})$ for Dinic's Algorithm applied to finding a maximum matching on a bipartite graph.[2]

6. (a) It takes me about 15 hours to finish this assignment.
   (b) I prefer a 3/5 score for its difficulty.
   (c) I have no collaborators. Papers and websites referred to are listed below.

References

[1] Wikipedia. Dinic's Algorithm. https://en.wikipedia.org/wiki/Dinic's_algorithm.

[2] OI-Wiki. Maximum Flow. http://oi-wiki.com/graph/flow/max-flow.