Washington University in St. Louis

Engineering

Maximum Weight Matchings in General Graphs – part 2

Jon Turner
Computer Science & Engineering
Washington University

www.arl.wustl.edu/~jst

Washington University in St. Louis — Engineering

Weighted Matchings in General Graphs

- Define LP with same optimal solutions as ILP
  - » maximize $weight(X)=\Sigma_e\, x_e w(e)$ subject to
    $\Sigma_{e\in E(u)} x_e \leq 1$ for all u with edges $E(u)$
    $\Sigma_{e\subseteq B}\, x_e \leq (|B|-1)/2$ for every *non-trivial* odd subset $B\subseteq V$
- In matrix form
  - » define $G=[g_{B,e}]$ with row for *every* odd subset $B\subseteq V$,
    - $g_{B,e}=1$ if $e\subseteq B$ or $B\in e$, else $g_{B,e}=0$
  - » let $W=[w(e)]$ be column vector of edge weights and
    let $X=[x_e]$ be column vector containing the LP variables
    let $K=[k_B]$ be column vector with entry per odd subset B
    - $k_B=\max\{1,(|B|-1)/2\}$
  - » primal problem becomes
    - maximize $weight(X)=W^TX$ subject to $GX\leq K$

2

Dual Version

- Dual version uses variables $Z=[z_B]$
  - » minimize $cost(Z)=K^TZ$ subject to and $G^TZ\geq W$
  - » equivalently, minimize $\Sigma_B\, k_B z_B$ subject to $z_e \geq w(e)$ for all edges e
    - $z_e=\Sigma_B\, z_B$ – sum is over odd subsets B where $e\subseteq B$ or $B\in e$
- Complementary slackness implies that if X and Z are optimal solutions
  - » $(G^TZ-W)^TX=[0]$ and $(K-GX)^TZ=[0]$
  - » the first condition says that for each edge $e\in M$, $z_e=w(e)$
  - » the second says
    - for every free vertex u, $z_u=0$ and
    - for every non-trivial odd subset B with $z_B\neq 0$, the number of matching edges in B is k_B

3

- *Theorem*. Let $G=(V,E)$ be a graph with edge weights $w(u,v)$, let M be a matching in G, let each odd subset B have a non-negative label z_B. For an edge e, let $z_e=\Sigma_B z_B$ where sum is over odd subsets B where $e \subseteq B$ or $B \in e$. If

  (1) $z_e \geq w(e)$ for all $e \in E$

  (2) $z_e = w(e)$ for all $e \in M$

  (3) $z_B = 0$ if B is a free vertex or the number of matching edges in B is $<(|B|-1)/2$

  then M is a maximum weight matching.

  Proof. Assume conditions (1) to (3) hold with respect to some matching M, let N be any other matching.

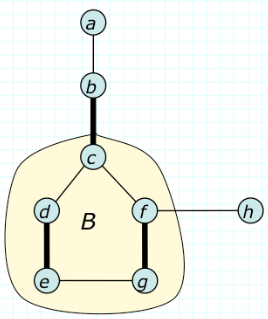
$$\sum_{e\in N} w(e) \le \sum_{e\in N} z_e$$
$$= \sum_{\{u,v\}\in N} (z_u + z_v) + \sum_{e\in N} \sum_{B:e\subseteq B} z_B$$
$$= \sum_{\{u,v\}\in N} (z_u + z_v) + \sum_{B:|B|>1} z_B \begin{pmatrix} \text{\# of edges in } N \text{ with} \\ \text{both endpoints in } B \end{pmatrix}$$
$$\le \sum_{u\in V} z_u + \sum_{B:|B|>1} z_B (|B|-1)/2$$
$$= \sum_{\{u,v\}\in M} (z_u + z_v) + \sum_{b:|B|>1} z_B \begin{pmatrix} \text{\# of edges in } M \text{ with} \\ \text{both endpoints in } B \end{pmatrix}$$
$$= \sum_{e\in M} z_e = \sum_{e\in M} w(e)$$

Hence, *M* is a maximum weight matching. ■

- *Equality edges* have $z_e = w(e)$
  - » can find max weight augmenting paths using equality edges alone

Equality Edges Yield Max Weight

- If M and z satisfy conditions (1) and (2), all free vertices have same label and only blossoms B have $z_B > 0$, augmenting paths using equality edges are max weight paths



Weight of augmenting path:

$$w(a,b)+w(c,d)+w(e,g)+w(f,h) - (w(b,c)+w(d,e)+w(f,g))$$
$$\le (z_a+z_b)+(z_c+z_d+z_B)+(z_e+z_g+z_B)+(z_f+z_h) - ((z_b+z_c)+(z_d+z_e+z_B)+(z_f+z_g+z_B))$$
$$= z_a+z_h$$

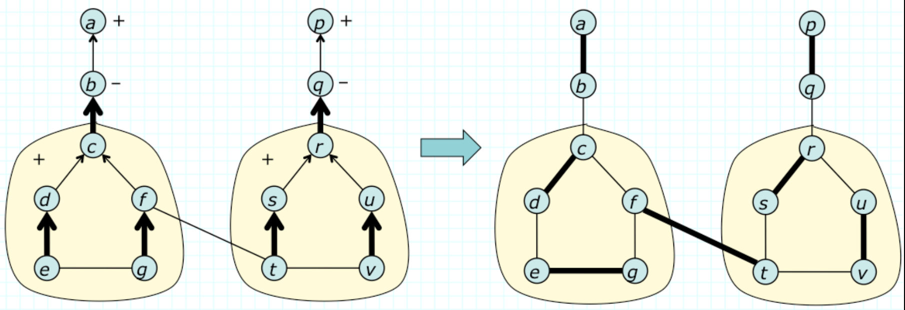
If all equality edges, then path weight equals z_a+z_h

If all free vertices have same label, any such path is max weight augmenting path

General Matching Using Labeling

- Algorithm maintains variables z_B only for vertices and blossoms B; others are implicitly 0
  - » note, this means that condition (3) in theorem holds automatically if B is not a free vertex
- Initialization
  - » $M = \{\}$, $z_u=(1/2) \max_e w(e)$
    - note: this satisfies conditions (1) and (2)
- Search for augmenting paths using equality edges
  - » if (3) becomes true, algorithm halts
  - » whenever search "stalls", modify the labels
  - » when augmenting path found, augment matching and make unexpanded blossoms unreached
    - expand only those blossoms with $z_B=0$

Augmenting Without Expanding



- Blossoms B with $z_B>0$ are retained following augmentation, along with their labels
  - » necessary to maintain condition (3)
- Means blossoms may be unreached, odd or even

- View each vertex as belonging to some (possibly trivial) blossom in the current shrunken graph
  - maintain variable z_B for all blossoms B, including those contained in other blossoms
    - z_B=0 for each new blossom; blossom expanded only if z_B=0
    - z_B values are changed only for outer-most blossoms
  - for each vertex u, let B_u denote the outermost blossom containing u in the current graph
    - state of u (odd, even, unmatched) is inherited from B_u
    - let *mate*(B_u) be outer blossom at other end of matching edge incident to B_u
  - for each blossom B, maintain an edge *entry*(B) which is the edge to the parent blossom of B in tree containing B
- Note: for any matched edge not in a blossom
  - either both endpoints are unreached, or one is even, while the other is odd

Washington University in St.Louis — Engineering

Adjusting Labels

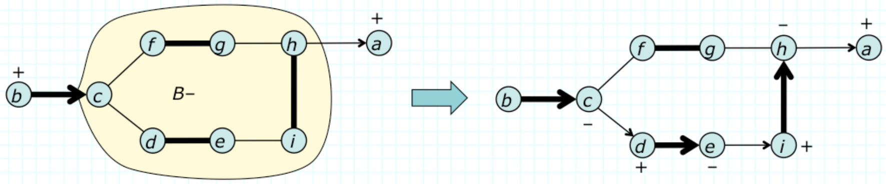
- Whenever the search runs out of eligible edges, we select a value δ and adjust labels
  - » for vertices *u*
    - subtract δ from z_u if *u* even, add δ if *u* odd
  - » for outer-most blossoms *B*
    - add 2δ to z_B if *B* even, subtract 2δ if *B* odd
- Observations
  - » for unreached, blossom or tree edges *e*, z_e doesn't change
    - for *e* contained in a blossom, change to labels for edge endpoints are balanced by change for blossom
    - for *e* outside any blossom, z_e is sum of endpoint labels and either the changes balance, or neither changes
  - » for remaining edges, take care to avoid violations of (1)

10

Choosing δ

- Select δ as follows
  - » let δ_1=min $\{z_u | u \text{ is even}\}$
    δ_2=min$\{z_e - w(e) \mid e=\{u,v\}, u \text{ even}, v \text{ unreached}\}$
    δ_3=min$\{(z_e - w(e))/2 \mid e=\{u,v\}, u, v \text{ even and not in same blossom}\}$
    δ_4=min$\{z_B/2 \mid B \text{ is a top-level odd blossom}\}$
    δ =min$\{\delta_1, \delta_2, \delta_3, \delta_4\}$
- Observations
  - » this choice ensures that labels remain non-negative
  - » label change causes one or more of the following to occur
    - algorithm terminates immediately (if $\delta=\delta_1$)
    - one or more equality edges are created (if $\delta=\delta_2$ or δ_3)
    - for at least one odd blossom *B*, z_B becomes zero (if $\delta=\delta_4$); this allows *B* to be expanded

Expanding Odd Blossoms



- When label adjustment makes z_B=0 for an odd blossom *B*, it is expanded
  - » for sub-blossoms on path from *entry*(*B*) to sub-blossom at base of *B*
    - assign new odd/even status and update *entry* edge
  - » other sub-blossoms become unreached, *entry* undefined
- Implies that a vertex can alternate between odd, unreached many times during one search
  - » complicates required data structures

Putting It Together

- Initialization
  - » $M = \{\}$, $z_u=(1/2) \max_e w(e)$
- Repeat the following step until $z_u=0$ for all free u
  - » if there is an equality edge $\{u,v\}$ with u even, v unreached
    - expand tree containing u to include B_v and *mate*(B_v), setting odd/even status and *entry* edge
  - » if there is an equality edge with $\{u,v\}$ with u, v even
    - if u, v are in same tree, form new even blossom, set *entry*
    - if u, v are in different trees, augment matching and make blossoms on augmenting path unreached, *entry* undefined
  - » if neither of the previous cases apply, adjust labels
    - if this makes $z_B=0$ for some odd blossom B, expand B and update status of sub-blossoms

Running Time Analysis

- During one augmenting path search
  - » $\leq n/2$ steps that extend the collection of trees
    - add edges at new even vertices to set of eligible edges
  - » at most $\leq n/2$ steps form new blossoms
    - since new blossoms are always even and are not expanded
  - » no edge can become an equality edge more than once
    - so, # of label adjustments that add equality edges is $\leq m$
  - » steps that expand odd blossoms
    - $\leq n/2$ since blossoms that become odd were originally formed before current search
- So, $O(m)$ steps per search, $O(mn)$ altogether
  - » with no special data structures takes $O(mn^2)$ time
  - » with appropriate data structures can cut to $O(mn \log n)$

About Data Structures

- Use heaps with *addtokeys* as in bipartite case
- Blossom structure forest
  - » contains a node for every original vertex, blossom and sub-blossom in the current graph
  - » parent of *x* is inner-most blossom that contains *x*
  - » trees implemented using doubly-linked circular lists of siblings, plus child pointer
- Split-join sets data structure to find B_u, given *u*
  - » ordered base set with *join*, *split* and *find* operations
  - » can be implemented using binary search trees
- Group heap
  - » divides heap into *groups* that can be *active* or *inactive*
    - *addtokeys* affects *active* groups