Optimization Algorithms **Winter 2019**

Maximum Weight Bipartite Matching

Choong Wey Yeh

Consider a weighted bipartite graph, where the graph G has a *weight* $w(i, j)$. Then, the weight of the matching M is the sum of the weights of the edges: $w(M) = \sum_{e \in M} w(e)$. We wish to find the matching in the graph G with the **maximum weight**. More specifically, we shall look at the Kuhn-Munkres algorithm for maximum weight bipartite matching on bipartite graphs with equal partitions.

1 Paths and Trees

Definition 1 *Let M be a matching of a graph G. A vertex v is matched if it's an endpoint of an edge in M, else it is free*

Definition 2 *A path is called alternating if the edges in the path p alternate between M and $E - M$. An alternating path is called augmenting if the endpoints of the path are free vertices.*

Note that the augmenting path has one less edge in M than in $E - M$. Hence, replacing the edges in the augmenting path that are in M with those that are in $E - M$ will maintain the matching, but introduce one more edge into the matching.

Definition 3 *An alternating tree is a tree rooted at a free vertex v in which every path is alternating.*

2 Labelings and Equalities

In the presence of weights, the maximum flow reduction does not work anymore. Instead, we need to perform a reduction to minimum cost maximum flow. However, we shall see that there is another algorithm that does not use maximum flow which is more efficient.

2.1 Feasible Labelings

Definition 4 *A vertex labeling is a function $l : V \to R$.*

The vertex labelling function is used to assign weights or labels to each vertex, which we will use later in the algorithm to achieve optimality. A *feasible labeling* is one where the sum of the labels of the two endpoints of an edge e is at least that of the weight of the edge $w(e)$. That is,

$$l(x) + l(y) \geq w(x, y), \forall x \in X, y \in Y$$

2.2 Equality Graphs

Definition 5 *The **equality graph** with respect to the labeling function l is the graph $G = (V, E_l)$ where*

$$E_l = \{(x, y) : l(x) + l(y) = w(x, y)\}$$

The equality graph consists of the edges with feasible labelings, but with a stricter condition that the sum of the two endpoints must be *equal* to that of the weight of the edge.

Theorem 6 *If l is a feasible and M a perfect matching in E_l, then M is a maximum weight bipartite matching.*

Proof: Denote any edge e in E as $e = (x, y)$.
Let the matching M' be any perfect matching in G, not necessarily in E_l. Since M' is a perfect matching, every endpoint is covered by at least one edge in M' and we have:

$$w(M') = \sum_{e \in M'} w(e) \leq \sum_{e \in M'} l(x) + l(y) = \sum_{v \in V} l(v)$$

Hence, the labelings placed on each vertex in the feasible labeling forms an *upper bound* for the cost of any perfect matching.

Now, let the matching M be a perfect matching in E_l. Then,

$$w(M) = \sum_{e \in M} w(e) = \sum_{e \in M} l(x) + l(y) = \sum_{v \in V} l(v)$$

because in the equality graph G, for each edge $e \in E_l$, the weight of the edge $w(e)$ is exactly equals to the sum of the labels of the endpoints $l(x) + l(y)$.Hence $w(M') \leq \sum_{v \in V} l(v) = w(M)$ and M is the optimal maximum weight perfect matching. ■

For the feasible labeling l, we have that the sum of the labels of the endpoints is at least that of the weight of the edge $w(e)$. However, we also cannot make the labelings too expensive because then there would be no edges in the equality graph $G = (V, E_l)$. Intuitively, we can think of the labelings being high enough to accommodate edges in E_l such that there is a perfect matching. That is, the weights of the edges $w(e)$ form a sort of upper bound on the labelings.

This theorem is also known as the **Kuhn-Munkres** theorem, and essentially transform the optimization problem of finding a maximum-weight matching into the combinatorial form of finding a perfect matching, which is a classic trick used to solve many optimization problems.

Notice that the proof for the Kuhn-Munkres theorem implies that for any matching M and any feasible labeling l, then

$$w(M) \leq \sum_{v \in V} l(v)$$

which is very similar to the capacity of the minimum cut upper bounding any flow in the Max-Flow Min-Cut Theorem.

3 Hungarian Algorithm

The Kuhn-Munkres algorithm or also known as the Hungarian algorithm, is as follows:

Algorithm 1 Kuhn-Munkres Algorithm

```
Start with a feasible labeling l and some matching M in E_l
while M is not perfect do
    if ∃ augmenting path in E_l then
        Find an augmenting path p for M in E_l
        Augment down path p to increase size of M
    else
        Improve labeling l to l' such that E_l ⊂ E_l'
    end if
end while
```

Note that at each iteration of the algorithm, we either increase the size of the current matching M if possible by finding an augmenting path, or we increase the size of E_l by relabeling. Hence, the algorithm must terminate. Furthermore, the algorithm terminates only when the while loop terminates, which indicates that the resulting matching must be perfect in E_l for some labeling l. By the Kuhn-Munkres theorem, we have that the perfecting matching M returned in maximum weight.

3.1 Initialization

Finding an initial labeling that satisfies the feasibility constraint is simple. We can use the following assignment:

$$\forall v \in X, l(v) = max_{u \in Y} w(v, u)$$
$$\forall v \in Y, l(v) = 0$$

where X, Y are partition vertex sets in the graph G. With this labeling, it is clear that the feasibility constraint is satisfied: $\forall x \in X, \forall y \in Y, w(x) \leq l(x) = l(y)$.

3.2 Updating Labelings

Now we need a way to update the labelings such that we have a new labeling that is feasible, but includes more edges into our equality graph without removing any current edges.

Let l be a feasible labeling, and define the neighbours of a vertex $u \in V$ reachable from u in the equality graph to be:

$$N_l(u) = \{v : (u, v) \in E_l\}$$

and the set $S \subset V$ to be

$$N_l(S) = \cup_{u \in S} N_l(u)$$

Lemma 7 *Let $S \subset X$ and $T = N_l(S) \neq Y$. Denote*

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

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

Then $l'(v)$ is a new feasible labeling and there is at least one edge introduced into the equality graph E_l. We can see that:

- α_l represents the minimum difference between labelings of vertices in the two partitions x, y and the weight of their edge $w(x, y)$ for edges $e \notin E_l$. That is,α_l denotes the minimum change in the labelings of the vertices such that at least one additional edge satisfies the equality constraint.
- If $(x, y) \in E_l, x \in S, y \in T$, then we have that $l(x)$ is decremented by α_l, and $l(y)$ is incremented by α_l. That is, $l'(x) + l'(y) = (l(x) - \alpha_l) + (l(y) + \alpha_l) = l(x) + l(y) \geq w(x, y)$ Hence the labeling for $l(x), l(y)$ is still feasible. We have that $(x, y) \in E_{l'}$.
- If $(x, y) \in E_l, x \notin S, y \notin T$, then the labelings for both endpoints $l(x), l(y)$ are unchanged, and therefore still remains feasible and $(x, y) \in E_{l'}$.
- For some edge $e = (x, y), x \in S, y \notin T$, then the improved labeling l' introduces this edge into the equality graph. More specifically, for the edge $e = (x, y) \notin E_l, x \in S, y \notin T$ with the minimal difference $l(x) + l(y) - w(x, y)$, we have that the labeling $l'(x) = l(x) - \alpha_l = w(x, y) - l(y)$ and hence $l'(x) + l'(y) = w(x, y) - l(y) + l(y) = w(x, y) \rightarrow$ equality constraint satisfied for (x, y). Since this difference is minimal, for all other edges $(x, y) \notin E_l$, $l'(x) + l'(y) \leq w(x, y)$. This step is important as it introduces a new edge that is not in the neighbours of $u, N_l(u)$ into the equality graph
- For $(x, y) \in E, x \notin S, y \in T$, we have that the $l'(x) = l(x), l'(y) = l(y) + \alpha_l$ and $l'(x) + l'(y) = l(x) + l(y) + \alpha_l \geq w(x, y)$. Hence the labeling remains feasible.

3.3 Full Hungarian Method

Algorithm 2 Complete Kuhn-Munkres Method

1: Generate initial labeling:
2: $\quad \forall v \in X, l(v) = max_{u \in Y} w(v, u)$
3: $\quad \forall v \in Y, l(v) = 0$
4: Generate initial matching M
5:
6: **if** M is perfect **then**
7: $\quad$ terminate
8: **else**
9: $\quad u \in X \leftarrow$ free vertex in X
10: $\quad S \leftarrow \{u\}, T = \emptyset$
11: **end if**
12:
13: **if** $N_l(S) = T$ **then** update labels:
14:

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

15:

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

16: **end if**
17:
18: **if** $N_l(S) \neq T$ **then** pick $y \in N_l(S) - T$:
19: $\quad$ **if** y is free **then** $u - y$ is an augmenting path:
20: $\quad\quad$ augment M down this path and go to 6
21: $\quad$ **else**
22: $\quad\quad$ y is matched(to z) $\rightarrow$ extend alternating tree:
23: $\quad\quad$ $S = S \cup \{z\}, T = T \cup \{y\}$
24: $\quad\quad$ go to 13
25: $\quad$ **end if**
26: **end if**

4 Proof of Correctness

4.1 Initialization

As mentioned before, we use the initial labeling function as in the algorithm. This labeling is feasible for all edges. Additionally, we can always start with the trivial naive empty matching $M = \emptyset$ to start the algorithm, which is a valid matching.

4.2 Maintenance

If $N_l(S) = T$, that means that we have exhausted all possible neighbours reachable in the equality graph E_l. In this case, by Lemma 7, we can always update the labels such that the current edges in E_l are still in $E_{l'}$, while introducing a new edge that is not previously in the equality graph. Because of the way we chose the update to be the minimal difference in the labels and the weights of edges not in the equality graph, we do not disrupt the current structure too much, which ensures that the edges in the current matching M which are in $E_l \subset E_{l'}$ are still valid.

If $N_l(S) \neq T$, then we can always augment the alternating tree with our chosen $(x, y), y \in N_l(S) - T$. Because the current matching is not perfect, there exists a y that is not matched, and hence at some point in time y chosen will be free and we can augment down this path.

4.3 Termination

Just as previously mentioned, each iteration increases either the size of the matching(by augmenting down the alternating tree) or increases the size of the neighbours reachable in the equality graph E_l by updating the labels. The algorithm terminates when the matching size cannot be further increased, that is when the matching is perfect. By Kuhn-Munkres theorem, M is a perfect matching found in E_l and hence, optimal.

5 Complexity

Let us denote each phase of the algorithm as the period of time between increments in the size of the matching M. Then, the work done between phases is mainly influenced by the calculation of the α_l term, which happens during:

- Initialization when the labels are computed. This takes $O(V)$ time for V vertices to be labeled.
- Updating the labels when all neighbours are exhausted. This takes $O(V)$ time per labeling update, and for each labeling update, we introduce at least one new edge into the equality graph. This can happen only V times if no augmenting path is found each time, since each time an edge extends from the set S, and hence augments an alternating tree. Since the maximum size matching is $\frac{V}{2}$, eventually after V updates an augmenting path of length V is present, in which can be augmented to increase the matching size. Hence, the total upper bound on this stage is $O(V^2)$.
- Augmenting the alternating tree. In this case, when we extend the current alternating tree with the matched vertex z, we need to update the labelings which takes $O(V)$ time. Since $S \to S \cup \{z\}$ can only happen a total of $\frac{V}{2}$ times by adding each of the $\frac{V}{2}$ vertices into S, this takes a total of $O(V^2)$ time

Since the maximum total matching size for a perfect matching is $\frac{V}{2}$, the number of phases is $O(V)$ since there can only be a total of $\frac{V}{2}$ increases. Therefore, the total time complexity is $O(V^3)$.