The Hungarian Algorithm

Zachary Campbell

January 31, 2018

1 Maximum cardinality matching problem.

In this problem we are given a bipartite graph $G = (S, T, E)$ and our goal is to find a matching M of maximum size. That is, for any other matching $M^{'}$ on G, we want that $|M| \geq |M^{'}|$. We begin with some basic definitions.

Definition . Let $G = (S, T, E)$ be a bipartite graph, and $M \subseteq E$ a matching on G. We define an *alternating tree* relative to M to be a tree which satisfies the following:

1. the tree contains exactly one node $s \in S$ that is exposed w.r.t. M, which we call the *root*;
2. all paths from the root s to any other node in the alternating tree are alternating paths.

The algorithm we describe initializes with the empty matching. It then successively makes exposed nodes in S the root of an alternating tree, to which it then adds edges and nodes. At some point in the computation, one of two things will occur. Either an exposed node in T is added to the alternating trees, or else it is no longer possible to add more nodes and edges to any of the alternating trees. In the first case we just augment the matching with our new alternating trees. In the second case, the trees are what we call *Hungarian*, and our matching will be maximal. The Hungarian trees will actually do more than just indicate that our matching is maximal. In order to understand exactly what these trees tell us, we will need to rephrase the problem a bit. We will approach this problem from a primal dual perspective. The primal problem is given by the linear program

$$\begin{aligned} \text{maximize } & \sum_{i,j} x_{ij}, \\ \text{subject to } & \sum_{j} x_{ij} \leq 1, \\ & \sum_{i} x_{ij} \leq 1, \\ & x_{ij} \geq 0. \end{aligned}$$

The variables x_{ij} are assigned to edges of the graph, and take on values in the range $[0, 1]$. The dual is then

$$\begin{aligned} \text{minimize } & \sum_i u_i + \sum_j v_j, \\ \text{subject to } & u_i + v_j \geq 1, \\ & u_i \geq 0, \\ & v_j \geq 0. \end{aligned}$$

It turns out that solving this dual is equivalent to solving vertex cover on our graph. It tells us that each edge must be covered by vertices whose values add up to at least 1. In the case where we restrict to integers, this says that for an edge (i, j), either i or j must be included in our solution. Moreover, when we've constructed our Hungarian trees, we can use them to construct this dual solution by taking all out-of-tree nodes in S and all in-tree nodes in T.
We want our algorithm for maximum cardinality matching to maintain primal dual feasibility at all steps in the computation.
We give the following algorithm for this.

- *Step 0* (*Initialization*) Given a bipartite graph $G = (S, T, E)$. Let our matching $M := \emptyset$. No nodes are labeled.
- *Step 1* (*Labeling*)
  - (1.0) For each exposed node in S, we give the label "$\emptyset$".
  - (1.1) If there are no unscanned labels, go to *Step 3*. Otherwise, find a node i with an unscanned label. If $i \in S$, go to (1.2); if $i \in T$, go to (1.3).
  - (1.2) Scan $i \in S$ as follows. For each arc $(i, j) \notin M$ incident to i, give node j the label "i," unless j already has a label. Return to (1.1).
  - (1.3) Scan $i \in T$ as follows. If i is exposed, we have found an augmenting path, go to *Step 2*. Otherwise, identify the unique $(i, j) \in M$ incident to i and give j the label "i." Return to (1.1).
- *Step 2* (*Augmentation*) An augmenting path p that terminates at i has been found. Backtrack the labels until you reach the label "$\emptyset$." Set $M := M \oplus p$. Remove all labels from nodes and return to (1.0)
- *Step 3* (*Hungarian labeling*) The labeling is Hungarian, no augmenting paths exist, and M is a maximum cardinality matching. Let $L \subseteq S \cup T$ denote the set of labeled nodes. Then $K := (S \setminus L) \cup (T \cap L)$ is a minimum cardinality vertex cover.

It is clear that this algorithm outputs a maximum cardinality matching; as with many algorithms for this problem, it runs until there are no more augmenting paths, which occurs exactly when no more edges can be added to the matching. However, it is not immediately clear why this K should be a minimum vertex cover. We will prove this, using the following.

Lemma . Let $G = (S, T, E)$ be a bipartite graph. Let M be a matching of G, and C a covering of G such that $|M| = |C|$. Then M is a maximum matching and C is a minimum covering.

Proof. Let M' be a minimum matching on G, and C' a maximum covering on G. For eah $(i, j) \in M'$, C' must include either i or j, which tells us that $|M'| \leq |C'|$. Then we have

$$|M| \leq |M'| \leq |C'| \leq |C|.$$

Thus, if $|M| = |C|$, we have equalities above, which means that the size of a maximum matching is equal to the size of a minimum covering. $\square$

Theorem (Kőnig-Egervary). For any bipartite graph, the maximum number of arcs in a matching is equal to the minimum number of nodes in a vertex cover.

Proof. Let $G = (S, T, E)$ be a bipartite graph, and let M be a maximum matching of G. Furthermore, define

$$A := \{s \in S \mid s \text{ unsaturated}\}$$

and

$$B := \{ \text{ all vertices connected to nodes in } A \text{ by alternating paths } \}.$$

Let $L = B \cap S$ and $R = B \cap T$. Then we have the following:

1. Every node in R is saturated,
2. $N(L) = R$,

where $N(L)$ denotes the "neighbors" of the L (all the things that are connected to elements of L). The first comes from the fact that, if M is a maximum matching, then our alternating paths starting at nodes in A must have length ≥ 2, and have even length (otherwise we would have an augmenting path, which contradicts our assumption that M is a maximum matching).
The second comes from the fact that every node in $N(L)$ is connected to vertices in A by an alternating path. Now, define $K := (S \setminus L) \cup R$. Every edge in G must have at least one of its nodes in K. If not, there would be an edge with one end in L and one end in $T \setminus R$, which contradicts $N(L) = R$. So K is a covering of G. Moreover, $|K| = |M|$, since for each edge in M we've included one of its nodes in K (the vertices we've chosen are those in $N(L)$ and those in $S \setminus L$). Thus, by the previous lemma, K is a minimum covering. $\square$

Looking back, our algorithm constructs this very K during its labeling procedure. We will use the basic intuition of this algorithm to construct an algorithm for maximum weight matching.

2 Maximum weight matching - the Hungarian algorithm

The setting is that we have a bipartite graph $G = (S, T, E)$ with weighted edges, and we want to find a maximum-weight perfect matching on this graph.

For each $(i, j) \in E$ we have a weight w_{ij}. We use the variable x_{ij} to indicate whether or not we are including (i, j) in our matching. We formulate a linear program for the maximum-weight perfect matching problem as follows:

$$\begin{aligned} \text{maximize } & \sum_{i,j} w_{ij} x_{ij}, \\ \text{subject to } & \sum_{j} x_{ij} \leq 1, \\ & \sum_{i} x_{ij} \leq 1, \\ & x_{ij} \geq 0. \end{aligned}$$

Constructing the dual is straightforward: we have a variable u_i for each $\sum_j x_{ij}$ and a variable v_j for eah $\sum_i x_{ij}$. So the dual is:

$$\begin{aligned} \text{minimize } & \sum_i u_i + \sum_j v_j, \\ \text{subject to } & u_i + v_j \geq w_{ij}, \\ & u_i \geq 0, \\ & v_j \geq 0. \end{aligned}$$

It follows that satisfying the following conditions is necessary and sufficient for optimality of primal and dual solutions:

$$x_{ij} > 0 \implies u_i + v_j = w_{ij}, \tag{1}$$

$$u_i > 0 \implies \sum_j x_{ij} = 1, \tag{2}$$

$$v_j > 0 \implies \sum_i x_{ij} = 1. \tag{3}$$

This tells us that for an edge (i,j) with nonzero x_{ij}, we should make $u_i + v_j = w_{ij}$ (i.e. the lowest we can make it while still satisfying the dual). Similarly, if we have nonzero values for our dual variables, we should make the corresponding $\sum_j x_{ij}$ and $\sum_i x_{ij}$ as high as we can while still satisfying the primal.

The Hungarian algorithm maintains primal dual feasibility at all steps in the computation. It also satisfies the first and third conditions above, but doesn't necessarily satisfy the second until termination.
We will have a variable π_j associated with each node $j \in T$ that tells us by how much we must change the dual variables so that j can be added to an alternating tree. A node $j \in T$ is considered scanned if $\pi_j = 0$.

- Step 0 (*Initialization*) Given a bipartite graph $G = (S, T, E)$ and weights w_{ij} for each $(i,j) \in E$. Initialize our matching $M := \emptyset$. Let $u_i = \max w_{ij}$ for each $i \in S$. Set $v_j = 0$ and $\pi_j = \infty$ for each $j \in T$. No nodes are labeled.
- Step 1 (*Labeling*)
  - (1.0) Give label "$\emptyset$" to each unsaturated node in S.
  - (1.1) If there are no unscanned labels, or if the only unscanned labels on nodes $i \in T$ for which $\pi_i > 0$, go to *Step 3*. Otherwise, find node i with unscanned label. If $i \in S$, go to (1.2); if $i \in T$, go to (1.3).
  - (1.2) Scan $i \in S$ as follows. For each arc $(i,j) \notin M$ incident to i, if $u_i + v_j - w_{ij} < \pi_j$, then label j with "i" (replacing any existing label) and set $\pi_j = u_i + v_j - w_{ij}$. Return to (1.1).
  - (1.3) Scan $i \in T$ as follows. If i is unsaturated, we have found an augmenting path; go to *Step 2*. Otherwise, identify the unique $(i,j) \in M$ incident to i and give j the label "i." Return to (1.1).
- Step 2 (*Augmentation*) An augmenting path p that terminates at i has been found. Set $M := M \oplus p$. Set $\pi_j = \infty$ for each $j \in T$. Remove all labels. Return to (1.0).

- Step 3 (*Change in dual variables*) Find

$$\begin{aligned}\delta_1 &= \min\{u_i \mid i \in S\},\\ \delta_2 &= \min\{\pi_j \mid \pi_j > 0,\ j \in T\},\\ \delta &= \min\{\delta_1, \delta_2\}.\end{aligned}$$

  We then perform the following computations:

  1. Subtract δ from u_i, for each labeled $i \in S$.
  2. Add δ to v_j for each node $j \in T$ with $\pi_j = 0$.
  3. Subtract δ from π_j for each labeled node $j \in T$ with $\pi_j > 0$.

  If $\delta < \delta_1$ go to (1.0). Otherwise, M is a maximum weight matching and the u_i and v_j are an optimal dual solution.

Theorem . The Hungarian algorithm terminates with a solution (assignment to variables) that satisfies all orthogonality conditions (1), (2), and (3).

Proof. First, the algorithm begins with the feasible solution $M = \emptyset$ and $u_i = \max\{w_{ij}\}$ and $v_j = 0$. This satisfies the orthogonal conditions (1) and (3), but not (2). The algorithm continues in such a way that conditions (1) and (3) are satisfied, and the number of conditions (2) not satisfied decreases monotonically.
The algorithm looks for augmenting paths containing arcs for which $u_i + v_j = w_{ij}$. More specifically, we look for an augmenting path from an exposed $i \in S$ that has $u_i > 0$. If an augmenting path from this i is found, we increase the size of our matching by 1. Also, conditions (1) and (3) remain satisfied, and one more condition in (2) is satisfied, since for that specific $u_i > 0$ we will have $\sum_j x_{ij} = 1$. If we cannot augment, we make our change δ in the dual variables. We choose δ to be bounded below by $\min\{u_i, \pi_j \mid i \in S,\ \pi_j > 0\}$. If $\delta = u_i$, this tells us that there is no amount π_j that we can adjust by to satisfy $u_i + v_j = w_{ij}$, which is what would allow us to add more edges to the matching. So for nodes i that are exposed, when $u_i = 0$, we are done. Conditions (2) will be satisfied, making the solution optimal in the primal and dual cases.
If, however, our δ is less that $\min\{u_i\}$, we have edges (i, j) such that $u_i + v_j > w_{ij}$. These edges in question are incident to an $i \in S$ in an alternating tree and $j \in T$ not in any alternating trees. If we subtract δ from each u_i where i is in a tree (is labeled), and add δ to each v_j where j is in a tree (is labeled), we change the net value of $u_i + v_j$ only for arcs with one end in a tree and one end out. If the in-tree node is $j \in T$, then $u_i + v_j$ is increased by δ. If, however, the in-tree node in $i \in S$, then $u_i + v_j$ is decreased by δ, potentially to w_{ij}, in which it may be added to the tree. □

In this way the algorithm continually looks to augment in such a way that we increase the number of conditions (2) that are satisfied. In order to do this, we sometimes have to make a "dual" move in which we adjust the dual variables in order to have some "slack."