Network Algorithms (CS 541) **Fall 1995**

Topic 18: Optimal Assignment

Lecturer: Subhash Suri *Scribe: Stephen Scott*

1 Introduction

In earlier lectures we discussed the assignment problem, i.e. given a bipartite graph $G = (X, Y, E)$ (all edges $(x, y) \in E$ are such that $x \in X$ and $y \in Y$), find a matching of maximum cardinality. This matching represents the largest assignment of nodes in X (e.g. workers) to nodes in Y (e.g. jobs). Now we generalize the problem to include weights on the edges where $w(x, y)$ might represent worker x's competence at job y. The goal now is to find a perfect matching M maximizing $\sum_{(x,y)\in M} w(x, y)$ with the assumption that a perfect matching exists. If a perfect matching does not exist, additional edges with weight 0 can be added.

We start by reviewing the Hungarian method for finding a maximum cardinality matching. Then we describe an extension of the Hungarian method (called the Kuhn-Munkres algorithm) to find an optimal assignment when weighted edges are used.

2 Review of the Hungarian Method

Recall from last lecture the Hungarian method for finding a perfect matching, given in Figure 1.

Hungarian(G, M) $\qquad\triangleright$ M is some matching in G
1 **if** M is a perfect matching
2   **then return** M
3   **else** $u \leftarrow$ a vertex in X unmatched in M
4   Build an alternating tree T from u
5   **if** an unmatched vertex $v \in Y$ is found when building T
6   **then** use the path from u to v to augment M, yielding M' $\qquad\triangleright$ $|M'| > |M|$
7   **return** Hungarian(G, M')
8   **else** $S \leftarrow T \cap X$ $\qquad\triangleright$ $N(S) \subseteq Y$, $|N(S)| = |S| - 1 < |S|$
9   **return** S as evidence that no perfect matching exists

Figure 1: Pseudocode for the Hungarian method.

If a maximum cardinality matching is desired, the above algorithm can be modified to run lines 4–7 on every unmatched vertex in X. This algorithm takes time $O(|E|)$ to build the tree, a process done $O(|X|)$ times, so the time complexity of the Hungarian method is $O(|X| \cdot |E|)$. Note that there is an implementation that runs in time $O\left(\sqrt{|X|} \cdot |E|\right)$, which matches the bound of $O\left(\sqrt{|V|} \cdot |E|\right)$ obtained by reducing the problem to maximum flow and applying Dinic's algorithm. This is the best known time bound on bipartite matching.

3 The Kuhn-Munkres Algorithm

Unfortunately the Hungarian method as it stands does not work for the optimal assignment problem. In this section we describe a variation of the Hungarian method called the *Kuhn-Munkres algorithm.* More detail on this algorithm can be found in *Graph Theory with Applications* by Bondy and Murty.

3.1 Basic Terminology

In this algorithm we will assign a label $\ell(v)$ to all nodes $v \in X \cup Y$. We say that a labeling is *feasible* iff $\ell(u) + \ell(v) \geq w(u, v)$ for all $(u, v) \in X \times Y$. This is similar to the distance labels used in the shortest path algorithms which required $\ell(v) \leq \ell(u) + w(u, v)$ for all $(u, v) \in V \times V$. Note that there always exists a feasible labeling of a bipartite graph, namely $\ell(v) = 0$ for all $v \in Y$ and $\ell(u) = \max_{v \in Y}\{w(u, v)\}$ for all $u \in X$. An example of a feasible labeling is in Figure 2. Each node's label lies within that node.

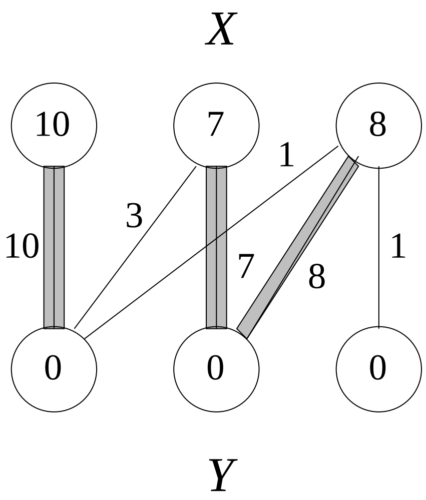

Figure 2: An example of a feasible labeling and an equality subgraph.

An *equality subgraph* $G_= = (X, Y, E_=)$ is identical to G except that $E_= = \{(u, v) \in E \mid \ell(u) + \ell(v) = w(u, v)\}$. In Figure 2, the edges of $E_=$ are shaded. The correctness of the Kuhn-Munkres algorithm depends on the following property of equality subgraphs.

Theorem 1 *If there exists a perfect matching M^* in the equality subgraph $G_=$, then M^* is an optimal assignment in G.*

Proof: *Define $w(M^*) = \sum_{(i,j) \in M^*} w(i, j) = \sum_{(i,j) \in M^*} (\ell(i) + \ell(j)) = \sum_{v \in X \cup Y} \ell(v)$. The second equality holds because M^* is a matching in the equality subgraph. The third equality holds because M^* is a matching and thus contains node-disjoint edges. Now say there is another matching M with weight greater than M^*'s. Then $w(M) = \sum_{(i,j) \in M} w(i, j) > w(M^*) = \sum_{v \in X \cup Y} \ell(v)$. But $\sum_{(i,j) \in M} w(i, j) \leq \sum_{(i,j) \in M} (\ell(i) + \ell(j))$ because the labeling is feasible. Also, $\sum_{(i,j) \in M} (\ell(i) + \ell(j)) \leq \sum_{v \in X \cup Y} \ell(v)$ because M is a matching and thus must contain node-disjoint edges. This establishes a contradiction, so M^* is an optimal assignment in G.* ■

3.2 The Algorithm

Thanks to Thoerem 1, we can run the Hungarian algorithm on $G_=$ to find a perfect matching and thus an optimal assignment. However, a perfect matching may not exist in $G_=$, so we need to update $G_=$ if this occurs. The algorithm is in Figure 3.

Kuhn-Munkres(G, ℓ) $\triangleright$ ℓ is a feasible labeling on G

1 $G_= \leftarrow$ the equality subgraph induced by ℓ

2 $M \leftarrow$ any matching in $G_=$

3 **if** M is a perfect matching

4   **then return** M $\triangleright$ M is an optimal assignment by Theorem 1

5   **else** let $u \in X$ be an unmatched vertex in M

6   **while** $N(S) \neq T$ $\triangleright$ S is the set returned by Hungarian when

7   **do** grow the alternating tree T $\triangleright$ the graph has no perfect matching

8   **if** a new matching M' is found

9   **then** $M \leftarrow M'$

10   **goto** 3

$\triangleright$ $N(S) = T$ at this point

11     $\alpha_\ell \leftarrow \min\limits_{\substack{x \in S \\ y \notin T}} \{\ell(x) + \ell(y) - w(x, y)\}$

12   For all $v \in X \cup Y$, $\overline{\ell}(v) \leftarrow \begin{cases} \ell(v) - \alpha_\ell & \text{if } v \in S \\ \ell(v) + \alpha_\ell & \text{if } v \in T \\ \ell(v) & \text{otherwise} \end{cases}$

13   **return** Kuhn-Munkres$(G, \overline{\ell})$

Figure 3: Pseudocode for the Kuhn-Munkres algorithm.

In Figure 3, line 11 finds an edge not yet in $G_=$ and line 12 creates a new labeling $\overline{\ell}$ that will bring that edge into $G_=$ on the recursive call. This process guarantees that all edges of G will be added if necessary and they will be added in the order that decreases the sum of all labels (and thus the matching's total weight) as little as possible at each step. To see this, note first that once an edge enters $G_=$ it never leaves because if an edge (x, y) was in $G_=$, then either $x \in S$ and $y \in T$ or $x \notin S$ and $y \notin T$. In the latter case, $\overline{\ell}(x) = \ell(x)$ and $\overline{\ell}(y) = \ell(y)$. In the former case, $\ell(x) + \ell(y) = w(x, y)$, so $\overline{\ell}(x) + \overline{\ell}(y) = \ell(x) - \alpha_\ell + \ell(y) + \alpha_\ell = \ell(x) + \ell(y) = w(x, y)$. Also note that for the x and y that give α_ℓ in line 11, edge (x, y) will be added to $G_=$ on the recursive call because $\overline{\ell}(x) = \ell(x) - \ell(x) - \ell(y) + w(x, y)$ (since $x \in S$) and $\overline{\ell}(y) = \ell(y)$ (since $y \notin T$), implying $\overline{\ell}(x) + \overline{\ell}(y) = \ell(x) - \ell(x) - \ell(y) + w(x, y) + \ell(y) = w(x, y)$. Finally, the way the labels are modified guarantees that $\overline{\ell}$ will be feasible.

Because of the order that edges are added to $G_=$ and because all edges are added if necessary, a perfect matching will eventually be found in $G_=$ since we assumed one existed in G. This shows the algorithm's correctness.

The number of executions of lines 11–13 is $\leq |Y|$ because the new edge (x, y) added to $G_=$ will cause y to appear in T the next time around. After each recursive call, the tree will need to be rebuilt, requiring $O(|X|^2)$ time, so the time complexity is $O(|X|^3)$.