CS535 Fall 2022 HW2 Sample Solutions

1. Suppose $|M| < |N|$ and $|N| = |M| + k$. $M \bigoplus N$ consists of alternating paths and circuits, within which there are at least k M-augmenting paths.

   If we assign in such a path matched edges originally from M to N and matched edges originally from N to M, the size difference of M and N is reduced by 2, therefore we "flip" $\lfloor k/2 \rfloor$ M-augmenting paths.

   Algorithm steps:

   (1) Compute $G = M \bigoplus N$.

   (2) Repeatedly find M-augmenting paths in G and flip them for $\lfloor k/2 \rfloor$ times.

   G can be computed in $O(|M| + |N|)$, and going through $\lfloor k/2 \rfloor$ takes $O(|M| + |N|)$.

2. A Hall set $T \subseteq U$ satisfies that

$$T = \operatorname*{argmin}_{S \subseteq U} |N(S)| - |S| = \operatorname*{argmax}_{S \subseteq U} |S| - |N(S)| = \operatorname*{argmax}_{S \subseteq U} |S| + (|W| - |N(S)|)$$

   Given any $S \subseteq U$, $S \cup (W \setminus N(S))$ is an independent set. Thus $T \cup (W \setminus N(T))$ is a max independent set, for otherwise, given a max independent set I, let $T' = I \cap U$, we must have

$$\begin{aligned} & |I| = |T'| + (|W| - |N(T')|) > |T| + (|W| - |N(T)|) \\ \implies & |T'| - |N(T')| > |T| - |N(T)| \\ \implies & |N(T')| - |T'| < |N(T)| - |T| \end{aligned}$$

   contradicting the fact that T is a Hall set. Therefore we can compute a max independent set I and $I \cap U$ is a Hall set.

   In any graph, the complement of a min vertex cover is a max independent set. We can compute a min vertex cover in a bipartite graph from a max matching. Denote by U_M the unmatched vertices in U, denote by R_M the vertices reachable from U_M by alternating paths, the set $(U \setminus R_M) \cup (W \cap R_M)$ is a min vertex cover. The correctness follows from König's theorem. R_M can be computed by a linear-time search of alternating paths from U_M using DFS or BFS.

3. We compute the inessential vertices first, and then subtract them from M to get the essential vertices.

   A vertex is inessential if there exists a max matching that doesn't cover it. We start from M to discover all such vertices. For all unmatched vertex v, without loss of generality, suppose $v \in U$. Start from v to discover all vertices reachable from v via alternating paths. Every vertex $u \in U$ reachable from v is inessential, for there exists a $v - u$ alternating even path p. If we flip the matched and unmatched edges on p, we get a new max matching that covers v and doesn't cover u, proving that all such u is not essential.

   Repeat this process starting from all unmatched vertices in M without revisiting already discovered edges, we can mark all inessential vertices in M. This can be done by BFS or DFS in linear time. The remaining unmarked vertices in M are essential.

4. From the directed graph $D = (V, A; w)$, we use the node-splitting technique to construct a undirected bipartite graph $G = (V, V', A'; w')$, where

$\forall v \in V$, create a copy $v' \in V'$
$A' = \{(v, u') \text{ for } (v, u) \in A\} \cup \{(v, v') \text{ for } v \in V\}$,
$w'((v, u')) = w((v, u))$ for $(v, u) \in A$, and
$w'((v, v')) = 0$ for $v \in V$.

In G, all self-edges form a perfect matching M_0 of 0 weight. According to

- circuit C in D $\Leftrightarrow$ M_0-alternating circuit C^+ in G
- $M_0 \bigoplus C^+$ is a perfect matching with weight $w(C)$

Therefore, finding a collection of vertex-disjoint circuits in D whose total edge weight is maximum is equal to finding the max-weight perfect matching $M_0 \bigoplus C^+$ in G.

Hence, we can apply the Hungarian algorithm to find the maximum cost perfect matching $M_0 \bigoplus C^+$ for G. Then convert C^+ from G back to a collection of circuits of the original D. Since the node splitting takes linear time, the total time complexity equals to the time complexity of Hungarian algorithm which is $O(|V|(|A| + |V|log|V|))$.

5. Suppose machine i has a job sequence $J_i = j_1, j_2, \cdots j_l$, the sum of the finishing time is

$$\sum_{k=1}^{l} = (l - k + 1) \cdot p_{i_k}$$

The intuition is that if a job is scheduled as the k^{th} last job, it contributes k times its processing time to the total finishing time on that machine.

Each machine may contain up to n jobs. We build a bipartite graph $G = (U, W; E)$. U contains n vertices $u_1 \cdots u_n$ for job $1 \cdots n$. W contains nm vertices representing each possible position on each machine. More precisely, vertex $w_{ij} \in W (1 \leq i \leq m, 1 \leq j \leq n)$ represents the j^{th} last position on machine i. E are edges linking every job with every position, with length

$$l(u_k, w_{ij}) = j \cdot p_{ij}, (1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq n)$$

as the contribution to the total finishing time for job k scheduled at the j^{th} last position on machine i.

We first construct a max matching with edges $(u_j, w_{1j}), 1 \leq j \leq n$, scheduling all the jobs on machine 1 in the input job order. The weight of this matching represents the current total finishing time. Now find a minimum weighted matching with size n and the output will provide a scheduling with the lowest total finishing time.

6. (a) Since it is a bipartite graph, the total edge weight of all vertices in U is equal to that in W.

$$\sum_{v \in U} \sum_{e \in v} l(e) = \sum_{v \in W} \sum_{e \in v} l(e)$$
$$\Longrightarrow \sum_{v \in U} 1 = \sum_{v \in W} 1$$
$$\Longrightarrow |U| = |W|$$

$\forall S \subseteq U$, the total edge weight of $N(S)$ is at least that of S.

$$\sum_{v \in S} \sum_{e \in v} l(e) \leq \sum_{v \in N(S)} \sum_{e \in v} l(e)$$
$$\Longrightarrow \sum_{v \in S} 1 \leq \sum_{v \in N(S)} 1$$
$$\Longrightarrow |S| \leq |N(S)|$$

Therefore, G has a matching covering U, G has a perfect matching.

(b) We find the first perfect matching M in no longer than $O(m^2)$. For each edge in M reduce the weight by $\min_{e \in M_1} l(e)$. After doing this at least one edge disappears from M. For each disappeared edge $e(u, v)$, start from M and search for an augmenting path from u to v in $O(m)$. Whenever the current M is augmented to size n, i.e. a perfect matching, reduce the weights on all edges in M. Repeat until the entire graph has no more edges.

Each time you reduce the edge weights in the current matching, you essentially decompose and get one perfect matching with its weight. After each reduction, the total weight of edges incident to any vertex in the graph is still equivalent, meaning that there is still a perfect matching in the graph, making sure your next iteration can be computed.

Each augmented path removes at least one edge, and the last augmenting path removes exactly n last edges, so you repeat augmenting at most $m - n$ times with $O(m^2)$ in total, and you get at most $m - n$ perfect matchings. Adding the first perfect matching, you get a decomposition of size no larger than $m - n + 1$. In each perfect matching you reduce the edge weights n times, $O(nm)$ in total. The whole process can be done in $O(m^2)$.