# IEOR 8100: Matchings
# Lecture 6: The Hungarian Algorithm

Let $G = (V, E)$ be a bipartite and weighted graph, with $|V| = n$ and $|E| = m$. The weight of edge $(i, j)$ is denoted by $c_{ij}$. As $G$ is bipartite, $V$ can be divided into two non-overlapping sets $A$ and $B$ such that there are no edges with both endpoints in $A$ and no edges with both endpoints in $B$. We shall describe an algorithm which finds a min-weight perfect matching in $G$. Note that we may assume WLOG that $G$ is complete- if needed, we can add edges with a large enough weight. We will also assume that the graph has a perfect matching. Finally, we will assume, as usual, that the data is integral.

## 1 LP Formulation and The Dual

The LP associated with the min-weight perfect matching problem (on a bipartite graph) is:

$$\begin{array}{lccll} \min & \sum_{i\in A, j\in B} c_{ij}x_{ij} & & & \\ s.t. & \sum_{j\in B} x_{ij} & = & 1 & \forall i \in A \\ & \sum_{i\in A} x_{ij} & = & 1 & \forall j \in B \\ & x_{ij} & \geq & 0 & \forall (i,j) \in E \end{array}$$

Perfect matchings in $G$ are in a 1-1 and onto correspondence with feasible binary vectors $x$ of this problem. Thus, when we refer to a vector $x$ as a matching we mean that it is the binary vector associated with that matching. Furthermore, remember that we have proven that for bipartite graphs, the extreme points of the feasible set of the LP above are exactly all perfect matchings. In particular, this implies that there is an optimal solution to this LP which occurs in a perfect matching vector, and since this LP is a relaxation of the min-weight perfect matching (bipartite) problem, it follows that this vector corresponds to an optimal perfect matching. Thus, all we have to do is to find that matching.

The dual of the LP above is:

$$\begin{array}{lccccll} \max & \sum_{i\in A} u_i & + & \sum_{j\in B} v_j & & & \\ s.t. & u_i & + & v_j & \leq & c_{ij} & \forall (i,j) \in E \end{array}$$

Define the reduced cost vector $w$ by $w_{ij} = c_{ij} - u_i - v_j \ \forall (i, j) \in E$. By complementary slackness, if we can find a perfect matching $M$ (with associated

binary vector $x$) and vectors $u$ and $v$ s.t. $w_{ij} \geq 0\ \forall (i,j) \in E$ and $w_{ij} = 0$ $\forall (i,j) \in M$, we would have an optimal solution. This is exactly the path our algorithm takes.

# 2 The Hungarian Algorithm

The algorithm we present now is called the Hungarian algorithm, and it solves the min-weight perfect bipartite matching problem. It operates by maintaining a feasible dual solution and a (generally infeasible) primal solution of the form of a (generally non-perfect) matching, while making sure that they satisfy complementary slackness. The algorithm ends when the primal solution it has is feasible, namely the matching it has is perfect. Note that the algorithm makes use of the unweighted maximal bipartite matching algorithm we developed in lecture 5 as a procedure; $L$ is defined and obtained the same way as in lecture 4. We define $G' = (V, E')$ to be the subgraph of $G$ which includes only the edges with reduced cost 0.

**Algorithm 1** Hungarian Algorithm

$u_i \leftarrow 0\ \forall i$
$v_j \leftarrow \min_i c_{ij}\ \forall j$
$w_{ij} = c_{ij} - u_i - v_j\ \forall (i,j) \in E$
$found \leftarrow false$
**while** $!found$ **do**
  Run Bipartite-Matching(G'), store the returned matching in $M$ and store $L$ as defined in lecture 4
  **if** $M$ is perfect **then**
    $found \leftarrow true$
  **else**
    $\delta \leftarrow \min_{i \in A \cap L \wedge j \in B \setminus L} w_{ij}$
    $u_i \leftarrow u_i + \delta\ \forall i \in A \cap L$
    $v_j \leftarrow v_j - \delta\ \forall j \in B \cap L$
  **end if**
**end while**
**return** $M$

## 2.1 Correctness

For correctness, we need to show the following:

1. The dual solution is always feasible. Note that $u, v$ are clearly dual feasible in the beginning. Now, assume that $u, v$ are dual feasible in the beginning of some iteration. They will only be changed during that iteration if there is no perfect matching in $G'$. Feasibility simply means nonnegativity of reduced costs for all edges. Note that the only reduced costs that decrease are for edges $(i, j)$ where $i \in A \cap L$ and $j \in B \backslash L$. However, note that the reduced costs of those edges decrease by exactly $\delta$, and by definition of $\delta$ equals the minimum of them; thus, none of them drop below 0 and dual feasibility is maintained.

2. Complementary slackness is maintained. Trivial: note that the matching is found only along edges with zero reduced cost. (Note that this is true at the point in iteration immediately after $M$ is computed, which is what we need since this is the point where we will eventually exit the loop).

3. The algorithm reaches an ending. First, note that $\delta$ always stores a strictly positive number. If $\delta$ gets the value 0 at some point, then there is an edge $(i, j)$ for which $i \in A \cap L$, $j \in B \backslash L$, and $w_{ij} = 0$; however, since $w_{ij} = 0$, then $(i, j) \in E'$, and $j$ is reachable from $A \cap L$ or $i$ was reached via $(i, j)$ (if $(i, j) \in M$), so $j \in L$, which is not possible. Next, recall that after we run the bipartite (unweighted) matching algorithm, $C = (A \backslash L) \cup (B \cap L)$ is a vertex cover of the same cardinality as the matching $M$ we find. Note that in every iteration, the dual increase is $\delta|A \cap L| - \delta|B \cap L| = \delta(|A \cap L| + |A \backslash L| - |A \backslash L| - |B \cap L|) = \delta(|A| - |C|) = \delta(\frac{n}{2} - |M|) \geq \delta$ where we get $\frac{n}{2} - |M| \geq 1$ from the fact that $M$ is not perfect. Thus, each iteration increases the dual by at least $\delta$, which is at least 1 since the data is integral; since the primal problem is feasible (there exists a perfect matching in the original graph), the dual problem is bounded, and so the dual solution value can't increase forever and the algorithm must terminate.

Now, since clearly the algorithm ends with a perfect matching (i.e. a primal feasible solution), the above discussion shows that we also have a dual feasible solution and that those two solutions maintain complementary slackness. Therefore, we have shown that the algorithm ends in finite time and correctly.

## 2.2 Running Time

The key fact for running time analysis is the following: every iteration that does not increase the cardinality of the matching, increases the cardinality of $L$.

Let us prove this fact: Consider two subsequent iterations (call them iterations I and II) inbetween which $|M|$ does not increase. First, remember that

we have already shown that the reduced costs never drop below 0. Note that the only reduce costs that increase are of edges from $A \backslash L$ to $B \cap L$ (where $L$ is as computed during iteration I). Also note that by definition of $\delta$, and since we have shown that $\delta$ is always strictly positive, at least one additional edge from $A \cap L$ to $B \backslash L$ gets reduced cost zero.

In conclusion, the way $G'$ changes in between the iterations is an addition of at least one edge between $A \cap L$ and $B \backslash L$, which trivially was not part of $M$ found in iteration I, and (potentially) loss of edges between $A \backslash L$ and $B \cap L$. The edges lost are unmatched during iteration I (otherwise, since its $B$-endpoint is in $L$, the algorithm would've continued to add its $A$-endpoint to $L$), and are not used in the very last "BFS" of the matching procedure of iteration I (since otherwise clearly both its endpoints would be in $L$). Now, in iteration II, by assumption, $M$ remains a maximum matching; assuming that the search procedure finds it, then in the very last BFS, as discussed, the edges lost wouldn't have had to be used anyway and so no nodes are removed from $L$; on the other hand, the additional edge from $A \cap L$ to $B \backslash L$ would remain unmatched and hence used to add its $B$ endpoint to $L$, increasing the size of $L$ by at least one. So our proof is done.

Using the fact we've just proven, we have that every $n$ iterations, the size of $M$ increases by at least 1, and so we get a bound of at most $\frac{n}{2} \cdot n$ iterations. Each iteration's running time is dominated by the perfect matching procedure, which takes $O(\sqrt{n}m)$. As we have shown, this takes an overall of $O(n^{2.5}m)$ time. Note that if we were more careful, we could've got a running time as low as $O(mn \lg n)$.