Datenstrukturen und effiziente Algorithmen

Markus Vieth David Klopp

19. Januar 2016

Inhaltsverzeichnis

1 Vorlesung

1.1 Das Heiratsproblem - Maximum cardinality matching in bipartiten Graphen

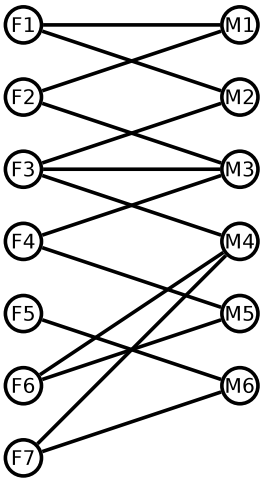


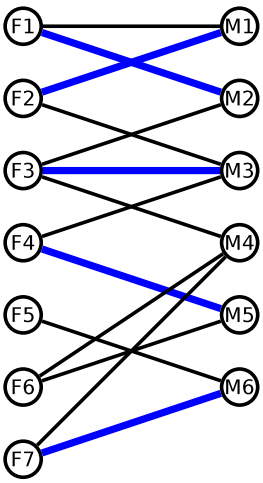
Abbildung 1.1: Ausgangsproblem

$$G = (V_1 \dot{\cup} V_2, E)$$

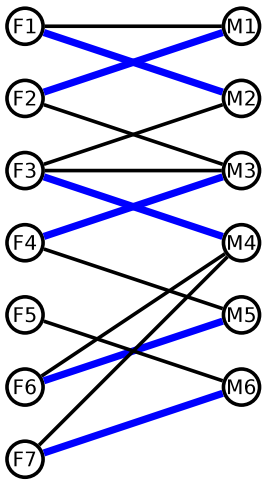
$E \subseteq E$ heißt Matching, wenn jeder Knoten zu höchstens einer Kante aus M inzident ist. Freie Knoten sind an keiner Matching-Kante beteiligt.

M heißt maximales Matching, wenn M durch Hinzunahme einer weiteren Kante nicht vergrößert werden kann.

Gesucht ist ein maximum-Matching M^* mit $|M^*| \geq |M| \; \forall \; M$ Matching.



(a) Nicht optimales Matching



(b) optimales Matching

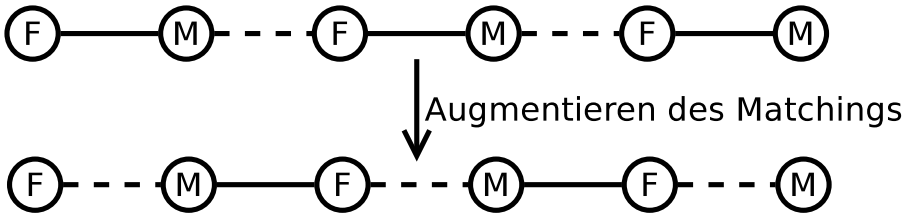


Abbildung 1.3: Alternierender Pfad

Alternierender Graph, der mit einem Singleknoten startet und endet, nennt man einen augmentierten Pfad.

Zum finden eines augmentierten Pfades verwenden wir folgenden Graphen
$G_M = (V_1 \cup V_2 \cup \{s\}, E')$

$$E' = \{(v_1, v_2) | v_1 \in V_1, v_2 \in V_2, (v_1, v_2) \in E \setminus M\}$$
$$\cup \{(v_2, v_1) | v_1 \in V_1, v_2 \in V_2, (v_1, v_2) \in M\} \cup \{(s, v_1) | v_1 \in V_1 \text{ frei}\}$$

Mit Hilfe von BFS oder DFS können wir in G_M augmentierende Pfade leicht finden. Also

$$\text{Zei } \mathcal{O}(|V| + |E|)$$

1.1.1 Lemma: (Berge)

Ein Matching M ist ein maximum-Matching $\Leftrightarrow$ Es gibt keinen M-augmentierenden Pfad.

1.1.2 Beweis:

"$A \Rightarrow B$"

$\neg B \Rightarrow \neg A$ Es gibt M-augm. Pfad $\Rightarrow$ M ist kein maximum Matching

"$A \Leftrightarrow B$"

$\neg A \Rightarrow \neg B$ Sei M noch kein maximum Matching.

z.z. Es gibt ein M-augm. Pfad
M^* sei ein maximum Matching, d.h. $|M^*| > |M|$.
Betrachte den Graphen $\tilde{G} = (V_1 \cup V_2, M \oplus M^*)$
Alle Knoten in $\tilde{G}$ haben höchstens Grad 2, ansonsten wäre ein Knoten inzident zu zwei Kanten aus dem Gleichen Matching M oder M^*.
$\tilde{G}$ besteht aus einzelnen Knoten, Pfaden gerader oder ungerader Länge und Zyklen gerader Länge.

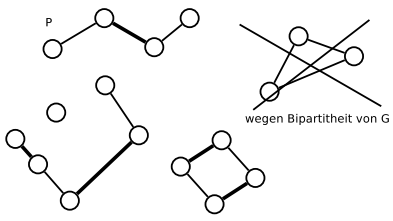


Abbildung 1.4: Beispiel

z.z. Es gibt in $\tilde{G}$ mindestens einen M-augment. Pfad p, der mehr Kanten aus M^* als aus M besitzt.
Dies gilt, weil ansonsten $|M^*| \leq |M|$

q.e.d.

1.1.3 Pseudo-Code

```
1  M = ∅;
2  do {
3      P = findAugmPath(G_M);
4      if (P = = NULL) break;
5      M = M ⊕ P;
6  } while(true);
```

Wiederholung: Symmetrische Differenz

$$A \oplus B = A \setminus B \cup B \setminus A$$

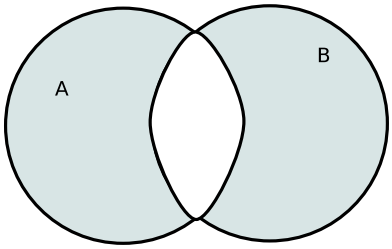


Abbildung 1.5: Symmetrische Differenz

1.2 Laufzeit

$$\begin{aligned} &\mathcal{O}(\min(|V_1|, |V_2|) \cdot (|V| + |E|) \\ &= \mathcal{O}(|V| \cdot |E|) \end{aligned}$$

1.3 Hopcroft-Karp-Algorithmus

erzielt Laufzeit von $\mathcal{O}(\sqrt{|V|} \cdot |E|)$

```
1  M = ∅;
2  do {
3      G_L = buildLevelGraph(G_M);        //Knotendisjunkte M-augm. Pfade
4      𝒫 = findAugmPath(G_L);             //𝒫 = P_1 ∪̇ P_2 ∪̇ ... ∪̇ P_k
5      M = M ⊕ \mathcal{P};
6  } while(𝒫 ≠ ∅);
```

G_L kann mittels BFS in Zeit $\mathcal{O}(|V| + |E|)$ konstruiert werden. Zum Auffinden einer maximalen Menge von M-augmentierenden Pfaden in G_L verwenden wir DFS und entfernen jedes mal den gefunden PfadP_i aus G_L. DFS sorgt dafür, dass P_i in Zeit $\mathcal{O}(|P_i|)$ gefunden und gelöscht werden kann.
$\Rightarrow$ `findAugmPaths(`G_L`)` hat nur Laufzeit $\mathcal{O}(|E|)$

Abbildungsverzeichnis