Graph Theory: Matchings and Factors

Pallab Dasgupta,

Professor, Dept. of Computer Sc. and Engineering, IIT Kharagpur

pallab@cse.iitkgp.ernet.in

Matchings

- A ***matching*** of size *k* in a graph G is a set of *k* pairwise disjoint edges.
  - The vertices belonging to the edges of a matching are ***saturated*** by the matching; the others are ***unsaturated***.
  - If a matching saturates every vertex of G, then it is a ***perfect matching*** or ***1-factor***.

Alternating Paths

- Given a matching M, an ***M-alternating path*** is a path that alternates between the edges in M and the edges not in M.
  - An M-alternating path P that begins and ends at M-unsaturated vertices is an ***M-augmenting path***
  - Replacing $M \cap E(P)$ by $E(P) - M$ produces a new matching M' with one more edge than M.

Symmetric Difference

- If G and H are graphs with vertex set V, then the ***symmetric difference***, $G \Delta H$ is the graph with vertex set V whose edges are all those edges appearing in exactly one of G and H.
  - If M and M′ are matchings, then $M \,\Delta\, M' = (M \cup M') - (M \cap M')$

Key result

- **A matching M in a graph G is a *maximum matching* in G iff G has no M-augmenting path.**

Bipartite Matching

When G is a bipartite graph with bipartition *X,Y* we may ask whether G has a matching that saturates *X.*

- We call this a matching of *X into Y*.

Results…

- [Hall's Theorem: 1935]

  If G is a bipartite graph with bipartition *X,Y*, then G has a matching of *X* into *Y* if and only if $|N(S)| \geq |S|$ for all $S \subseteq X$.

- For $k>0$, every k-regular bipartite graph has a perfect matching.

Vertex Cover & Bipartite Matching

- A *vertex cover* of G is a set *S* of vertices such that *S* contains at least one endpoint of every edge of G.
  - The vertices in S *cover* the edges of G.

- [König and Egerváry: 1931]

  If G is a bipartite graph, then the maximum size of a matching in G equals the minimum size of a vertex cover of G.

Edge Cover

- An ***edge cover*** **of G is a set of edges that cover the vertices of G.**
  - **only graphs without isolated vertices have edge covers.**

Notation…

- We will use the following notation for independence and covering problems

$\alpha(G)$: maximum size of independent set

$\alpha'(G)$: maximum size of matching

$\beta(G)$: minimum size of vertex cover

$\beta'(G)$: minimum size of edge cover

Min-max Theorems

- In a graph G, $S \subseteq V(G)$ is an independent set if and only if S' is a vertex cover, and hence $\alpha(G) + \beta(G) = n(G)$.

- If G has no isolated vertices, then $\alpha'(G) + \beta'(G) = n(G)$.

- If G is a bipartite graph with no isolated vertices, then $\alpha(G) = \beta'(G)$ (max independent set = min edge cover)

Augmenting Path Algorithm

Input: A bipartite graph G with a bipartition *X,Y,* a matching M in G, and the set U of all *M*-unsaturated vertices in *X.*

Idea:

- Explore *M*-alternating paths from U, letting $S \subseteq X$ and $T \subseteq Y$ be the sets of vertices reached.
- *Mark* vertices of *S* that have been explored for extending paths.
- For each $x \in (S \cup T) - U$, record the vertex before *x* on some *M*-alternating path from U.

Augmenting Path Algorithm

Initialization: Set $S=U$ and $T=\phi$

Iteration:

- If S has no unmarked vertex, the stop and report $T \cup (X-S)$ as a minimum cover and M as a maximum matching.
- Otherwise, select an unmarked $x \in S$.
- To explore *x*, consider each $y \in N(x)$ such that $xy \notin M$.
  If *y* is unsaturated, terminate and trace back from *y* to report an *M*-augmenting path from U to *y*. Otherwise, *y* is matched to some $w \in X$ by M. In this case, include *y* in T and *w* in S.
- After exploring all such edges incident to *x*, mark *x* and iterate.

Augmenting Path Algorithm

- Repeated application of the Augmenting Path Algorithm to a bipartite graph produces a matching and vertex cover of the same size.
  - The complexity of the algorithm is $O(n^3)$.
  - Since matchings have at most n/2 edges, we apply the augmenting path algorithm at most n/2 times.
  - In each iteration, we search from a vertex of X at most once, before we mark it. Hence each iteration is O(e(G)), which is $O(n^2)$.

Weighted Bipartite Matching

- A *transversal* of an n X n matrix A consists of n positions – one in each row and each column.
  - Finding a transversal of A with maximum sum is the *assignment problem*.
  - This is the matrix formulation of the *maximum weighted matching problem*, where A is the matrix of weights w_{ij} assigned to the edges $x_i y_j$ of $K_{n,n}$ and we seek a perfect matching M with maximum total weight $w(M)$.

Minimum Weighted Cover

- **Given the weights $\{w_{ij}\}$, a *weighted cover* is a choice of labels $\{u_i\}$ and $\{v_j\}$ such that $u_i + v_j \geq w_{ij}$ for all *i,j.***

- **The *cost c(u,v)* of a cover *u,v* is $\Sigma\, u_i + \Sigma\, v_j$.**

- **The *minimum weighted cover problem* is the problem of finding a cover of minimum cost.**

Min Cover & Max Matching

- If M is a perfect matching in a weighted bipartite graph G and u,v is a cover, then $c(u,v) \geq w(M)$.
  - Furthermore, $c(u,v) = w(M)$ if and only if M consists of edges $x_i y_j$ such that $u_i + v_j = w_{ij}$. In this case, M is a maximum weight matching and u,v is a minimum weight cover.

Hungarian Algorithm

Input: A matrix of weights on the edges of $K_{n,n}$ with bipartition X,Y.

Idea: Maintain a cover u,v, iteratively reducing the cost of the cover until the equality subgraph, $G_{u,v}$ has a perfect matching.

Initialization: Let u,v be a feasible labeling, such as $u_i = max_j\, w_{ij}$ and $v_j = 0$, and find a maximum matching M in $G_{u,v}$.

Hungarian Algorithm

Iteration:

- If M is a perfect matching, stop and report M as a maximum weight matching.
- Otherwise, let U be the set of M-unsaturated vertices in X.
- Let S be the set of vertices in X and T the set of vertices in Y that are reachable by M-alternating paths from U. Let

$$\varepsilon = \min\{u_i + v_j - w_{ij} : x_i \in S,\ y_j \in Y - T\}$$

- Decrease u_i by ε for all $x_i \in S$, and increase v_j by ε for all $y_j \in T$. If the new equality subgraph G′ contains an M-augmenting path, replace M by a maximum matching in G′ and iterate. Otherwise, iterate without changing M.

Hungarian Algorithm

- **The Hungarian Algorithm finds a maximum weight matching and a minimum cost cover.**

Stable Matchings

Given *n* men and *n* women, we wish to establish *n* stable marriages.

- If man *x* and woman *a* prefers each other over their existing partners, then they might leave their current partners and switch to each other.
- In this case we say that the unmatched pair *(x,a)* is an unstable pair.
- A perfect matching is a *stable matching* if it yields no unstable matched pair.

Gale-Shapley Proposal Algorithm

Input: Preference rankings by each of *n* men and *n* women.

Iteration:

- Each man proposes to the highest woman on his preference list who has not previously rejected him.
- If each woman receives exactly one proposal, stop and use the resulting matching.
- Otherwise, every woman receiving more than one proposal rejects all of them except the one that is highest on her preference list.
- Every woman receiving a proposal says "*maybe*" to the most attractive proposal received.

The algorithm produces a stable matching.

Matchings in General Graphs

- A *factor* of a graph G is a spanning sub-graph of G.
- A *k-factor* is a spanning *k-regular* sub-graph.
- An *odd component* of a graph is a component of odd order; the number of odd components of H is o(H).

- [Tutte 1947]:
  A graph G has a 1-factor iff $o(G - S) \leq |S|$ for every $S \subseteq V(G)$.

- [Peterson 1891]:
  Every 3-regular graph with no cut-edge has a 1-factor.

Edmond’s Blossom Algorithm

- Let M be a matching in a graph G, and let *u* be an M-unsaturated vertex.
- A ***flower*** is the union of two *M*-alternating paths from *u* that reach a vertex *x* on steps of opposite parity.
- The ***stem*** of the flower is the maximal common initial path.
- The ***blossom*** of the flower is the odd cycle obtained by deleting the stem.

Edmond's Blossom Algorithm

Input: A graph G, a matching M in G, and an M-unsaturated vertex *u*.

Initialization: S = {*u*} and T = { }

Iteration:

- Is S has no unmarked vertex, stop
- Otherwise, select an unmarked vertex $v \in S$. To explore from *v*, successively consider each $y \in N(v)$ such that $y \notin T$.
- If *y* is unsaturated by M, then trace back from *y* to report an M-augmenting *u,y*-path.
- If $y \in S$, then a blossom has been found. Contract the blossom and continue the search from this vertex in the smaller graph.
- Otherwise, *y* is matched to some *w* by M. Include *y* in *T* (reached from *v*), and include *w* in *S*.
- After exploring all such neighbors of *v*, mark *v* and iterate.