Hungarian Method

May 22, 2020

Description

The Hungarian method solves the assignment problem. Given n people, n tasks, and an $n \times n$ cost (value) matrix, find the assignment that minimizes (maximizes) the cost (value). Although the problem can also be solved using the matrix directly, it is easier to reason about by using bipartite graphs.

Vocabulary

Definition: A *labelling* is attaching a number to each vertex of a graph: $\ell : V \to \mathbb{R}$.
Definition: A *feasible labelling* is one such that the sum of the labels of every edge are greater than or equal to its weight: $\ell(x) + \ell(y) \geq w(x, y)\ \forall x, y \in V$.
Definition: The *equality graph* is a subset of edges where the sum of the labels are equal to its weight: $G = (V, E_\ell)$, where $E_\ell = \{(x, y) \mid \ell(x) + \ell(y) = w(x, y)\}$.
Definition: A *matching* is a subset of edges such that every vertex is incident to at most one edge.
Definition: A *perfect matching* is a subset of edges such that every vertex is incident to exactly one edge.
Definition: A *maximum weight perfect matching* is a perfect matching with the maximum possible sum of edge weights.
Definition: A vertex is *matched* if it is incident to an edge in a matching. Otherwise it is *free*.
Definition: The *neighborhood* with respect to ℓ of a set S is the union of all the neighbors in E_ℓ of the vertices in S: $N_\ell(S) = \bigcup_{x \in S} \{y \mid (x, y) \in E_\ell\}$.
Definition: An *augmenting path* in E_ℓ is a path that starts and ends with unmatched edges, and alternates in between: $P = e_0 e_1 \dots e_n$, where n is even and $e_i \in M$ if i is odd.

Definition: The *trivial labelling* of a bipartite graph with partitions X and Y is to label each vertex in X with the weight of its largest incident edge, and label the vertices of Y with 0:

$$\ell(v) = \begin{cases} \max_y w(v,y) & \text{if } v \in X \\ 0 & \text{if } v \in Y \end{cases}$$

Theorems

Kuhn-Munkres Theorem: Let ℓ be a feasible labelling and M be a perfect matching in E_ℓ. Then M is a maximum weight perfect matching.

Proof. Let M' be an arbitrary perfect matching in G. Then

$$w(M') = \sum_{(x,y)\in M'} w(x,y) \leq \sum_{(x,y)\in M'} (\ell(x) + \ell(y))$$

Since M' is a perfect matching, every edge is incident to exactly one vertex, so

$$\sum_{(x,y)\in M'} (\ell(x) + \ell(y)) = \sum_{v\in V} (\ell(v))$$

Now, we have

$$w(M) = \sum_{(x,y)\in M} w(x,y) = \sum_{(x,y)\in M} (\ell(x) + \ell(y))$$

since $M \subseteq E_\ell$. Again, since M is a perfect matching,

$$\sum_{(x,y)\in M} (\ell(x) + \ell(y)) = \sum_{v\in V} (\ell(v))$$

Therefore, $w(M') \leq w(M)$, so M is a maximum weight perfect matching. □

This theorem tells us that if we find a perfect matching in an equality graph of G, it is guaranteed to be the optimal perfect matching.

Theorem: Let G be a bipartite graph partitioned into vertex sets X and Y and ℓ be a feasible labelling. Let $S \subseteq X$, $T = N_\ell(S) \neq Y$. Let

$$\alpha = \min_{(x\in S, y\notin T)} (\ell(x) + \ell(y) - w(x,y))$$

Let

$$\ell'(v) = \begin{cases} \ell(v) - \alpha & \text{if } v \in S \\ \ell(v) + \alpha & \text{if } v \in T \\ \ell(v) & \text{otherwise} \end{cases}$$

Then, all of the following are true:

1. ℓ' is a feasible labelling.
2. $N_{\ell'}(S) \neq T$.

Proof.

1. There are 4 types of edges (x, y) to consider:
   (a) $x \in S, y \in T$. Then,

   $$\ell'(x) + \ell'(y) = \ell(x) - \alpha + \ell(y) + \alpha = \ell(x) + \ell(y) \geq w(x, y)$$

   (b) $x \notin S, y \notin T$. Then,

   $$\ell'(x) + \ell'(y) = \ell(x) + \ell(y) \geq w(x, y)$$

   (c) $x \in S, y \notin T$. Then,

   $$\ell'(x) + \ell'(y) = \ell(x) - \alpha + \ell(y)$$

   Since α was the minimum value, we know

   $$\begin{aligned} \alpha &\leq \ell(x) + \ell(y) - w(x, y) \\ -\alpha &\geq -\ell(x) - \ell(y) + w(x, y) \\ \ell(x) - \alpha + \ell(y) &\geq w(x, y) \\ \ell'(x) + \ell'(y) &\geq w(x, y) \end{aligned}$$

   (d) $x \notin S, y \in T$. Then

   $$\ell'(x) + \ell'(y) = \ell(x) + \ell(y) + \alpha \geq \ell(x) + \ell(y) \geq w(x, y)$$

Thus, ℓ' is a feasible labelling.

2. Let $x \in S, y \notin T$ be the vertices that gave us α. Since $N_\ell(S) = T$, we know that $y \notin N_\ell(S)$. From 1(c), using equality instead of inequality, we can see that $\ell'(x) + \ell'(y) = w(x, y)$. Therefore, $(x, y) \in E_{\ell'}$, so $y \in N_{\ell'}(S)$. Since $y \notin T$, $N_{\ell'}(S) \neq T$.

□

The Algorithm

The idea of the algorithm is to start with an empty matching and the trivial labelling. In each phase, we add edges to the equality graph until we create an augmenting path that we can "flip" without causing any conflicts. Since the augmenting path starts and ends with unmatched edges, this will increase the size of our matching by one. The algorithm is as follows:

```
def hungarian(G):
  Start with trivial labelling L and empty matching M
  while M is not a perfect matching:
    Choose a free vertex u in X
    S = {u}
    T = {}

    if N_L(S) == T:
      alpha = min([L[x] + L[y] - w(x,y) for x in S, y not in T])
      L[x] -= alpha for x in S
      L[y] += alpha for y in T

    Choose y in N_L(S)\T
    if y is matched to some x in S:
      S.add(x)
      T.add(y)
    else:
      for each edge e in augmenting path u...y:
        if e in M:
          M.remove(e)
```

```
        else:
          M.add(e)

  return M
```

Correctness

1. We can always start with trivial labelling and and empty matching.

2. When we update the labelling of E_ℓ in the case where $N_\ell(S) = T$, M is guaranteed to remain in the resulting equality graph $E_{\ell'}$.

   Proof. Suppose $(x, y) \in M \subseteq E_\ell$. Again, there are 4 types of edges $(x, y) \in M$ to consider:

   (a) $x \in S, y \in T$.
   By part 1(a) of the lemma, $\ell'(x) + \ell'(y) = w(x, y)$, so $(x, y) \in E_{\ell'}$

   (b) $x \notin S, y \notin T$.
   By part 1(b) of the lemma, $\ell'(x) + \ell'(y) = w(x, y)$, so $(x, y) \in E_{\ell'}$

   (c) $x \in S, y \notin T$.
   Since $N_\ell(S) = T$, but $y \notin T$, this means (x, y) cannot be in E_ℓ and therefore $(x, y) \notin M$, so this case is not possible.

   (d) $x \notin S, y \in T$.
   If $y \in T$ and $(x, y) \in M$, then it must be the case that $x \in S$ since in our algorithm, when we add y to T, we always add the matching x as well. Therefore, this case is not possible.

   □

3. When $N_\ell(S) \neq T$ and $y \in N_\ell(S) \setminus T$ is free, $u \dots y$ is an augmenting path and flipping that path will increase the number of matchings by 1.

   We first show by strong induction that $u \dots y_k$ is an augmenting path for any y_k that we visit in a phase (the iterations over which the size of the matching does not change).

   Base case: From the algorithm, we know u is not matched. When we pick y_0, it must be a neighbor of u, since $S = \{u\}$. Therefore, (u, y_0) is not matched, so uy_0 is an augmenting path.

Inductive Hypothesis: Assume true for the i^{th} y that we visit, $i < k$, that we visit.

Inductive Step: Consider a chosen vertex y_k. That means y_k was a neighbor of some $x \in S$. But to get into S, that x must have been matched to some y_j, $j < k$, that we already put into T, so (x, y_j) is matched. Since a vertex can only be incident to a single matching edge, (x, y_k) is unmatched. By induction, we know $u \dots y_j$ is an augmenting path. Therefore, $u \dots y_j x y_k$ is also an augmenting path.

From the algorithm, we know u and y are not matched, so giving them matchings will not conflict with any existing matchings. All intermediate vertices on the augmenting path have one edge that is a matching and another that is not, so switching them will not cause a conflict either. Since, the augmenting path starts and ends with unmatched edges and flipping it does not eliminate any existing matchings, the flip increases the number of matchings by 1.

4. The algorithm is guaranteed to terminate. Since we increase the size of M at each phase, we can have a maximum of n phases. In each phase, we can only reach the case where $N_\ell(S) \neq T$ up to n times. This is because we keep moving the matched ys to T, and we have to reach a free y at some point; otherwise, if all the ys were matched, we would have already had a perfect matching. Also, we already showed by our lemma, that every time we update our matchings when $N_\ell(S) = T$, we move to the case where $N_\ell(S) \neq T$, so this can happen n times per phase as well.

5. The algorithm is correct because it is guaranteed to terminate with a perfect matching in the equality set of a feasible labelling. Therefore, by the theorem, it is a maximal perfect matching.

Time Complexity

For efficiency, we keep track of the alphas of each $y \notin T$ throughout each phase.

As explained previously, we have at most n phases. In each phase, we can hit the cases of $N_\ell(S) = T$ and $N_\ell(S) \neq T$ each n times. In the former case, we have to get the minimum of all the alphas and then update them to account for the new labelling. This takes $O(n)$ time. In the latter case, we add a vertex x to S, so we must check if (x, y) is smaller than the current alpha for each $y \notin T$. This takes $O(n)$ time as well. Altogether, we have up to n phases, and each phase contains up

to n iterations, and each iteration takes $O(n)$ time, giving us an asymptotic runtime of $O(n^3)$.