Maximum Weight Matchings in General Graphs – Part 1

Jon Turner
Computer Science & Engineering
Washington University

www.arl.wustl.edu/~jst
Maximum Weight Augmentation

- Given graph $G=(V,E)$ and matching $M$, define weight of path $p$ to be total weight of its free edges minus total weight of its matched edges.

- **Theorem 9.2.** Let $M$ be a matching of maximum weight among matchings of size $|M|$, let $p$ be an augmenting path for $M$ of maximum weight, and let $M'$ be the matching formed by augmenting $M$ using $p$. Then $M'$ is of maximum weight among matchings of size $|M|+1$.

  **Proof.** Let $M''$ be a matching of maximum weight among matchings of size $|M|+1$. Let $N$ be the set of edges in $M$ or $M''$ but not both.
Define the weight of a path or cycle in $N$ with respect to $M$. Any cycle of even length path in $N$ must have weight $\leq 0$, since otherwise we could increase the weight of $M$ without changing its size, by exchanging the edges on the cycle or path.

Since $N$ contains exactly one more edge in $M'$ than in $M$, we can pair all but one of the odd-length paths so that each pair has an equal number of edges in $M$ and in $M'$. Each such pair of paths must have total weight $\leq 0$ by the same reasoning as before.

Augmenting $M$ using the remaining path gives a matching of size $|M| + 1$ with same weight as $M'$. This must be a maximum weight augmenting path for $M$ since if there were an augmenting path with larger weight, we could construct a matching of size $|M| + 1$ with larger weight than $M'$. ■
Theorem 9.2 provides a basis for a weighted matching algorithm

» Finding max weight augmenting paths directly is difficult, especially for general graphs
» Can be done using LP duality
  • Dual variables can be viewed as vertex/blossom labels
  • Label values of edge endpoints are related to edge weights
Matching and Linear Programming

- Matchings defined by selection variables $X=\{x_e\}$
  - $x_e=1$ if $e$ is an edge in the matching
- Objective is to maximize weight($X$)=$\sum_e x_e w(e)$
- Constraints:
  - for each vertex $u$ with incident edges $E(u)$, $\sum_{e\in E(u)} x_e \leq 1$
  - for each edge $e$, $x_e=0$ or $x_e=1$
- The constraints on the $x_e$s make this an integer linear programming problem
  - Edmonds showed that for bipartite graphs, we can replace these constraints with $x_e \leq 1$
    - this ordinary LP has same optimal solutions as original ILP
    - we’ll use duality to obtain a more efficient algorithm
Dual Version of Matching LP

- First, re-state primal version in matrix form
  - define the \( n \times m \) edge incidence matrix \( G = [g_{u,e}] \) where \( g_{u,e} = 1 \) if \( u \) is an endpoint of \( e \), else \( g_{u,e} = 0 \)
  - let \( W = [w_e] \) be column vector of edge weights and let \( X = [x_e] \) be column vector of selection variables
  - primal problem becomes
    - maximize weight(\( X \)) = \( W^T X \) subject to \( GX \leq [1] \)

- Dual version uses variables \( Z = [z_u] \)
  - minimize cost(\( Z \)) = \([1]^T Z \) subject to \( G^T Z \geq W \)
  - equivalently, minimize \( \sum_u z_u \) subject to \( z_u \geq 0 \) and for all edges \( e \), \( z_e \geq w_e \) where \( z_e = z_u + z_v \) for \( e = \{u,v\} \)
  - complementary slackness implies that if \( X^* \) and \( Z^* \) are optimal, \( z_e = w_e \) for matching edges \( e \) and \( z_u = 0 \) if \( u \) is free
Max Wt. Matchings & Vertex Labeling

**Theorem.** Let $G=(V,E)$ be a bipartite graph with edge weights $w(e)$, let $M$ be a matching in $G$ and let each vertex $u$ have a non-negative label $z_u$. If

1. $z_e \geq w(e)$ for $e \in E$ ($z_e = z_u + z_v$)
2. $z_e = w(e)$ for $e \in M$
3. $z_u = 0$ if $u$ is free

then $M$ is a maximum weight matching.

**Proof.** Let $M$ and $z$ satisfy the conditions in the theorem and let $N$ be any other matching.

$$
\Sigma_{e \in N} w(e) = \Sigma_{e \in N} z_e \leq \Sigma_z z_u = \Sigma_{e \in M} z_e = \Sigma_{e \in M} w(e)
$$

**Edges with $w(e) = z_e$ are called equality edges**

- augmenting path using equality edges has max weight
Bipartite Matching Using Vertex Labels

- Initialization
  - \( M = \emptyset \) and \( z_u = (\text{max edge weight})/2 \) for all \( u \)
    - this satisfies conditions (1) and (2) in theorem
- At each step, search for augmenting paths using only equality edges (by building trees, as before)
  - halt if condition (3) becomes true
  - if search fails to find an augmenting path, modify labeling
    - this makes condition (3) true or creates more equality edges
    - in latter case, continue search for augmenting path using newly created equality edges
  - after finding a path, augment and reset even/odd status, but retain \( z \) values
    - note, augmentation maintains truth of (1), (2)
Adjusting Labels

Whenever the search runs out of eligible edges
  » if all free vertices have zero labels, terminate
  » let $\delta_1 = \min \{ z_u | u \text{ is even} \}$
    $\delta_2 = \min \{ z_e - w(e) | e = \{ u, v \}, u \text{ even}, v \text{ unreached} \}
    \delta_3 = \min \{ (z_e - w(e))/2 | e = \{ u, v \}, \text{ both even} \}
    (\delta_2, \delta_3 \text{ are undefined if no suitable edge})
    \delta = \min \{ \delta_1, \delta_2, \delta_3 \} - \text{ignore } \delta_2, \delta_3, \text{ when undefined}
  » subtract $\delta$ from labels for even vertices, add $\delta$ to labels for odd vertices
    • note: this maintains truth of (1), (2)
    » if $\delta = \delta_1$, this makes condition (3) true and algorithm halts
      • since labels start with same value and free vertices experience same sequence of changes
    » if $\delta = \delta_2$ or $\delta_3$, search can resume using new equality edges.
Implementation Details

- Use heaps to compute $\delta_l$ values efficiently
  - $h_{1e}$ and $h_{1o}$ store the even and odd vertices respectively, with $z_u$ as the key for vertex $u$
  - $h_2$, $h_3$ store edges, with keys $z_e \cdot w(e)$
    - $h_2$ has edges with one even endpoint and one unreached
    - $h_3$ has edges with both endpoints even
    - When a vertex $u$ becomes even, add its edges to $h_2$ or $h_3$

- To enable fast updating of labels use heap with fast `addtokeys(x)` operation
  - Adds $x$ to keys of all items in a heap
  - $d$-heap can be extended to do this in constant time

- Eligible equality edges appear at top of $h_2$ and $h_3$
  - Can be selected directly from the heaps
Running Time Analysis

- Number of augmentations is at most $n/2$
  - at end of search, update vertex labels using $h_{1e}$, $h_{1o}$
  - also, clear heaps in preparation for next search
- Each step that extends a tree adds edges to heaps and removes edges from heaps
  - during one augmenting search, each edge added to a heap $\leq 2$ times, removed $\leq 2$ times
  - so, $O(mn \log n)$ time for these heap operations
- All but the last label adjustment adds at least one equality edge and does not eliminate any
  - so total # of label adjustments is $O(m)$ and since these require only findmin and addtokeys, we get $O(m)$ time
D-Heap with Addtokeys Operation

- Operation \( \text{addtokeys}(x) \) adds \( x \) to keys of all items in a heap
  - add internal variable \( \Delta \) to heap implementation
  - every \( \text{addtokeys}(x) \) operation increases \( \Delta \) by \( x \)
  - let \( \Delta(t) \) be value of \( \Delta \) at time \( t \), then
    - from time \( t_1 \) to time \( t_2 \), \( \text{key}(j) \) increases by \( \Delta(t_2) - \Delta(t_1) \)

- When inserting item \( j \) with key \( k \) into heap, use \( k - \Delta \) as the stored value, in place of \( k \)
  - preserves relative values of all items in heap

- To obtain the “true key” for item \( j \), add the current value of \( \Delta \) to the stored key value