# Algorithm Design and Analysis
# Assignment 5
# Deadline: Jun 2, 2023

1. (35 points) **[Common System of Distinct Representatives]** Given a ground set $U = \{1, \dots, n\}$ and a collection of $k$ subsets $\mathcal{A} = \{A_1, \dots, A_k\}$, a *system of distinct representatives* of $\mathcal{A}$ is a "representative" collection $T$ of distinct elements from the sets in $\mathcal{A}$. Specifically, we have $|T| = k$, and the $k$ *distinct* elements in $T$ can be ordered as $u_1, \dots, u_k$ such that $u_i \in A_i$ for each $i = 1, \dots, k$. For example, $\{A_1 = \{2, 8\}, A_2 = \{8\}, A_3 = \{4, 5\}, A_4 = \{2, 4, 8\}\}$ has a system of distinct representatives $\{2, 4, 5, 8\}$ where $2 \in A_1, 4 \in A_4, 5 \in A_3, 8 \in A_2$, while $\{A_1 = \{2, 8\}, A_2 = \{8\}, A_3 = \{4, 8\}, A_4 = \{2, 4, 8\}\}$ does not have a system of distinct representatives.

   (a) (15 points) Design a polynomial time algorithm to decide if $\mathcal{A}$ has a system of distinct representatives.

   (b) (20 points) Given a ground set $U = \{1, \dots, n\}$ and two collections of $k$ subsets $\mathcal{A} = \{A_1, \dots, A_k\}$ and $\mathcal{B} = \{B_1, \dots, B_k\}$, a *common system of distinct representatives* is a collection $T$ of $k$ elements that is a system of distinct representatives of both $\mathcal{A}$ and $\mathcal{B}$. Design a polynomial time algorithm to decide if $\mathcal{A}$ and $\mathcal{B}$ have a common system of distinct representatives.

   For each part, prove the correctness of your algorithm, and analyze its time complexity.

   (a) Construct a directed graph $G = (V, E)$. $V = \{u_1, u_2, ..., u_n, a_1, a_2, ..., a_k\}$ and

   - $\forall i \in [1, n],\ (s, u_i, 1) \in E,$
     [$(s, u_i, 1)$ means an edge from $s$ to $u_i$ and its capacity is 1.]
   - $\forall i \in [1, k],\ (u_i, a_j) \in E$ if $i \in A_j,$
   - $\forall i \in [1, k],\ (a_j, t) \in E.$

   Compute the max flow of $G$ by Dinic's Algorithm and $\mathcal{A}$ has a system of distinct representatives if and only if the max flow equals to $k$.

   The capacities are all integer, so the correctness is given by **Flow Integrality Theorem**. For specific, the max flow is an integral flow. And if it is equal to k, then it corresponds to a system of distinct representatives.

   Complexity: $|V| = O(n)$, $|E| = O(nk) = O(n^2)$. We run Dinic on a bipartite graph, so the time complexity is $O(|E|\sqrt{|V|}) = O(n^2\sqrt{n})$.

   (b) Construct a directed graph $G(V, E)$.

   $V = (a_1, a_2, ..., a_k, u_1, u_2, ..., u_n, v_1, v_2, ..., v_n, b_1, b_2, ..., b_k)$, and

   - $\forall i \in [1, k]$, $(s, a_i, 1) \in E$, $(b_i, t, 1) \in E$,

- $\forall i \in [1, n]$, $(u_i, v_i, 1) \in E$,
- $\forall i \in [1, n]$, $(a_j, u_i, 1) \in E$ if $i \in A_j$, $(v_i, b_j, 1) \in E$ if $i \in B_j$.

Compute the max flow of $G$ by Dinic's Algorithm and $\mathcal{A}$ and $\mathcal{B}$ have a common system of distinct representatives if and only if the max flow equals to $k$.

Correctness: The capacities are all integer, so the correctness is given by **Flow Integrality Theorem**. For specific, the max flow is an integral flow. And if it is equal to k, then it corresponds to a common system of distinct representatives

Complexity: $|V| = O(n)$, $|E| = O(nk) = O(n^2)$. We run Dinic on a general graph, so the time complexity is $O(|V|^2|E|) = O(n^4)$.

2. (35 points) Consider the maximum flow problem $(G = (V, E), s, t, c)$ on graphs where the capacities for all edges are 1: $c(e) = 1$ for each $e \in E$. You can assume there is no pair of anti-parallel edges: for each pair of vertices $u, v \in V$, we cannot have both $(u, v) \in E$ and $(v, u) \in E$. You can also assume every vertex is reachable from $s$.

   (a) (20 points) Prove that Dinic's algorithm runs in $O(|E|^{3/2})$ time.

   (b) (15 points) Prove that Dinic's algorithm runs in $O(|V|^{2/3} \cdot |E|)$ time. (Hint: Let $f$ be the flow after $2|V|^{2/3}$ iterations of the algorithm. Let $D_i$ be the set of vertices at distance $i$ from $s$ in the residual network $G^f$. Prove that there exists $i$ such that $|D_i \cup D_{i+1}| \leq |V|^{1/3}$.)

(a) Finding a blocking flow in a level graph takes $O(|E|)$ time. Then we need to prove that number of iterations is $O(\sqrt{|E|})$.

If the algorithm terminates within $O(\sqrt{|E|})$ iterations, we are already done.

Otherwise, let $f$ be the flow after $\sqrt{|E|}$ iterations. Consider the maximum flow in $G^f$. If the maximum flow in $G^f$ is at most $\sqrt{|E|}$, then we are done.

By Integrality Theorem, there exists a maximum integral flow $f'$ in $G^f$. Then $f'$ consists of edge-disjoint paths with flow 1 since $c(e) = 1, \forall e \in E$.

By analysis to Dinic's algorithm, $\text{dist}^{G^f}(s, t) \geq \sqrt{|E|}$.

Then there are at most $\frac{|E|}{\sqrt{|E|}} = \sqrt{|E|}$ paths in $f'$ by edge-disjointness.

So we have $v(f') \leq \sqrt{|E|}$.

(b) Finding a blocking flow in a level graph takes $O(|E|)$ time. Then we need to prove that number of iterations is $O(|V|^{2/3})$.

If the algorithm terminates within $O(|V|^{2/3})$ iterations, we are already done.

Otherwise, let $f$ be the flow after $|V|^{2/3}$ iterations. Consider the maximum flow in $G^f$. If the maximum flow in $G^f$ is at most $|V|^{2/3}$, then we are done. By max-flow-min-cut theorem, we only need to prove the there exits a cut such that it is not greater than $|V|^{2/3}$.

Let $f$ be the flow after $2|V|^{2/3}$ iterations of the algorithm. Let $D_i$ be the set of vertices at distance $i$ from $s$ in the residual network $G^f$. We claim the there exits $i$ such that $|D_i| + |D_{i+1}| \leq |V|^{1/3}$.

Prove that by contradiction. Assume that $\forall i, |D_i| + |D_{i+1}| > |V|^{1/3}$, then we have

$$|V| \geq \sum_{i=0}^{\text{dist}^{G^f}(s,t)} |D_i| > \frac{1}{2}\text{dist}^{G^f}(s, t)|V|^{1/3}.$$

After $2|V|^{2/3}$ iterations, we also have $\text{dist}^{G^f}(s,t) \geq 2|V|^{2/3}$, and then

$$\frac{1}{2}\text{dist}^{G^f}(s,t)|V|^{1/3} \geq |V|,$$

Contradiction.

Suppose that $|D_k + D_{k+1}| \leq |V|^{1/3}$. We can find a cut $\{L, R\}$, where $L = \cup_{i=1}^{k} D_i$ and $R = V/L$. Then $(u, v, 1) \in \text{cut}\{L, R\}$ if $u \in D_k$ and $v \in D_{k+1}$. The number of edges in cut $\{L, R\}$ is no more than $|V|^{1/3} \times |V|^{1/3} = |V|^{2/3}$. So $c(L, R) \leq |V|^{2/3}$.

Q.E.D.

3. (35 points) 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) (5 points) Explain that the following is an LP-relaxation for the maximum matching problem.

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

   (b) (5 points) Write down the dual of the above linear program, and justify that the dual program is an LP-relaxation to the minimum vertex cover problem.

   (c) (10 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) (10 points) Use results in (a), (b) and (c) to prove König-Egerváry Theorem.

   (e) (5 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:

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

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 $1 \times 1$ 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. (35 points) The network flow problem only restricts the capacity of each edge. Consider the following variant, each edge $e$ does not only have a capacity $c_e$, but also a demand $d_e$. Finally, the edge should have flow $d_e \leq f_e \leq c_e$. Please find out how to solve the following problems by reducing them to the original max flow problem.

   (a) (20 points) In the original network flow problem, finding a feasible flow is easy. (a zero flow is surely feasible.) However, whether there exists a feasible flow is not straightforward in this new variant. How to determine the existence of a feasible flow by using the original max flow algorithm? (i.e., each $f_e$ should satisfy $d_e \leq f_e \leq c_e$ and the flow conservation constraint should hold at all vertices other than $s$ and $t$.)

   (b) (15 points) Based on the previous part, can we further find a maximum feasible flow? Please also use the original max flow algorithm.

Suppose we are given a graph $G = (V, E)$. First, without loss of generality, we can make the following assumptions: (1) the lower bound of every edge is non-negative (Otherwise, if $d_e < 0$, we can add a reversed edge if $c_e > 0$, or reverse edge $e$ if $c_e < 0$); (2) there is at most one edge between two vertices (Otherwise, we can merge the edges between two vertices into a single edge); (3) There do not exist anti-parallel edges between every two vertices.

Then, we will construct a new graph $G' = (V', E')$ in which each edge $e' \in E'$ has the same lower bound 0 and its own capacity defined as follows.

- Add two super vertices $s'$ and $t'$, and they are the new source node and target node of graph $G'$ respectively.
- For each vertex $v$, create an edge from vertex $s'$ to $v$, and the capacity of $(s', v)$ is set as $\sum_{u \in V} d_{(u,v)}$; create an edge from vertex $v$ to $t'$, and the capacity of $(v, t')$ is set as $\sum_{u \in V} d_{(v,u)}$.
- For each edge $e = (u, v) \in E$, create an edge $(u, v)$ in $G'$ and set the capacity of $(u, v)$ as $c_e - d_e$.
- Create an edge from the original target vertex $t$ to the original source vertex $s$ with capacity $\infty$.

Further, we run Fold-Fulkerson algorithm on this new graph $G'$ and get a maximum flow $f : E \to \mathbf{R}^*$ of graph $G'$. The following lemma implies that we can determine whether G is feasible according to the value of $f$.

**Lemma 1.** *The original graph $G$ has a feasible flow if and only if the value of the maximum flow in $G'$ equals to $\sum_{e \in E} d_e$.*

*Proof. Sufficiency.* If the original graph $G$ has a feasible flow $g : E \to \mathbf{R}$, consider the following flow $g' : E' \to \mathbf{R}^*$ on the new graph $G'$:

$$\begin{aligned}
&g'(e) = g(e) - d_e, \forall e \in E \\
&g'((t,s)) = |g| \\
&g'((s',v)) = \sum_{u \in V} d_{(u,v)}, \forall v \in V \\
&g'((v,t')) = \sum_{u \in V} d_{(v,u)}, \forall v \in V
\end{aligned}$$

Since $g$ is feasible, then $d_e < g(e) < c_e, \forall e \in E$. Thus, for each vertexv $v \in V'$ (other than $s, t$),

$$\begin{aligned}
\sum_{(u,v) \in E'} g'(v) &= g'((s',v)) + \sum_{(u,v) \in E', u \in V} g'(v) \\
&= \sum_{u \in V} d_{(u,v)} + \sum_{e=(u,v) \in E, u \in V} (g(e) - d_e) \\
&= \sum_{e=(u,v) \in E, u \in V} g(e) = \sum_{e=(v,u) \in E, u \in V} = \sum_{(v,u) \in E'} g'(v),
\end{aligned}$$

which demonstrates $g'$ satisfies flow conservation. Moreover, it is not hard to verify $g'$ satisfy the capacity restriction of G'. Since the sum of the capacity of the edges incident to $s'$ is $\sum_{e \in E} d_e$ and $\sum_{e=(s,u) \in E'} g'(e) = \sum_{e \in E} d_e$.

*Necessity.* If the maximum flow of $G'$ equals to $\sum_{e \in E} d_e$, it means each edge incident to $s'$ is saturated. Then we can create a new flow $f'$ on graph $G$, as follows:

$$f'(e) = f(e) + d_e, \forall e \in E.$$

It is also not hard to verify $f'$ is feasible, and satisfies flow conservation since $f$ satisfies flow conservation and $0 \le f(e) \le c_e - d_e$. □

Based on this lemma, we can check whether the maximum flow of $G'$ equals to $\sum_{e \in E} d_e$ to determine the existence of a feasible flow in $G$.

**Time complexity Analysis** The total time for constructing the graph $G'$ and running Fold-Fulkerson(Edmonds-Karp)is $O(|E|) + O(|V||E|^2) = O(|V||E|^2)$.

(b) Based on the algorithm above. we first check whether $G$ has a feasible flow. If $G$ has a feasible flow $f$, then we can construct a new graph $G' = (V, E')$ as follows. The vertex set of $G'$ is the same as G, but I add some reverse edges for $G'$. For each edge $e = (u,v) \in E$, create edges $e_1 = (u,v)$ and $e_2 = (v,u)$ for $E'$ and assign weights $c_e - f((u,v))$ to $e_1$ and $f((u,v)) - d_e$ to $e_2$.

We then run Edmonds-Karp algorithm on $G'$, and it outputs a maximum flow $g$. Further we can construct a flow $g'$ based on $g$, for each edge $e = (u, v)e \in E$, assign $g'((u, v)) = f((u, v)) + g((u, v)) - g((v, u))$.

Next, we argue that $g'$ is the maximum feasible flow of graph $G$:

- Since $0 \leq g((u, v)) \leq c_{(u,v)} - f((u, v))$ and $0 \leq g((v, u)) \leq f((u, v)) - d_{u,v})$ for each edge $e = (u, v) \in E$,then $d_e \leq g'((u, v)) \leq c_e$. Thus, $g'$ is a feasible flow.
- Since $G'$ can be seen as the residual Network of the original graph $G$ after pushing a flow $f$. For this reason, the value of the maximum flow of $G$ is equal to the value of the maximum flow of $G'$ plus $|f|$. On the other hand, since the value of $g'$ is just equal to $|g|$ plus $|f|$, we can concludes that $g'$ is indeed the maximum flow of $G$.

According to the above explanation, we can find a maximum feasible flow based on the algorithm in (a) within $O(|V||E|^2)$ time.

5. How long does it take you to finish the assignment (including thinking and discussing)? Give a score (1,2,3,4,5) to the difficulty. Do you have any collaborators? Write down their names here.