Das Heiratsproblem

Szenario:

- Gegeben: n Frauen und $m > n$ Männer.
- Bekanntschaftsbeziehungen zwischen allen Männern und Frauen.

Fragestellung:

- Wann gibt es für jede der Frauen einen Heiratspartner?
- Modellierung als bipartiter Graph $K_{n,m}$.
- Gesucht ist knotendisjunkte Kantenmenge E der Größe n.

Definition Matching

Sei $G = (V, E)$ ein Graph. Ein *Matching* $M \subseteq E$ ist eine knotendisjunkte Kantenmenge, d.h. dass für alle verschiedenen $e, e' \in M$ gilt $e \cap e' = \emptyset$. Die Größe von M ist $|M|$.

1. M *überdeckt* ein $v \in V$ gdw $v \cap e \neq \emptyset$ für ein $e \in M$.
2. M ist *perfekt*, falls M alle Knoten überdeckt, d.h. falls $|M| = \frac{|V|}{2}$.

Heiratssatz

Satz von Hall

Sei $G = (A \dot\cup B, E)$ bipartit mit Knotenpartitionen $A \dot\cup B$. G enthält ein Matching M der Größe $|M| = |A|$ gdw $|X| \leq |\bigcup_{x \in X} \Gamma(x)|$ für alle $X \subseteq A$.

- Sei $\Gamma(X) = \bigcup_{x \in X} \Gamma(x)$ für alle $X \subseteq A$
- "$\Rightarrow$": Sei M ein Matching der Größe $|M| = |A|$.
- Wir betrachten den vom Matching definierten schwachen Teilgraph $G' = (A \dot\cup B, M)$.
- In G' besitzt jedes $X \subseteq A$ genau $|X|$ Nachbarn.
- Damit besitzt in G jedes $X \subseteq A$ mindestens $|X|$ Nachbarn.

Rückrichtung des Heiratssatzes

- "$\Leftarrow$": **Annahme:** G besitzt maximales Matching M mit $|M| < |A|$.
- Dann existiert ein nicht-überdecktes $a_1 \in A$.
- Wegen $|\Gamma(a_1)| \geq 1$ besitzt a_1 mindestens einen Nachbarn.
- Falls $b_1 \in \Gamma(a_1)$ nicht-überdeckt ist, füge $\{a_1, b_1\}$ zu M hinzu. (Widerspruch: M ist ein maximales Matching.)
- Sonst wähle überdecktes b_1 und starte folgenden Algorithmus AUGMENTIERENDER-PFAD.

Algorithmus AUGMENTIERENDER-PFAD

Algorithmus AUGMENTIERENDER-PFAD

EINGABE: $G = (A \dot{\cup} B, E)$, M, a_1, b_1

1. $k \leftarrow 1$
2. While (b_k wird von M überdeckt)
   1. $a_{k+1} \leftarrow$ Nachbar von b_k im Matching M
   2. If ($\exists$ nicht-überdecktes $v \in \Gamma(\{a_1, \ldots, a_{k+1}\}) \setminus \{b_1, \ldots, b_k\}$)
      1. $b_{k+1} \leftarrow v$
      2. $b_{k+1} \leftarrow$ beliebiges $v \in \Gamma(\{a_1, \ldots, a_{k+1}\}) \setminus \{b_1, \ldots, b_k\}$
   3. $k \leftarrow k + 1$.

AUSGABE: augmentierender Pfad $p_a = (a_1, b_1, \ldots, a_k, b_k)$

Korrektheit von AUGMENTIERENDER-PFAD

Korrektheit:

- Zeigen zunächst, dass b_{k+1} in Schritt 2.2 stets existiert. Es gilt
$$|\Gamma(\{a_1, \ldots, a_{k+1}\}) \setminus \{b_1, \ldots, b_k\}| \geq (k+1) - k = 1.$$
- Für alle $\{a_i, b_i\}$, $i \in [k]$ gilt, dass $\{a_i, b_i\} \in E \setminus M$.
- Damit sind diese k Kanten nicht im Matching enthalten.
- Die $k-1$ Kanten $\{b_i, a_{i+1}\}$ sind dagegen alle in M.
- D.h. durch Entfernen aller $\{b_i, a_{i+1}\}$ aus M und Hinzunahme aller $\{a_i, b_i\}$ zu M wird das Matching um Eins vergrößert.

  (Widerspruch: M ist nach Annahme maximal.)

Konstruktion eines maximalen Matchings

Algorithmus Maximales-Matching

EINGABE: $G = (A \cup B, E)$ mit $|X| \leq |\Gamma(X)|$ für alle $X \subseteq A$

1. $M \leftarrow \emptyset$
2. While (es gibt ein nicht-überdecktes $a_1 \in A$)
   1. $b_1 \leftarrow$ Nachbar von a_1
   2. $p = (a_1, b_1, \ldots, a_k, b_k) \leftarrow$ AUGMENTIERENDER-PFAD(G, M, a_1, b_1)
   3. For $i \leftarrow 1$ to k
      1. $M \leftarrow M \cup \{a_i, b_i\}$; If $(i > k)$ $M \leftarrow M \setminus \{b_i, a_{i+1}\}$;

AUSGABE: Matching M mit $|M| = |A|$

Korrektheit und Laufzeit:

- M wird in jeder Iteration um ein Element vergrößert.
- Nach $|A|$ Iterationen gilt $|M| = |A|$.

k-reguläre bipartite Graphen

Satz Perfekte Matchings für bipartite Graphen

Sei $G = (A \dot\cup B, E)$ ein k-regulärer bipartiter Graph. Dann gilt:

1. G besitzt ein perfektes Matching.
2. Der chromatische Index von G beträgt $\chi'(G) = k$.

Beweis: von (1)

- Falls $|X| \leq |\Gamma(X)|$ für alle $X \subseteq A$, verwende Satz von Hall.
- Liefert Matching für alle $a \in A$. Falls $|A| = |B|$, dann ist M perfekt.
- In jedem k-regulären bipartiten $G = (A \dot\cup B, E)$ gilt $|A| = |B|$, denn
$$|E| = \sum_{a \in A} \deg(a) = |A| \cdot k = \sum_{b \in B} \deg(b) = |B| \cdot k.$$
- **Annahme:** Es existiert ein $X \subseteq A$ mit $|\Gamma(X)| < |X|$.
- Wir betrachten die Multimenge $M = \bigcup_{x \in X} \Gamma(x)$.
- Da G ein k-regulärer Graph ist, gilt $|M| = k \cdot |X|$.
- Schubfachprinzip: $\exists b \in \Gamma(X)$ mit $\deg(b) \geq \left\lceil \frac{|M|}{|\Gamma(X)|} \right\rceil > \frac{k \cdot |X|}{|X|} = k$.
(Widerspruch: G ist k-regulär, d.h. $deg(b) = k$ für alle $b \in B$.)

$\chi'(G) = k$ in k-regulären bipartiten Graphen

Beweis: von (2)

- Wir beweisen $\chi'(G) = k$ per Induktion über k.
- **IV** für $k = 1$: G besitzt nach (1) ein perfektes Matching M.
- Daher sind alle $a \in A$ mit genau einem $b \in B$ verbunden.
- D.h. $M = E$ und alle Kanten können mit Farbe 1 gefärbt werden.
- **IS** $k - 1 \rightarrow k$: G besitzt nach (1) ein perfektes Matching M.
- $G' = (A \dot{\cup} B, E \setminus M)$ ist $(k-1)$-regulär.
- D.h. G' besitzt nach IA eine $(k-1)$-Kantenfärbung.
- Wir färben alle Kanten des Matchings M mit der Farbe k.

Gerichtete Graphen

Definition Gerichteter Graph bzw. Digraph

Ein *gerichteter Graph bzw. Digraph* ist ein Tupel $D = (V, E)$ mit

- der Knotenmenge $V = \{v_1, \ldots, v_n\}$ und
- der Kantenmenge $E = \{(u, v) \subseteq V \times V \mid u \neq v\}$.

Anmerkung: Wir definieren analog zu ungerichteten Graphen

- Weg, Pfad, u-v Pfad, Kreis, kürzester Pfad
- Nachbarschaft $\Gamma(v) = \{u \in V \mid (v, u) \in E\}$.

Definition DAG

Sei $D = (V, E)$ ein Digraph. D heißt *DAG* (Directed Acyclic Graph), falls D kreisfrei ist.

DAGs und die topologische Sortierung

Algorithmus TOPOLOGISCHE SORTIERUNG (DFS-Variante)

EINGABE: DAG $D = (V, E)$ in Adjazenzlistendarstellung

1. For alle $v \in V$: $\text{pred}[v] \leftarrow$ nil; $f[v] \leftarrow 0$;
2. $i \leftarrow n$; $S \leftarrow$ new Stack;
3. While (es gibt unbesuchte Knoten $s \in S$)
   1. $s \leftarrow$ minimaler unbesuchter Knoten; S.Push(s);
   2. While (S.Isempty() $\neq$ TRUE)
      1. $v \leftarrow S$.Pop();
      2. If (es gibt ein $u \in \Gamma$ mit $\text{pred}[u] =$nil) then
         S.Push(v); S.Push(u);
         $\text{pred}[u] \leftarrow v$;
      3. else if ($f[v] = 0$) then
         $f[v] \leftarrow i$; $i \leftarrow i - 1$;

AUSGABE: $\text{pred}[v], f[v]$ für alle $v \in V$

Finish-Zahl $f[v]$

Satz Finish-Zahl

Sei $D = (V, E)$ ein DAG und sei $f[v]$ für alle $v \in V$ definiert durch Anwendung von TOPOLOGISCHE SORTIERUNG auf D. Für $u, v \in V$ gilt: Falls $f[u] > f[v]$, dann folgt $(u, v) \notin E$.

Beweis:

- **Annahme:** $(u, v) \in E$
- Es gibt keinen v-u Pfad in G, denn dieser würde mit (u, v) zusammen einen Kreis schließen.
- Jedem $w \in V$ wird ein Wert $f[w] > 0$ zugewiesen, sobald alle Nachbarn von w besucht sind und w vom Stack entfernt wird. (Fortsetzung auf nächster Folie.)

Finish-Zahl $f[v]$

- **Fall 1:** Knoten v wird vor Knoten u besucht.
- Damit wird v vom Stack entfernt, bevor u auf den Stack kommt.
- Da der Zähler i für die Finishzahl dekrementiert wird, gilt $f[v] > f[u]$. (Widerspruch: $f[u] > f[v]$ nach Voraussetzung)
- **Fall 2:** Knoten u wird vor Knoten v besucht.
- Wegen $\{u, v\} \in E$ liegt u im Stack unter v.
- Sobald v vom Stack entfernt wird, erhält v ein $f[v] > 0$.
- Danach erst wird u eine Finish-Zahl $f[u]$ zugewiesen.
- Daraus folgt wiederum $f[u] < f[v]$. (Widerspruch)

Anwendung der Topologischen Sortierung

Szenario:

- **Gegeben:** n Aufgaben und Abhängigkeiten $(u, v) \in [n]^2$ mit der Bedeutung, dass u vor v ausgeführt werden muss.
- **Gesucht:** Gültige Reihenfolge, alle Aufgaben abzuarbeiten.

- Modellierung als Digraph D mit gerichteten Kanten (u, v).
- Falls D kein DAG ist, dann existiert ein Kreis in D. Dann sind die Abhängigkeiten widersprüchlich und es existiert keine Lösung.
- Sei also D ein DAG. Dann kann mit TOPOLOGISCHE SORTIERUNG für jeden Knoten $v \in V$ eine Finish-Zahl $f[v]$ berechnet werden.
- Ordne die Knoten nach steigender Finish-Zahl $f[v]$.
- Für $d[u] > d[v]$ gilt $\{u, v\} \notin E$.
- D.h. Aufgabe u kann nach Aufgabe v ausgeführt werden.
- Damit können alle Aufgaben in der Reihenfolge steigender Finish-Zahl durchgeführt werden.