Math 233B: Polyhedral techniques in combinatorial optimization
Instructor: Jan Vondrák
Lecture date: 1/25/2017 **Original scribe: Ian Post**

1 Weighted non-bipartite matching

Today we extend Edmond's matching algorithm to weighted graphs. The minimum weight perfect matching problem can be written as the following linear program:

$$\begin{array}{lll} \min & \sum_{e\in E} w_e x_e & \\ \text{s.t.} & \forall v \in V & x(\delta(v)) = 1 \\ & \forall U \subset V, |U| = \text{odd} & x(\delta(U)) \geq 1 \\ & \forall e \in E & x_e \geq 0 \end{array}$$

But this program has exponentially-many constraints.

One approach would be to use the ellipsoid algorithm: if we can implement a "separation oracle" in polynomial time, then we can solve the LP. The separation problem is, given $\mathbf{x}$, check all the constraints. Padberg and Rao showed how to solve the problem of minimizing $x(\delta(U))$ over all odd U. If the answer is at least 1, then the constraints are satisfied, and if for some U $x(\delta(U)) < 1$, then we have a separating hyperplane. However, we will present a different, more direct algorithm here.

1.1 Primal-dual algorithms

We use a primal-dual algorithm due to Edmonds. A primal-dual algorithm is a combinatorial algorithm that solves an LP using the primal/dual structure and was a major breakthrough in algorithm design at the time. The general approach is to keep track of a primal solution $\mathbf{x}$ and its set of tight constraints and try to find a dual solution $\mathbf{y}$ that certifies $\mathbf{x}$ is optimal. If this is impossible, the dual solution will give us a way to improve $\mathbf{x}$.

More specifically, for the following primal and dual LPs

$$\begin{array}{lll} \min & \mathbf{c}^T\mathbf{x} & \\ \text{s.t.} & \mathbf{A}\mathbf{x} & \geq \mathbf{b} \\ & \mathbf{x} & \geq 0 \end{array} \qquad \begin{array}{lll} \max & \mathbf{b}^T\mathbf{y} & \\ \text{s.t.} & \mathbf{A}^T\mathbf{y} & \leq \mathbf{c} \\ & \mathbf{y} & \geq 0 \end{array}$$

we will maintain a solution pair $(\mathbf{x}, \mathbf{y})$, where $\mathbf{y}$ is dual feasible and $(\mathbf{A}^T\mathbf{y})_i = c_i$ for each $x_i > 0$. If $\mathbf{x}$ is feasible and $(\mathbf{A}\mathbf{x})_j = b_j$ for $y_j > 0$, then $(\mathbf{x}, \mathbf{y})$ are optimal.

If there is no such $\mathbf{x}$, then by duality there exists $\mathbf{z}$ such that

$$\begin{array}{lll} \mathbf{A}_S^T\mathbf{z} & \leq 0 & \text{where } S = \{i : A_i^T\mathbf{y} = c_i\} \text{ are the tight constraints for } \mathbf{y} \\ \mathbf{z}_R & \geq 0 & \text{where } R = \{j : y_j = 0\} \\ \mathbf{b}^T\mathbf{z} & > 0 & \end{array}$$

in which case we can move along $\mathbf{z}$ and improve the dual.

1.2 Primal-dual algorithm for weighted perfect matching

The dual of the weighted perfect matching LP above is

$$\begin{array}{lll} \max & \sum_{|U|=\text{ odd}} y_U & \\ \text{s.t.} & \forall e \in E & \sum_{U:e\in\delta(U)} y_U \leq w_e \\ & \forall U, |U| \geq 3 \text{ odd}, & y_U \geq 0 \end{array}$$

Note that here, y_U includes the case of $|U| = 1$ and these dual variables represent the vertex constraints.

Our algorithm will maintain a feasible dual solution $\mathbf{y}$ and an infeasible primal solution $\mathbf{x}$. Additionally,

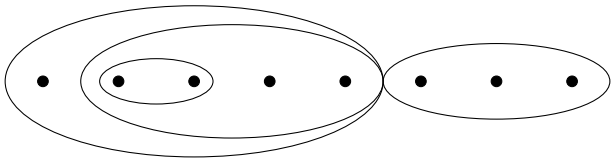
1. Our primal solution satisfies $x_e \in \{0, 1\}$, $\mathbf{x} = \chi_M$ where M is a matching and the edges in M always have tight dual constraints, i.e.

$$x_e = 1 \Rightarrow \sum_{U:e\in\delta(U)} y_U = w_e$$

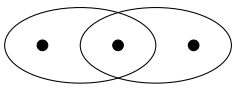
since we are trying to satisfy complementary slackness.

2. The dual solution uses only sets from some laminar family Ω.

Definition 1 *A family of sets Ω is a* laminar family *if for all $A, B \in \Omega$, $A \cap B = \emptyset$, $A \subseteq B$, or $B \subseteq A$.*



Laminar family



Non-laminar family

All singletons are part of Ω, and each $U \in \Omega$ with $|U| \geq 3$ corresponds to some contracted blossom, which the algorithm will keep track of, on the maximal sets contained in U. Note that Ω contains $O(n)$ sets, since the sets form a tree with internal degrees at least 3 and n leaves corresponding to vertices.

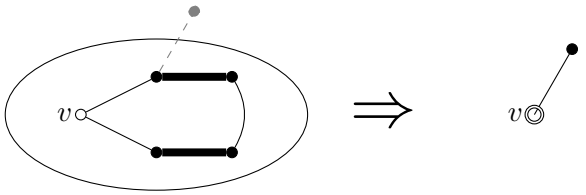
1.3 Algorithm

We assume G is simple, has a perfect matching and that $\mathbf{w} \geq 0$. The algorithm is as follows:

0. Initialization: $M = \emptyset$, $\Omega = \{\{v\} \mid v \in V\}$, $y_v = 0$ for each $v \in V$.

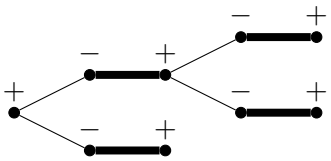
1. Let X be the exposed vertices in M and $E_y = \left\{e \mid \sum_{U:e\in\delta(U)} y_U = w_e\right\}$ be the tight edges. Build alternating trees from X in E_y. If there is an M-augmenting path, extend M and repeat. If M is a perfect matching then stop.

2. If there is an edge forming a blossom—an Even-Even edge—then shrink the blossom and add its vertex set U to Ω with $y_U = 0$. Note that U becomes an Even vertex.



Continue growing the tree or shrinking blossoms as long as possible. If there is an augmenting path, extend the matching M and return to 1 (but keep the contracted blossoms).

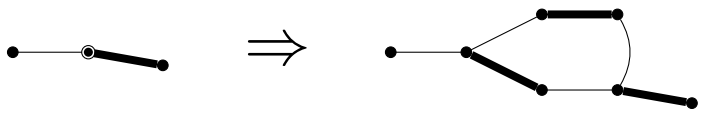
3. If no tree can be extended and there is no Even-Even edge, modify the dual solution:



Increase y_U for all Even nodes U and decrease y_U for all Odd nodes U at the same rate until you get stuck. Note that $\sum_{|U| \text{ odd}} y_U$ increases and edges inside the tree remain tight. If the process never gets stuck, then the primal is not feasible.

4. Why did we stop?

   (a) If $y_U = 0$ for some $|U| \geq 3$ (singletons can go below 0), de-shrink the blossom, add $\lfloor \frac{1}{2}|U| \rfloor$ matching edges back to M, remove U from Ω, and go back to 2. Note U must have been an Odd vertex. In the M-alternating tree, we keep only the path through U that consists of M-alternating edges; we remove the other part of the cycle.



   (b) If a new constraint $\sum_{U:e \in \delta(U)} y_U \leq w_e$ becomes tight, add e to E_y, and go back to 2.

1.4 Analysis

Why does the algorithm terminate in polynomial time?

Lemma 2 *A set U added to Ω cannot be removed until the next augmentation of M.*

Proof: When a set U is shrunk, it becomes an Even vertex, so y_U can only be increased. Shrinking or deshrinking other sets does not change its parity. If U is swallowed by a larger set U', y_U will not change anymore, and we can argue about U' instead. The only way U can become Odd is by augmenting M, and U cannot leave Ω until it becomes Odd. □

Corollary 3 *The algorithm terminates after $O(n^2)$ operations involving shrinking, deshrinking, growing the tree and augmenting M.*

Proof: By Lemma 3, the number of shrinkings between two augmentations of M can be at most the size of Ω at the end of this stage, and similarly the number of deshrinkings between two augmentations of M can be at most $|\Omega|$ at the beginning of this stage. Since Ω is laminar, both quantities are $O(n)$ and the number of shrinkings/deshrinkings between two augmentations is $O(n)$. The number of extensions of E_y is also $O(n)$: either we add new vertex to some tree, or we form a new blossom, which will be shrunk. Therefore the number of iterations between two augmentations of M is $O(n)$, and M will be augmented exactly $n/2$ times. □

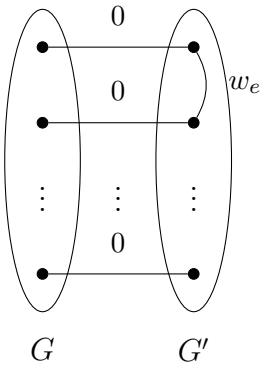
Lemma 4 *When the algorithm terminates, then M is optimal.*

Proof: As we proved, the algorithm cannot run forever. When it terminates, it is because no more operations can be performed and that can be only because M is a perfect matching. All edges in M have tight dual constraints, since the algorithm only uses tight edges ($M \subseteq E_y$). Now suppose $y_U > 0$ and $|U| \geq 3$. If U had no edges coming out, there would be an exposed vertex, and the algorithm would not have terminated. Variable y_U can only be nonzero if U is a shrunk blossom, and we only shrink sets with at most one edge coming out. In order for a second edge in $\delta(U)$ to be added, U must be expanded, which only occurs when y_U drops to 0. Therefore for every nonzero y_U, M has exactly one edge coming out of U. Since M and $\mathbf{y}$ satisfy complementary slackness they must be optimal solutions. □

This proves that minimum-weight perfect matching can be found in polynomial time. Similarly, maximum-weight perfect matching can be found in polynomial time, by flipping the weights. Finally, we can also solve the maximum-weight matching problem.

Corollary 5 *Maximum-weight matching can be found in polynomial time.*

Proof: We reduce it to max-weight perfect matching. Create two copies of the graph G, with corresponding nodes in each graph connected by edges of weight 0. The max-weight perfect matching in the resulting graph is twice the max-weight matching in G.



□