Algorithm Design and Analysis

Assignment 5

Deadline: Dec 25, 2022

1. (25 points) **[Fair Division Revisited]** Suppose we want to allocate m indivisible items to n agents. A $n \times m$ matrix $V = [v_{ij}]_{i=1,\ldots,n;j=1,\ldots,m}$ describes agents' preferences over the items, where v_{ij} is the value of item j according to agent i's preference. We assume agents' valuations are binary, i.e., $v_{ij} \in \{0, 1\}$ for any i and j. Given a subset S of items, agent i's value on S is naturally defined by $v_i(S) = \sum_{j \in S} v_{ij}$.

   An allocation $(A_1, \ldots, A_n)$ is a partition of $[m] := \{1, \ldots, m\}$, where $A_i \subseteq [m]$ is the set of the items allocated to agent i. An allocation is *proportional* if $v_i(A_i) \geq \lfloor \frac{1}{n} v_i([m]) \rfloor$ holds for any agent i. That is, in a proportional allocation, each agent receives his/her average integer value, rounded down, of the entire item set. For example, if $v_{ij} = 1$ for all i and j, the average value for each agent is m/n, and each agent needs to get at least $\lfloor \frac{m}{n} \rfloor$ items to guarantee proportionality.

   In this question, we are going to prove the existence of proportional allocations and design an algorithm to find one.

   (a) (5 points) Let us first assume, for this moment, that items are *divisible*. That is, we are allow to fractionally allocate a single item such that an agent gets some portion of it. Show that there exists a *fractional* allocation such that each agent i receives a value of at least $\frac{1}{n} v_i([m])$.

   (b) (5 points) Consider the flow network with $m+n+2$ vertices $s, u_1, \ldots, u_n, w_1, \ldots, w_m, t$ described as follows. There is a directed edge from s to each u_i with capacity $\frac{1}{n} v_i([m])$. There is a directed edge from u_i to w_j with capacity ∞ if $v_{ij} = 1$. There is a directed edge from each w_j to t with capacity 1. According to Part (a), what is the maximum flow in this network?

   (c) (5 points) Now, we modify the network by reducing the capacity of each edge (s, u_i) from $\frac{1}{n} v_i([m])$ to $\lfloor \frac{1}{n} v_i([m]) \rfloor$. What is the maximum flow in this network?

   (d) (5 points) Show that the maximum flow in Part (c) can be made integral. Use this result to show that the proportional allocation always exists even when items are *indivisible*.

   (e) (5 points) Design an algorithm to compute a proportional allocation.

   (a) The allocation where each item is evenly allocated to all the n agents (i.e., each agent get a $1/n$ portion of each item) satisfies the requirement. In this allocation, each agent i gets a value of exactly $\frac{1}{n} v_i([m])$.

   (b) It is easy to see that a flow f in this network corresponds to a fractional allocation, where $f(s, u_i)$ is the value received by agent i and $f(u_i, w_j)$ is the fraction of item

j allocated to agent i. From part (a), we know that there is a flow with value at least $\sum_{i=1}^{n} \frac{1}{n} v_i([m])$. In addition, this is the maximum value of any possible flow, as $\sum_{i=1}^{n} c(s, u_i) = \sum_{i=1}^{n} \frac{1}{n} v_i([m])$. Thus, the maximum flow has value $\sum_{i=1}^{n} \frac{1}{n} v_i([m])$.

(c) Let $\alpha_i = \lfloor \frac{1}{n} v_i([m]) \rfloor / \frac{1}{n} v_i([m])$. Let f be the maximum flow described in part (b), and let f' be the flow defined as follows:

- for each edge (s, u_i), $f'(s, u_i) = \alpha_i f(s, u_i)$;
- for each edge (u_i, w_j), $f'(u_i, w_j) = \alpha_i f(u_i, w_j)$;
- for each edge (w_j, t), $f'(w_j, t) = \sum_{u_i:(u_i,w_j)\in E} f'(u_i, w_j)$.

It is easy to verify that f' is a valid flow with maximum possible value $\sum_{i=1}^{n} \lfloor \frac{1}{n} v_i([m]) \rfloor$.

(d) By Flow Integrality Theorem, f' in part (c) can be made integral since all the capacities are integers. It is easy to see that f' corresponds to a *partial* proportional allocation, a proportional allocation where each item is either fully allocated to a single agent or unallocated. We can then allocate those unallocated items arbitrarily, and we obtain a proportional allocation.

(e) The algorithm first constructs a flow network as described in part (b) with capacities given by part (c). Then it computes a maximum integral flow by Edmonds-Karp algorithm (or any other algorithms for the maximum flow problem). This gives us a partial proportional allocation. The algorithm then terminates by allocating the remaining unallocated items arbitrarily (say, to agent 1).

2. (25 points) **[Hall Marriage Theorem]** Given a bipartite graph $G = (A, B, E)$ with $|A| = |B| = n$, *Hall Marriage Theorem* states that G contains a perfect matching (i.e., a matching with size n) if and only if $|N(S)| \geq |S|$ for any $S \subseteq A$, where $N(S) = \{b \in B \mid \exists a \in S \subseteq A : (a, b) \in E\}$ is the set of all the neighbors of the vertices in S. In this question, we are going to prove Hall Marriage Theorem.

   (a) (5 points) Prove the necessity part: if G contains a perfect matching, then $|N(S)| \geq |S|$ for any $S \subseteq A$.

   (b) (10 points) Construct a directed weighted graph with vertex set $V = A \cup B \cup \{s\} \cup \{t\}$. The edge set is defined as follows. For each $a \in A$, construct a directed edge (s, a) with weight 1; for each $a \in A$ and $b \in B$, construct a directed edge (a, b) with weight ∞ if (a, b) is an edge in the original graph G; for each $b \in B$, construct a directed edge (b, t) with weight 1. Prove that, if $|N(S)| \geq |S|$, the minimum s-t cut has value n.

   (c) (10 points) Prove the sufficiency part: if $|N(S)| \geq |S|$ holds for all $S \subseteq A$, then G contains a perfect matching.

   (a) Suppose G contains a perfect matching. For any $S \subseteq A$, $N(S)$ should contain those vertices matched to vertices in S. This already implies $|N(S)| \geq |S|$.

   (b) Notice that $L = \{s\}$ and $R = V \setminus \{s\}$ give a cut of value n, so the minimum cut has size at most n. Suppose for the sake of contradiction there exists a cut (L, R) with size less than n. The size of the cut is given by $|L \cap B| + |R \cap A| < n$. Moreover, there should be no edge between $L \cap A$ and $R \cap B$ in the original graph, for otherwise the cut has value infinity. Therefore, $N(L \cap A) \subseteq (L \cap B)$. On the other hand, $|L \cap B| < n - |R \cap A| = |L \cap A|$ (the last equality is due to $|L \cap A| + |R \cap A| = |A| = n$), so Hall's condition fails on $S = L \cap A$.

   (c) Suppose $|N(S)| \geq |S|$ holds for all $S \subseteq A$. The minimum s-t cut in the network in part (b) has size n. By the max-flow-min-cut theorem, the maximum flow has value n. Since the capacities are all integers, there exists an integral flow of value n (Flow Integrality Theorem). This corresponds to a matching of size n.

3. (25 points) **[LP-Duality and Total Unimodularity]** In this question, we will prove König-Egerváry Theorem, which states that, in any bipartite graph, the size of the maximum matching equals to the size of the minimum vertex cover. Let $G = (V, E)$ be a bipartite graph.

   (a) (4 points) Explain that the following linear program describes *the fractional version of* the maximum matching problem (i.e., replacing $x_e \geq 0$ to $x_e \in \{0, 1\}$ gives you the exact description of the maximum matching problem).

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} x_e \\ \text{subject to} \quad & \sum_{e:e=(u,v)} x_e \leq 1 && (\forall v \in V) \\ & x_e \geq 0 && (\forall e \in E) \end{aligned}$$

   (b) (4 points) Write down the dual of the above linear program, and justify that the dual program describes the fractional version of the minimum vertex cover problem.

   (c) (7 points) Show by induction that the *incident matrix* of a bipartite graph is totally unimodular. (Given an undirected graph $G = (V, E)$, the incident matrix A is a $|V| \times |E|$ zero-one matrix where $a_{ij} = 1$ if and only if the i-th vertex and the j-th edge are incident.)

   (d) (6 points) Use results in (a), (b) and (c) to prove König-Egerváry Theorem.

   (e) (4 points) Give a counterexample to show that the claim in König-Egerváry Theorem fails if the graph is not bipartite.

(a) x_e with restriction $x_e \in \{0, 1\}$ indicates whether e is selected in the matching. Thus, $\sum_{e \in E} x_e$ is the number of edges in the matching for which we want to maximize, and $\forall v \in V : \sum_{e:e=(u,v)} x_e \leq 1$ says that at most one edge incident to v can be selected.

(b) The dual program is as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in V} y_v \\ \text{subject to} \quad & y_u + y_v \geq 1 && (\forall (u, v) \in E) \\ & y_v \geq 0 && (\forall v \in V) \end{aligned}$$

Here, y_v with restriction $y_v \in \{0, 1\}$ indicates whether v is selected in the vertex cover. $\sum_{v \in V} y_v$ is then the size of the vertex cover, and $y_u + y_v \geq 1$ says that one or two of u, v must be selected for each edge (u, v).

(c) For the base step, each 1×1 sub-matrix, which is an entry, is either 1 or 0, which has determinant either 1 or 0.

For the inductive step, suppose the determinant of any $k \times k$ sub-matrix is from $\{-1, 0, 1\}$. Consider any $(k+1) \times (k+1)$ sub-matrix T. Since each edge have exactly two endpoints, each column of T can contain at most two 1s.

If there is an all zero-column in T, we know $\det(T) = 0$, and we are done.

If there is a column in T contains only one 1, then $\det(T)$ equals to 1 or -1 multiplies the determinant of a $k \times k$ sub-matrix, which is from $\{-1, 0, 1\}$ by the induction hypothesis. Thus, $\det(T) \in \{-1, 0, 1\}$.

If every column in T contains two 1s, by the nature of bipartite graph, we can partition the rows into the upper half and the lower half such that each column of T contains exactly one 1 in the upper half and one 1 in the lower half. By multiplying 1 to the upper-half rows and -1 to the lower-half rows, and adding all of them, we will get a row of zeros. This means the set of all rows of T are linearly dependent, and $\det(T) = 0$.

We conclude the inductive step.

(d) It is easy to see that the coefficients in the constraints of the primal LP form exactly the incident matrix, and the coefficients in the constraints of the dual LP form the transpose of the incident matrix. Since this matrix is totally unimodular and the values at the right-hand side of the constraints in both linear programs are integers, both linear programs have integral optimal solutions.

In addition, it is straightforward to check that any primal LP solution with $x_e \geq 2$ cannot be feasible and any dual LP solution with $y_u \geq 2$ cannot be optimal. Therefore, there exists an primal LP optimal solution $\{x_e^*\}$ with $x_e^* \in \{0, 1\}$, and there exists an dual LP optimal solution $\{y_v^*\}$ with $y_v^* \in \{0, 1\}$. We have argued in (a) and (b) that both solutions can now represent a maximum matching and a minimum vertex cover respectively. By the LP-duality theorem, the primal LP objective value for the optimal solution $\{x_e^*\}$ equals to the dual LP objective value for the optimal solution $\{y_v^*\}$. This implies König-Egerváry Theorem.

(e) The simplest counterexample would be a triangle, where the maximum matching has size 1 and the minimum vertex cover has size 2.

4. (25 points) **[LP and Vertex Cover]** In this question, we are going to design a 2-approximation algorithm for the minimum vertex cover problem that is different from the one learned in the class. Let $G = (V, E)$ be an undirected graph for which we want to find a minimum vertex cover.

   (a) (5 points) Explain that the following linear program describes *the fractional version* of the vertex cover problem (i.e., replacing $x_u \geq 0$ to $x_u \in \{0, 1\}$ gives you the exact description of the minimum vertex cover problem).

$$\begin{aligned} \text{minimize} \quad & \sum_{u \in V} x_u \\ \text{subject to} \quad & x_u + x_v \geq 1 && (\forall (u, v) \in E) \\ & x_u \geq 0 && (\forall u \in V) \end{aligned}$$

   (b) (5 points) Let $\{x_u^*\}_{u \in V}$ be an optimal solution to the linear program in Part (a). Prove that $\sum_{u \in V} x_u^*$ is a lower bound to the size of any vertex cover.

   (c) (10 points) Consider the following algorithm.

   - Solve the linear program in Part (a) and obtain an optimal solution $\{x_u^*\}_{u \in V}$;
   - Return $S = \{u \in V \mid x_u^* \geq 0.5\}$.

   Prove that the algorithm returns a valid vertex cover, and it is a 2-approximation algorithm.

   (d) (5 points) Show that there exists an integral optimal solution to the linear program in Part (a) if G is a bipartite graph. Notice that an integral optimal solution to this linear program directly gives you a minimum vertex cover. (Hint: This almost straightforwardly follows from one of the previous questions you have solved. Can you see it?)

(a) See part (b) of Question 3.

(b) This is straightforward if we notice that a vertex cover can be described by $\{x_u \in \{0, 1\}\}_{u \in V}$ that is a feasible solution to the linear program.

(c) For any $(u, v) \in E$, since the LP constraint implies $x_u^* + x_v^* \geq 1$, at least one of x_u^* and x_v^* is greater than or equal to 0.5, which means at least one of u, v is in S by our algorithm. Therefore, S is a valid vertex cover.

On the other hand, the size of the vertex cover is given by

$$|S| = \sum_{u \in V : x_u^* \geq 0.5} 1 \leq \sum_{u \in V : x_u^* \geq 0.5} 2x_u^* \leq 2 \cdot \sum_{u \in V} x_u^*,$$

which implies the algorithm is a 2-approximation by part (b).

(d) This is directly implied by part (c) of Question 3.

5. (Bonus 60 points) **[Dinic's Algorithm]** In this question, we are going to work out *Dinic's algorithm* for computing a maximum flow. Similar to Edmonds-Karp algorithm, in each iteration of Dinic's algorithm, we update the flow f by increasing its value and then update the residual network G^f. However, in Dinic's algorithm, we push flow along *multiple* s-t paths in the residual network instead of a *single* s-t path as it is in Edmonds-Karp algorithm.

   In each iteration of the algorithm, we find the *level graph* of the residual network G^f. Given a graph G with a source vertex s, its level graph $\overline{G}$ is defined by removing edges from G such that only edges pointing from level i to level $i+1$ are kept, where vertices at level i are those vertices at distance i from the source s. An example of level graph is shown in Fig. 1.

   Next, the algorithm finds a *blocking flow* on the level graph $\overline{G}^f$ of the residual network G^f. A blocking flow f in G is a flow such that each s-t path contains at least one *critical edge*. Recall that an edge e is *critical* if the amount of flow on it reaches its capacity: $f(e) = c(e)$. Fig. 2 gives examples for blocking flow.

   Dinic's algorithm is then described as follows.

   1. Initialize f to be the empty flow, and $G^f = G$.
   2. Do the following until there is no s-t path in G^f:
      - construct the level graph $\overline{G}^f$ of G^f.
      - find a blocking flow on $\overline{G}^f$.
      - Update f by adding the blocking flow to it, and update G^f.

   Complete the analysis of Dinic's algorithm by solving the following questions.

   (a) (15 points) Prove that, after each iteration of Dinic's algorithm, the distance from s to t in G^f is increased by at least 1.

   (b) (15 points) Design an $O(|V| \cdot |E|)$ time algorithm to compute a blocking flow on a level graph.

   (c) (10 points) Show that the overall time complexity for Dinic's algorithm is $O(|V|^2 \cdot |E|)$.

   (d) (20 points) **(challenging)** We have seen in the class that the problem of finding a maximum matching on a bipartite graph can be converted to the maximum flow problem. Show that Dinic's algorithm applied to finding a maximum matching on a bipartite graph only requires time complexity $O(|E| \cdot \sqrt{|V|})$.

   (a) Firstly, each new edge appears in the next iteration of the residual network G^f can only be those pointing from level $t+1$ to level t for some integer t. These new edges

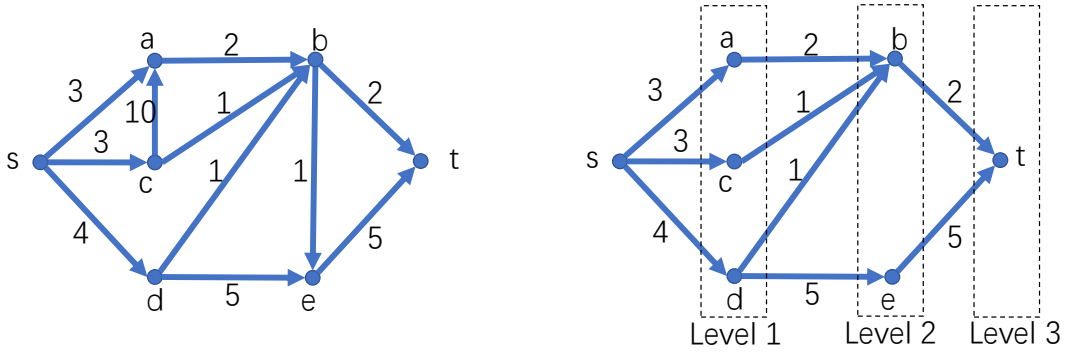

Figure 1: The graph shown on the right-hand side is the level graph of the graph on the left-hand side. Only edges pointing to the next levels are kept. For example, the edges (c, a) and (b, e) are removed, as they point at vertices at the same level. If there were edges pointing at previous levels, they should also be removed.

cannot help reduce the distance between s and t. Therefore, we see that the distance between s and t cannot decrease after each iteration.

By the nature of Dinic's algorithm, all the paths from s to t with length $\text{dist}(s, t)$ are blocked by critical edges after each iteration. In the next iteration, each s-t path must use at least one edge that is not in the level graph in the previous iteration. Each of such edges can only point from a vertex at level t to a vertex at a level that is at most t. Since all the shortest s-t paths must use at least one of such edges, the distance between s and t must be increased by at least 1.

(b) The algorithm iteratively finds a *maximal* path in $\overline{G}^f$ starting from s. In particular, when finding such a maximal path, at every vertex, the algorithm finds an arbitrary outgoing edge in $\overline{G}^f$ and appends it to the path. Two cases may happen: if the path eventually goes to t, we push as much flow as possible on this path, update the flow, and remove the critical edges (just as it is in Edmonds-Karp algorithm); if the path reaches a dead end u, we know that u is not in any s-t path, and we removing all the incoming edges of u. We do this until there is no more s-t paths in $\overline{G}^f$.

Each search of the maximal path requires $O(|V|)$ time, as each path has length $O(|V|)$. In addition, at least one edge is removed after each search of the maximal path (in the

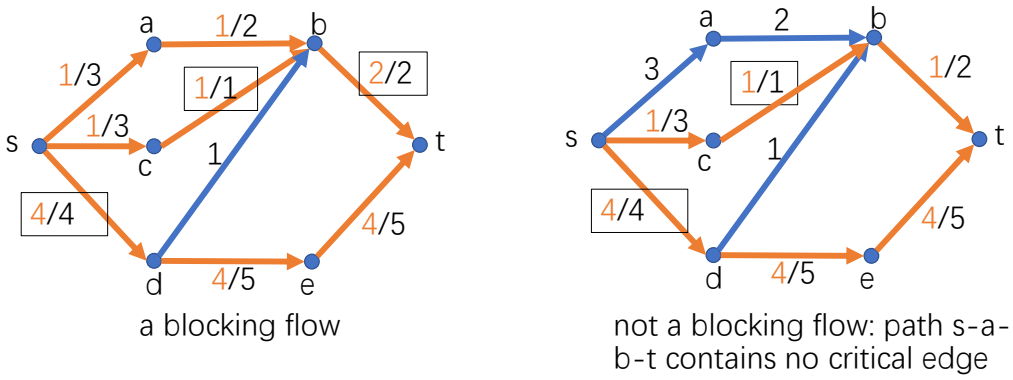

Figure 2: Blocking flow examples

first case, a critical edge is removed; in the second case, the edges going to the dead end are removed). There can be at most $O(|E|)$ searches. Therefore, the overall time complexity is $O(|V| \cdot |E|)$.

(c) In each iteration, the construction of the level graph requires a breadth-first search, which requires time complexity $O(|V| + |E|)$, the time complexity for finding a blocking flow is $O(|V| \cdot |E|)$ as shown in part (b), and the time complexity for updating f and G^f is $O(|E|)$. Therefore, each iteration takes $O(|V| \cdot |E|)$ time. There can be at most $O(|V|)$ iterations due to part (a). Thus, the overall time complexity for Dinic's algorithm is $O(|V|^2 \cdot |E|)$.

(d) The flow network is constructed similarly as it is in part (b) of question 2, *with the exception that each edge* (a, b) *for* $a \in A$ *and* $b \in B$ *has capacity* 1 *instead of* ∞. Therefore, in this flow network, all edges have capacity 1.

We first show that each iteration of Dinic's algorithm only requires $O(|E|)$ time. In each iteration, we perform a depth-first search starting from s. As before, two cases may happen. In the case we find a path from s to t, all the edges become critical after pushing 1 unit of flow along the path; we remove *all* the edges on the path and start over from s. Otherwise, during the DFS, whenever we reach a dead end and go backward, we can remove the edge just travelled (as this edge leads to a dead end and is not useful); in this case, we do not start over from s but continue the DFS process (until we find an s-t path). If we find a blocking flow in this way, each edge is travelled at most once.

Next, if the algorithm terminates before $\sqrt{|V|}$ iterations, the overall time complexity is $O(\sqrt{|V|} \cdot |E|)$ as we wished. Otherwise, we show that, *after* $\sqrt{|V|}$ *iterations, the maximum flow in* G^f *is at most* $\sqrt{|V|}$. Notice that this would imply the algorithm terminates within the next $\sqrt{|V|}$ iterations (as each iteration increases the value of the flow by at least 1), so we need a total of no more than $2\sqrt{|V|}$ iterations. This would also imply the overall time complexity is $O(\sqrt{|V|} \cdot |E|)$.

To see this, by part (a), after $\sqrt{|V|}$ iterations, the distance between s and t is more than $\sqrt{|V|}$. Consider a maximum *integral* flow in G^f (the integrality theorem guarantees that there exists a maximum integral flow). Since the capacities of all edges are 1, the flow consists of a union of multiple *edge-disjoint* s-t paths. Moreover, a crucial observation here is that, each vertex in G^f either has in-degree 1 or out-degree 1, which can be proved easily by induction. This observation guarantees that those edge-disjoint s-t paths are in fact *vertex-disjoint*. Since any of these vertex-disjoint paths has length more than $\sqrt{|V|}$, there can be at most $\sqrt{|V|}$ of them, which implies the maximum flow on G^f has value at most $\sqrt{|V|}$.