CMPUT 675: Topics in Combinatorics and Optimization **Fall 2016**

Lecture 19 (Oct. 21): Primal-Dual Algorithms: The Hungarian Method

Lecturer: Zachary Friggstad *Scribe: Zachary Friggstad*

19.1 Maximum-Weight Perfect Matching LP

Let $\mathcal{G} = (V; E)$ be a graph with $|V| = 2$ and $w : E \to \mathbb{R}$ a weight function (can be negative). In the Maximum-Weight Perfect Matching problem, we want to compute a matching $M \subseteq E$ of size $|V|/2$ (i.e. a perfect matching) of maximum possible weight or determine no such matching exists.

An LP relaxation for this problem is:

$$\begin{array}{rrcll} \text{maximize :} & \sum_{e\in E} w(e) \cdot \mathbf{x}_e & & & \\ \text{subect to :} & \mathbf{x}(\delta(v)) & = & 1 & \text{for each } v \in V \\ & \mathbf{x} & \geq & 0 & \end{array}$$

In non-bipartite graphs, this may have fractional extreme point solutions and may have value greater than the value of any perfect matching. For example, Figure 19.1 depicts a graph whose only perfect matching has value 2, yet assigning $\mathbf{x}_e = 1/2$ to all 1-weight edges (and $\mathbf{x}_e = 0$ to the 0-weight edge) yields a feasible LP solution with value 3. Further, by removing the middle edge we find that there is in fact no perfect matching yet the LP has an optimal value.

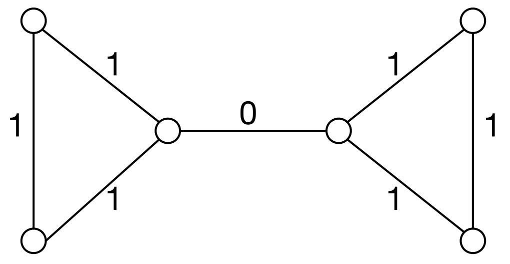

Figure 19.1: Bad gap example for the simple matching LP in nonbipartite graphs.

However, as we saw in the total unimodularity discussion, the extreme points of this LP are integral if $\mathcal{G}$ is bipartite. Rather than solving the LP using a generic LP solver, we describe a combinatorial algorithm that constructs a maximum-weight perfect matching. The steps of the algorithm will be guided by a solution to the dual of the LP, which is

$$\begin{array}{rrcll} \text{minimize :} & \sum_{v\in V} \mathbf{y}_v & & & \\ \text{subject to :} & \mathbf{y}_u + \mathbf{y}_v & \geq & w(e) & \text{for each } e = uv \in E \end{array}$$

Note the dual variables may be negative, as the primal constraints are equality constraints.

In this case, complementary slackness says that a feasible primal solution $\overline{\mathbf{x}}$ and a feasible dual solution $\overline{\mathbf{y}}$ are both optimal solutions for their respective LPs if and only if

$$\mathbf{x}_e \cdot (\mathbf{y}_u + \mathbf{y}_v - w(e)) = 0 \text{ for each } e = uv \in E.$$

So, if we have that $\mathbf{x}$ is an optimal integral solution, then any chosen edge has its corresponding dual constraint tight. As tight edges are important to this discussion, we use the following notation.

Definition 1 *Let $\overline{\mathbf{y}}$ be a feasible solution to the dual LP. We let $E_{\overline{\mathbf{y}}} = \{e = uv \in E : \overline{\mathbf{y}}_u + \overline{\mathbf{y}}_v = w(e)\}$ be the set of edges whose dual constraint is tight under $\overline{\mathbf{y}}$.*

Summarizing:

Lemma 1 *Let $M \subseteq E$ be a perfect matching and $\overline{\mathbf{y}}$ a feasible dual solution. If $M \subseteq E_{\overline{\mathbf{y}}}$ then M is a maximum-weight perfect matching.*

19.2 The Hungarian Method

Here we let L, R denote the two sides of the bipartite graph $\mathcal{G} = (V; E)$.

The Hungarian Method is a primal-dual algorithm that simultaneously constructs a perfect matching M and a feasible dual solution $\overline{\mathbf{y}}$ witnessing optimality of M (as per Lemma 1). We call it a **primal-dual** algorithm because the construction of the primal solution (the matching) is motivated by the desire to satisfy complementary slackness with some dual solution.

In the algorithm, we will start with a trivial feasible dual solution and "buy" edges whose dual constraint is tight or update the dual solution to produce more tight edges that can be bought. Ultimately, we will end up with a dual solution such that the set of tight edges contains a perfect matching.

We slightly adapt the concept of an M-alternating forest to the bipartite setting.

Definition 2 *Let $\overline{\mathbf{y}}$ be a feasible dual solution and $M \subseteq E_{\overline{\mathbf{y}}}$ a matching. A* **left-exposed** *M-alternating forest in $(V; E_{\overline{\mathbf{y}}})$ is a forest $\mathcal{T} = (U; F)$ that satisfies the following properties.*

- *$F \subseteq E_{\overline{\mathbf{y}}}$*
- *In each component, there is precisely one M-exposed node of L. Call this the* **root** *of the component.*
- *Every node with odd-height in U has precisely one child in $\mathcal{T}$: the vertex it is matched to.*

Note from the definition that even-height nodes of $\mathcal{T}$ lie in L and odd-height nodes of $\mathcal{T}$ lie in R. Furthermore, no M-exposed node in R can lie in U. Call such a forest **maximal** if for each $u \in U \cap L$ the only edges of the form $uv \in E_y$ either have $v \in U$ or v being M-exposed.

The algorithm starts with $M = \emptyset$ and $\overline{\mathbf{y}}$ being a trivially feasible solution. It attempts to increase the size of M by finding M-alternating paths in $E_{\overline{\mathbf{y}}}$. If this is not possible, it adjusts $\mathbf{y}$ to produce more edges in $E_{\overline{\mathbf{y}}}$.

Lemma 2 *Throughout, $M \subseteq E_{\mathbf{y}}$.*

Proof. This is straightforward to prove as a loop invariant. If M is updated, it is only along edges in $E_{\overline{\mathbf{y}}}$ (as the forest $(U; F)$ has $F \subseteq E_{\overline{\mathbf{y}}}$). If $\overline{\mathbf{y}}$ is updated, any $e \in F$ continues to be in $E_{\overline{\mathbf{y}}}$ because one endpoint is increased by Δ and the other is decreased by Δ. ■

Lemma 3 *If the algorithm returns* **no perfect matching** *then $|N_E(U \cap L)| < |U \cap L|$ where U is the current forest (i.e. there really is no perfect matching, by Hall's theorem).*

Algorithm 1 Hungarian Method

```
1: $\overline{\mathbf{y}}_u \leftarrow 0$ for each $u \in L$.
2: $\overline{\mathbf{y}}_v \leftarrow \max_{e \in E} c_e$ for each $v \in R$.    {Now $\mathbf{y}$ is feasible}
3: $M \leftarrow \emptyset$    {We maintain the invariant $M \subseteq E_{\overline{\mathbf{y}}}$}
4: while $M$ is not a perfect matching do
5:    Let $(U;F) \leftarrow$ be a maximal left-exposed $M$-alternating forest with $F \subseteq E_{\overline{\mathbf{y}}}$
6:    if some $uv \in E_{\overline{\mathbf{y}}}$ with $u \in U \cap L$ has $v \notin U$ then
7:       Let $P$ be the edges on the path from the root of $u$'s component to $u$, followed by $uv$.
8:       $M \leftarrow (M - P) \cup (P - M)$    {$P$ is an $M$-alternating path}
9:    else if $N_E(U \cap L) \subseteq U$ then
10:      return no perfect matching
11:   else
12:      $\Delta \leftarrow \min_{\substack{e=uv \in E \\ u \in U \cap L \\ v \in R-U}} \mathbf{y}_u + \mathbf{y}_v - w(e)$
13:      For each $u \in U, \mathbf{y}_u \leftarrow \begin{cases} \overline{y}_u - \Delta & \text{if } u \in L \\ \overline{y}_u + \Delta & \text{if } u \in R \end{cases}$
14:   end if
15: end while
```

Proof. The edges of M pair up all nodes in U, except for the exposed nodes which lie in L. So $|N_E(U \cap L)| = |U \cap L| + |L| - |M| \leq |U \cap L| + 1$ as M is not a perfect matching. ∎

Lemma 4 *Throughout, $\overline{\mathbf{y}}$ is a feasible dual solution.*

Proof. The initial $\overline{\mathbf{y}}$ is set to be trivially feasible. In the step where $\overline{\mathbf{y}}$ is updated, the only edges uv whose slack is reduced are those with $u \in U \cap L, v \in R - U$. Feasibility will be maintained by the choice of Δ. ∎

To see the algorithm eventually terminates, we note any iteration where M is not augmented and a declaration of no perfect matching is not made, the forest will grow by at least one edge. Namely, the one that achieved the minimum value when Δ was defined.

So the algorithm terminates either with a correct declaration there is no perfect matching or with a perfect matching $M \subseteq E_{\overline{\mathbf{y}}}$ where $\mathbf{y}$ is a feasible solution. Thus,

Theorem 1 *Algorithm 1 finds a minimum-cost perfect matching, as long as there is at least one.*

Finally, we can implement this algorithm in $O(n^3)$ time. We show there is $O(n^2)$ work done between iterations of M growing. Maintain a vector `slack[v]` for each $v \in R - U$ recording the minimum slack of an edge uv with $u \in U \cap L$. We can find the value of Δ in $O(n)$ time each step (and the actual edge if we also keep track of the edge itself having minimum slack) and also update the slacks in $O(n)$ time when $\overline{\mathbf{y}}$ is adjusted.

The forest $(U;F)$ can be grown in linear time and updated in constant time each time an edge goes tight. Furthermore, we can check for an alternating path each time an edge goes tight before adding it to the tree.

Finally, when the tree grows because an edge goes tight, the `slack[v]` values can be updated in $O(n)$ time by examining the neighbours of the new nodes added to the tree.

In total, the tree grows $O(n)$ times between increases in $|M|$ and each increase takes $O(n)$ time to implement (to compute Δ and update the slacks).