Effiziente Algorithmen (SS2015)

Kapitel 3
Matchings

Walter Unger

Lehrstuhl für Informatik 1

14:01 Uhr, den 19. Dezember 2018

Inhalt I

Motivation und Anwendungen

- vielfältige Zuordnungsprobleme:
  - Kunden auf Kundenberater
  - Anrufe im Callcenter auf Servicemitarbeiter
  - Heiratsvermittlung
  - Jobs auf Maschinen
  - Schwarzgeld auf Konten
- Grundsätzliche Probleme:
  - Bestimme größtes Matching.
  - Bestimme nicht erweiterbares Matching.
  - Bestimme größtes Matching mit Kantengewichten.
  - Bestimme günstigstes Matching (d.h. Kosten auf den Kanten) unter allen maximum Matchings.
- Im Folgenden: $G = (V, E)$ zusammenhängend, $|V| = n$, $|E| = m$
- Im Folgenden: $G = (V, W, E)$ bipartit und zusammenhängend, $|V \cup W| = n$, $|E| = m$

Definitionen

$\delta_G(v) = |N_G(v)|$ $N_G(v) = \{w \in V(G) \mid \{v, w\} \in E(G)\}$

Definition (Matching)

Sei $G = (V, E)$ ungerichteter Graph. $M \subseteq E$ heißt Matching:

$$\forall e, f \in M : e \cap f = \emptyset$$

D.h. $\forall v \in V : \delta_{G'}(v) \leqslant 1$ mit $G' = (V, M)$.

Definition ((inklusions-)maximales Matching)

Sei $G = (V, E)$ ungerichteter Graph. $M \subseteq E$ heißt maximales Matching:

$$\forall M' : M \subsetneq M' \subset E : M' \text{ ist kein Matching.}$$

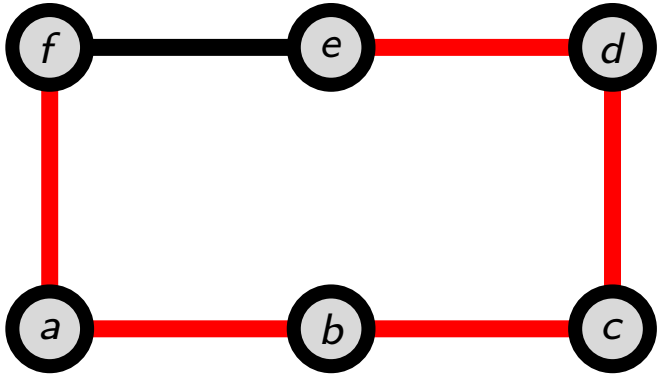
Definition (maximum Matching)

Sei $G = (V, E)$ ungerichteter Graph. $M \subseteq E$ heißt maximum Matching:

$$\forall M' \subset E : |M| < |M'| : M' \text{ ist kein Matching.}$$

Beispiel

- Betrachte Graph:
- $\{\{a, b\}\}$ Matching
- $\{\{a, b\}, \{b, c\}\}$ kein Matching
- $\{\{a, f\}, \{c, d\}\}$ maximales Matching
- $\{\{a, f\}, \{b, c\}, \{e, d\}\}$ maximum Matching



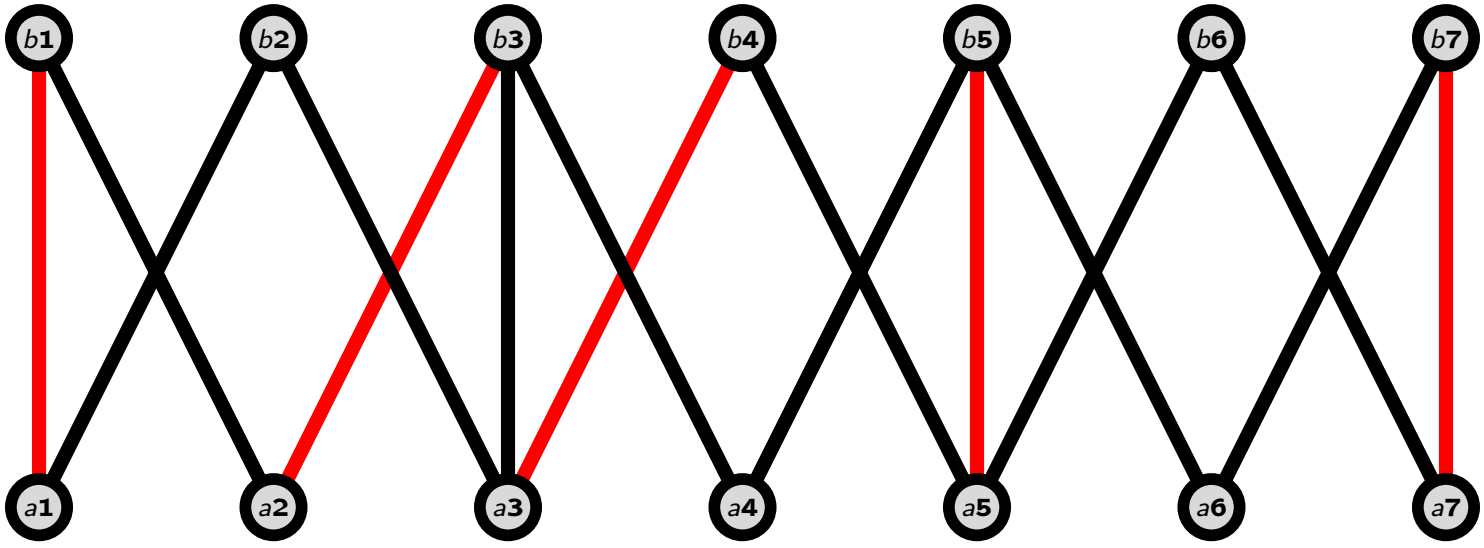
Algorithmus (maximales Matching)

Idee: Greedy, d.h. solange es geht, wähle Kante für das Matching.

1. Gegeben $G = (V, E)$
2. $M = \emptyset$
3. Solange E nicht leer, mache:
   1. Wähle $e \in E$.
   2. Setze $M = M \cup \{e\}$
   3. Setze $E = \{e' \in E \mid e' \cap e = \emptyset\}$
4. Ausgabe: M.

Beispiel

Idee: Greedy Algorithmus:



Probleme

Definition (Bipartites Matchingproblem)

Gegeben: $G = (V, W, E)$ bipartiter Graph.

Gesucht: Maximum Matching auf G.

Definition (Bipartites kostenminimales Matchingproblem)

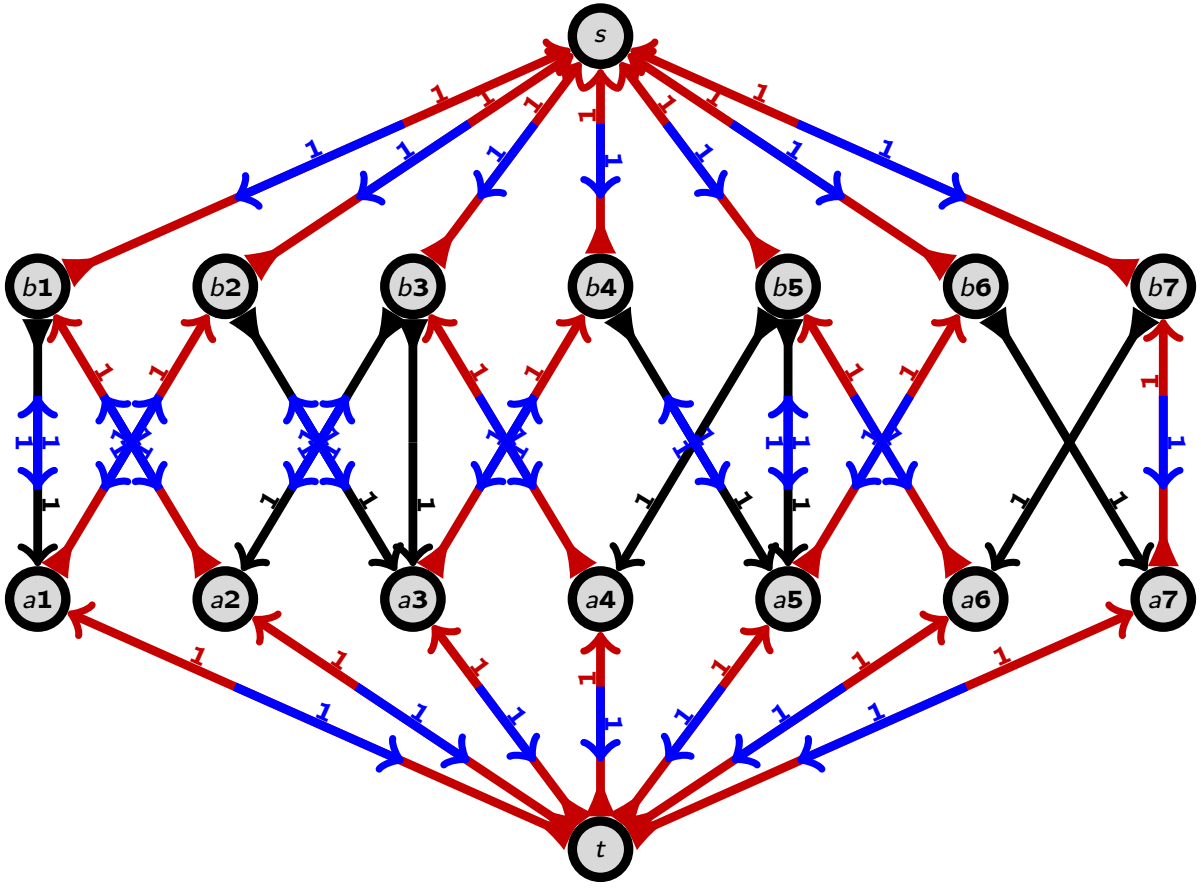
Gegeben: $G = (V, W, E)$ bipartiter Graph und Kostenfunktion $l : E \mapsto \mathbb{N}$.

Gesucht: Kostenminimales maximum Matching auf G.

Idee (bipartites Matching und Flüsse)

- Matching von V nach W entspricht "Transport".
- Jeder Knoten kann eine Einheit transportieren.
- Jede Matchingkante kann eine Einheit transportieren.
- Modelliere das durch Fluss, erweitere Graph:
  - Füge Quelle s ein und verbinde s zu jedem Knoten aus V.
  - Füge Senke t ein und verbinde jeden Knoten aus W zu t.
  - Diese neuen Kanten können eine Einheit transportieren.

Bipartites Matching und Flüsse



Formale Transformation

- Sei $G = (V, W, E)$ bipartiter Graph.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$
  - $E_t = \{(w, t) \mid w \in W\}$
- Setze $c : E' \mapsto \mathbb{N}$ mit $\forall e \in E' : c(e) = 1$.

Lemma

Das bipartite Matchingproblem kann in Zeit $O(n + m)$ auf das Flussproblem transformiert werden.
G hat Matching der Größe α genau dann, wenn $w(G') = \alpha$.

Theorem

Das maximum Matching Problem ist auf bipartiten Graphen in Zeit $O(n^3)$ lösbar.

Formale Transformation mit Kostenfunktion

- Sei $G = (V, W, E)$ bipartiter Graph und $l : E \mapsto \mathbb{Z}$.
- Bestimme $G' = (V \cup W \cup \{s, t\}, E')$ und $l' : E' \mapsto \mathbb{Z}$ mit:
  - $E' = E'' \cup E_s \cup E_t$
  - $E'' = \{(v, w) \mid \{v, w\} \in E \wedge v \in V \wedge w \in W\}$
  - $E_s = \{(s, v) \mid v \in V\}$ und $E_t = \{(w, t) \mid w \in W\}$.
  - $\forall (v, w) \in E'' : l'(v, w) = l(v, w)$.
  - $\forall e \in E_s \cup E_t : l'(e) = 0$.
- Setze $c : E' \mapsto \mathbb{N}$ mit $\forall e \in E' : c(e) = 1$.

Lemma

Das bipartite Matchingproblem mit Kosten kann in Zeit $O(n + m)$ auf das Flussproblem mit Kosten transformiert werden.

Theorem

G hat Matching der Größe α mit Kosten β genau dann, wenn $w(G') = \alpha$ und die Kosten des Flusses sind β.

Idee der Alternierenden Pfade

Betrachten wir die Berechnung eines Matchings auf bipartiten Graphen mit Hilfe von Flussalgorithmen genauer. Die Wirkung eines Flusspfades auf das Matching ist der Austausch von Matching und Nichtmatching Kanten. Diese ausgetauschten Kanten bilden dabei einen Pfad, der ein Matching vergrößert. Damit haben wir eine neue Idee, ein Matching schrittweise zu vergrößern.

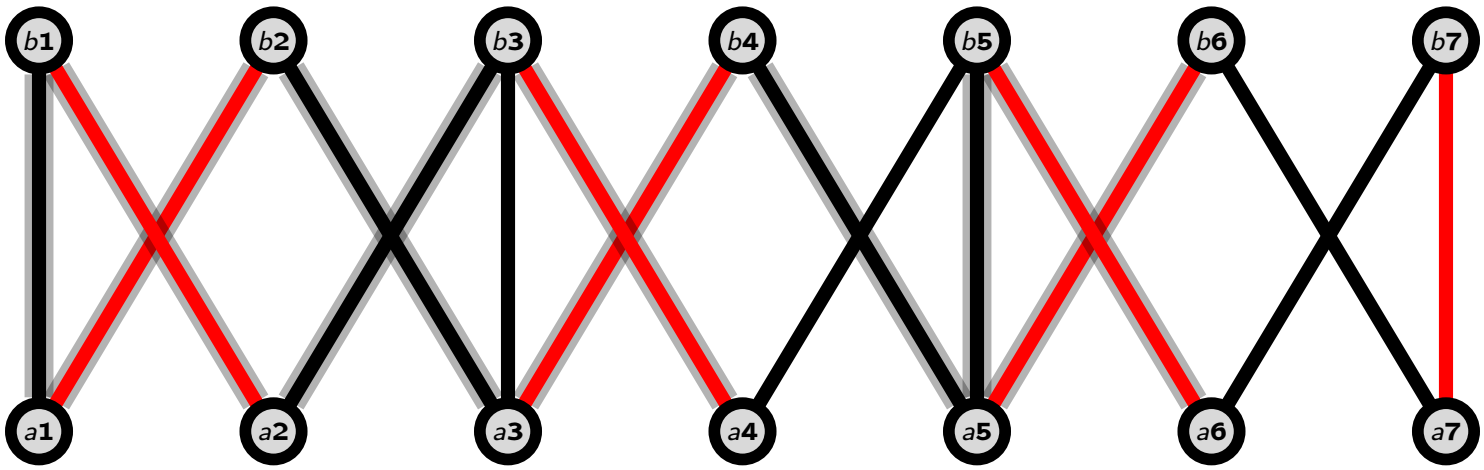
Aber es gibt noch eine zweite Möglichkeit, diese erweiternden alternierenden Pfade zu begründen. Betrachten wir zwei Matchings unterschiedlicher Größe. Wenn wir den Graphen betrachten, der aus diesen beiden Matchings besteht, so sehen wir ein System von Kreisen gerader Länge und Pfaden, welche keine gemeinsamen Knoten haben. Da das eine Matching mehr Kanten als das andere hat, muss es auch eine Komponente geben, wo das größere Matching mehr Kanten hat. Dies muss dann ein Pfad sein. Damit haben wir wieder einen alternierenden erweiternden Pfad. Diese Überlegung gilt auch für allgemeine Graphen.

Damit haben wir eine Methode, ein maximales Matching zu berechnen.

Nochmal das Beispiel

Betrachte das Beispiel ohne die Knoten s und t:

- Ein Fluss über eine Kante entspricht einer Matchingkante.
- Das Löschen einer Matchingkante entspricht einem Fluss über eine Rückwärtskante.
- Damit ein Fluss über eine Rückwärtskante fließt, muss er vorher und hinterher eine Vorwärtskante passieren.



Idee der alternierende Pfade

- Damit arbeitet der Flussalgorithmus wie folgt:
- Bestimme “erweiternden Pfad”:
  - Der Pfad startet an einem “freien” Knoten.
  - Der Pfad startet mit einer Vorwärtskante.
  - Der Pfad geht alternierend über Vorwärts- und Rückwärtskanten.
  - Der Pfad endet mit einer Vorwärtskante.
  - Der Pfad endet an einem “freien” Knoten.
- Damit können wir einen weiteren Algorithmus angeben.

Alternierende Pfade

$A \oplus B = (A \cup B) \setminus (A \cap B)$

- Sei $G = (V, E)$ ungerichteter Pfad und $M \subset E$ ein Matching.
- Ein Knoten $v \in V$ heißt frei, falls $v \notin \cup_{e \in M} e$.
- Ein Pfad $v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l$ heißt alternierend, falls $\{v_{i-1}, v_i\} \in M \Leftrightarrow \{v_i, v_{i+1}\} \notin M$ $(0 < i < l)$.

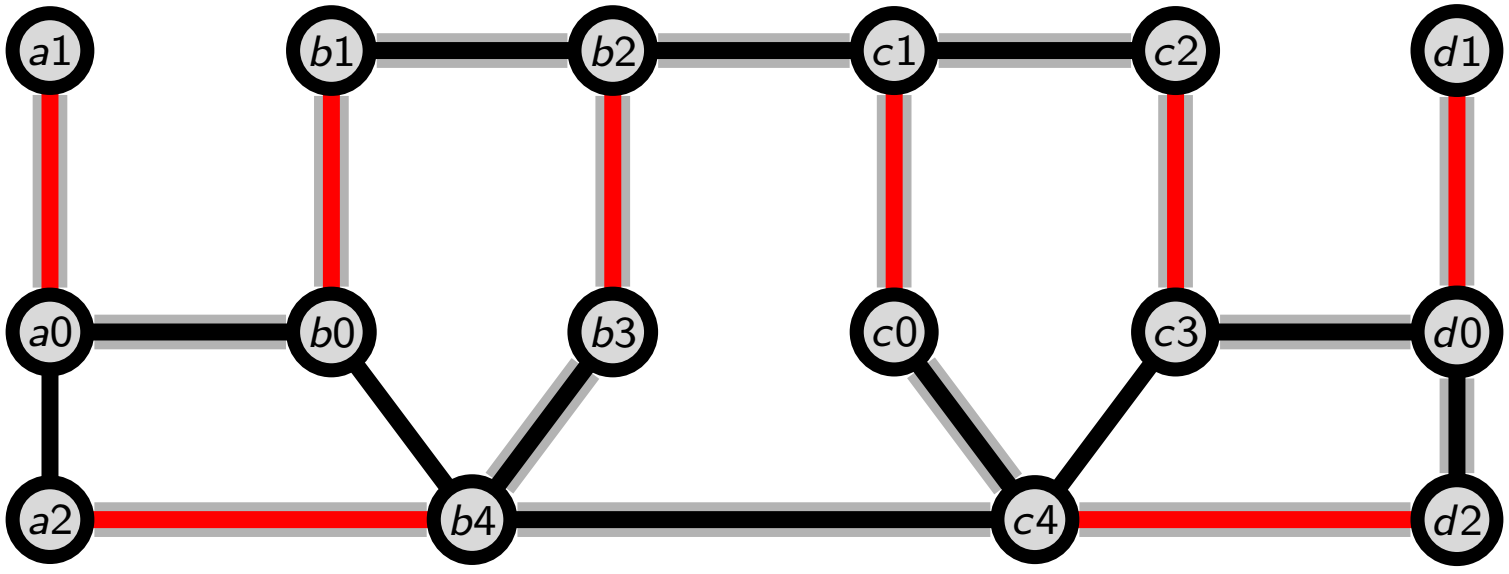
- Ein alterniernder Pfad
  $v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, v_{l-1}, \{v_{l-1}, v_l\}, v_l$ heißt erweiternd, falls v_0, v_l frei sind.

- Bemerkung: eine Kante zwischen freien Knoten ist ein verbessernder Pfad.
- Damit arbeitet der Algorithmus wie folgt:
  1. Setze $M = \emptyset$.
  2. Solange es erweiternden Pfad P gibt, mache:
     1. Erweitere M, d.h. $M = M \oplus E(P)$.

Beispiel allgemeiner Graph

$A \oplus B = (A \cup B) \setminus (A \cap B)$

Versuche verbessernde Pfade auf allgemeinen Graphen:



Ungerade Kreise können Probleme machen

Aussagen zu verbessernden Pfaden

$A \oplus B = (A \cup B) \setminus (A \cap B)$

Lemma (ein verbessernder Pfad)

Sei $G = (V, E)$ Graph, M Matching und P verbessernder Pfad. Dann ist $M \oplus E(P)$ Matching mit $|M \oplus E(P)| = |M| + 1$.

Beweis:

- Sei $G' = (V, M)$ und $G'' = (V, M \oplus E(P))$.
- Sei v der Startknoten von P, dann gilt $\delta_{G'}(v) = 0$ und $\delta_{G''}(v) = 1$.
- Sei w der Endknoten von P, dann gilt $\delta_{G'}(w) = 0$ und $\delta_{G''}(w) = 1$.
- Sei u ein Zwischenknoten auf P, dann gilt $\delta_{G'}(u) = \delta_{G''}(u) = 1$.
- Es wird eine Kante mehr hinzugefügt als entfernt.

Aussagen zu verbessernden Pfaden

$\delta(G) = \cup_{v \in V(G)} \delta_G(v)$

Lemma (Differenz zweier Matchings)

Sei $G = (V, E)$ Graph, M, N Matchings mit $|M| < |N|$. Dann enthält $H = (V, M \oplus N)$ mindestens $|N| - |M|$ knotendisjunkte verbessernde Pfade bezüglich M in G.

Beweis:

- Sei $G_M = (V, M)$ und $G_N = (V, N)$.
- Es gilt: $\delta(G_M) \subset \{0, 1\}$ und $\delta(G_N) \subset \{0, 1\}$.
- Damit folgt: $\delta(H) \subset \{0, 1, 2\}$.
- Seien $G_i = (V, E_i)$ $(1 \leqslant i \leqslant g)$ die Zusammenhangskomponenten von H.
- Wegen $\delta(G_i) \subset \{0, 1, 2\}$ folgt:
  - G_i ist ein isolierte Knoten oder
  - G_i ist ein Kreis gerader Länge oder
  - G_i ist ein Pfad gerader Länge oder
  - G_i ist ein Pfad ungerader Länge.
- Die Kanten in G_i stammen alternierend aus M und N.

Fortsetzung

$\delta(G) = \cup_{v \in V(G)} \delta_G(v)$

- Setze $d(G_i) = |E_i \cap N| - |E_i \cap M|$.
- Damit gilt: $d(G_i) \in \{-1, 0, 1\}$.
- Damit gilt weiter: $d(G_i) = 1$ falls G_i verbessernder Pfad.
- $\sum_{i=1}^{g} d(G_i) = |N \setminus M| - |M \setminus N| = |N| - |M|$
- Damit muss es mindestens $|N| - |M|$ Komponenten G_i geben mit $d(G_i) = 1$.
- Zusammengefasst gibt es mindestens $|N| - |M|$ verbessernde Pfade.
- Nach Definition der Komponenten sind diese knotendisjunkt.

Lemma

Sei $G = (V, E)$ und M Matching in G.
M ist maximum Matching genau dann, wenn keinen verbessernden Pfad bezüglich M gibt.

Algorithmus

1. Gegeben $G = (V, W, E)$
2. $M = \emptyset$
3. Solange es verbessernden Pfad P gibt, mache:
   1. Setze $M = M \oplus E(P)$
4. Ausgabe: M.

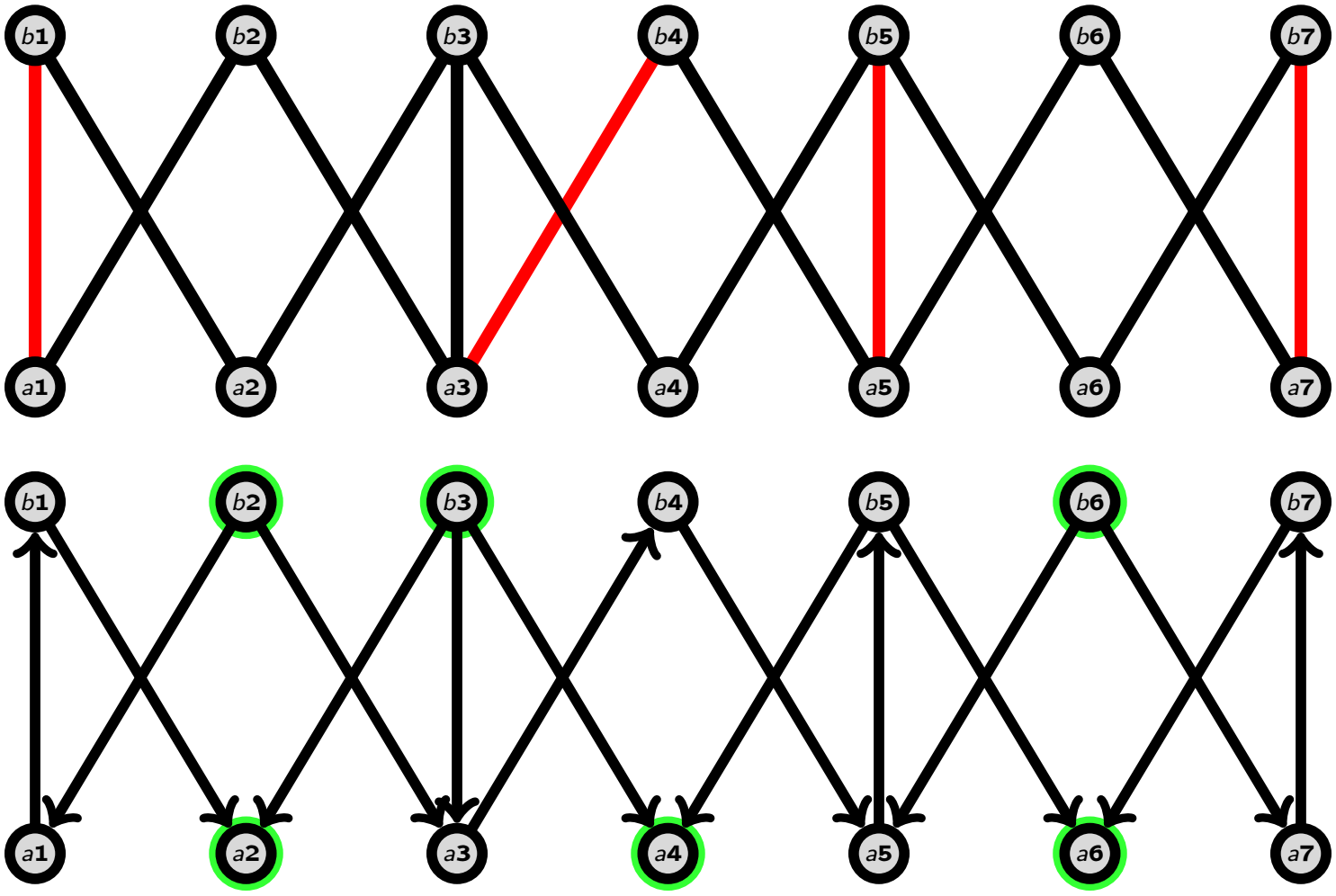
Theorem

Der obige Algorithmus hat eine Laufzeit von $O(n \cdot m)$.

Beweis:

- Wegen $|M| \leqslant \lfloor n/2 \rfloor$ gibt es höchstens $\lfloor n/2 \rfloor$ Schleifendurchläufe.
- Jede Schleife kann in Zeit $O(m)$ ausgeführt werden.

Beispiel (Schleifendurchlauf)



Schleifendurchlauf

- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  - $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  - $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$.
- Damit wird die Suche nach einem verbessernden Pfad eine Suche in G':
  - Von einem freien Knoten in V zu
  - einem freien Knoten in W.
- Damit ergibt sich eine Laufzeit von $O(m)$ durch Tiefensuche.

Idee zur Verbesserung der Laufzeit

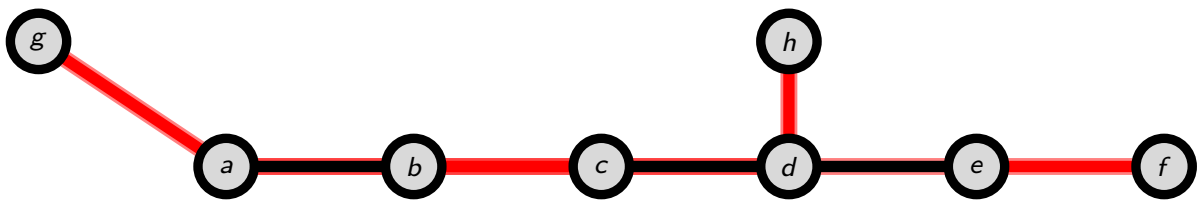
Zur Verbesserung der Laufzeit sollten, analog wie beim Flussproblem, mehr als ein alternierender Pfad gesucht werden. Damit die Suche strukturiert durchgeführt werden kann, wird ein System von kürzesten alternierenden Pfaden gesucht. Damit dieses System von alternierenden Pfaden sich nicht gegenseitig beeinflusst, wird ein System von knotendisjunkten Pfaden gesucht. Wir werden im Folgenden sehen, daß nach einem Vergrößern eines Matchings mit einem System von kürzesten alternierenden knotendisjunkten Pfaden, alle weiteren alternierenden knotendisjunkte Pfaden länger sind.

Danach muss noch abgeschätzt werden, wie oft so ein System von kürzesten alternierenden knotendisjunkten Pfaden bestimmt werden muss. Der Beweis besteht aus zwei Teilen (virtuellen Phasen der Berechnung), wobei nur der erste Teil nicht trivial ist. Dieser erste Teil (Phase) benutzt die folgende Idee:

Wenn uns noch x Kanten fehlen, um von einem Matching M der Größe g zu einem maximalen der Größe $x + g$ zu kommen, dann können die kürzesten verbessernden Pfade nicht zu lang sein. Dazu betrachtet man die gemeinsamen Kanten eines Pfades mit dem Matching M. Ein einfaches Durchschnittsargument zeigt, daß damit ein kürzester verbessernder Pfad höchstens x/g Kanten mit M gemeinsam hat.

Idee

- Falls Kanten oft die Rolle (Matching,nicht Matching) wechseln, erhalten wir eine schlechte Laufzeit.
- Am Anfang besteht der verbessernde Pfad aus einer Kante.
- Am Anfang sind die kürzesten verbessernden Pfade knotendisjunkt.
- Idee: suche kürzeste verbessernde Pfade.
- Falls diese knotendisjunkt sind, können diese gleichzeitig gesucht werden.
- Beispiel:



  - Matching: $M = \{\{b, c\}, \{d, e\}\}$
  - Erster kürzester verbessernder Pfad: a, b, c, d, e, f
  - Zweiter kürzester verbessernder Pfad: g, a, b, c, d, h
  - Widerspruch, da (g, a) noch kürzerer verbessernder Pfad.

Lemma 1

Lemma (kurzer Weg)

Sei M Matching in $G = (V, E)$ und P kürzester verbessernder Pfad in G bezüglich M. Sei weiter P' verbessernder Pfad bezüglich $M \oplus E(P)$, dann gilt:

$$|P'| \geqslant |P| + |P \cap P'|.$$

Beweis:

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.
- Nach Lemma "Differenz zweier Matchings" gibt es in $M \oplus N$ zwei knotendisjunkte verbessernde Pfade P_1, P_2.
- Wegen $|P_I| \geqslant |P|$ $(I \in \{1, 2\})$ gilt: $|M \oplus N| \geqslant 2 \cdot |P|$.
- Nun folgt:

Fortsetzung

Zeige: $|P'| \geqslant |P| + |P \cap P'|$

- Sei $N = (M \oplus P) \oplus P'$. Nach Lemma "ein verbessernder Pfad" gilt: $|N| = |M| + 2$.
- Nach Lemma "Differenz zweier Matchings" gibt es in $M \oplus N$ zwei knotendisjunkte verbessernde Pfade P_1, P_2.
- Wegen $|P_I| \geqslant |P|$ $(I \in \{1, 2\})$ gilt: $|M \oplus N| \geqslant 2 \cdot |P|$.
- Nun folgt:

$$
\begin{array}{rrcl}
 & M \oplus N & = & M \oplus ((M \oplus P) \oplus P') \\
 & & = & (M \oplus M) \oplus (P \oplus P') \\
 & & = & P \oplus P' \\
\text{Damit gilt:} & |P \oplus P'| & \geqslant & 2 \cdot |P| \\
\text{Betrachte:} & |P \oplus P'| & = & |P \cup P'| - |P \cap P'| \\
 & & \leqslant & |P| + |P'| - |P \cap P'| \\
\text{Damit gibt:} & 2 \cdot |P| & \leqslant & |P| + |P'| - |P \cap P'| \\
 & |P| + |P \cap P'| & \leqslant & |P'|
\end{array}
$$

Idee und Situation

- Ziel: Suche viele kürzeste verbessernde Pfade.
- Starte mit leerem Matching $M_0 = \emptyset$.
- Im Schritt i sei P_i ein kürzester verbessernder Pfad bezüglich M_{i-1}.
- Setze $M_i = M_{i-1} \oplus E(P_i)$.

Lemma (disjunkte kürzeste verbessernde Pfade)

Für $i, j \geqslant 1$ gilt:

1. $|P_i| \leqslant |P_{i+1}|$
2. $|P_i| = |P_j| \Longrightarrow P_i$ *und* P_j *sind knotendisjunkt.*

Beweis

1. $|P_i| \leqslant |P_{i+1}|$ folgt aus Lemma "kurzer Weg".
2. Folgt per Widerspruch:

Beweis (Teil 2)

- Sei $i < j$, $|P_i| = |P_j|$ und $V(P_i) \cap V(P_j) \neq \emptyset$.
- O.B.d.A.: $\forall k : i < k < j : \forall l \in \{i, j\} : V(P_l) \cap V(P_k) = \emptyset$.
- Damit folgt: P_j ist ein verbessernder Pfad bezüglich $M_i = M_{i-1} \oplus P_i$.
- Wegen $V(P_i) \cap V(P_j) \neq \emptyset$ folgt: $E(P_i) \cap E(P_j) \neq \emptyset$.
- Sei $v \in V(P_i) \cap V(P_j)$, genau die Matchingkante an v ist in $E(P_i) \cap E(P_j)$.
- $|E(P_i) \cap E(P_j)| \geqslant 1$.
- Aus Lemma "kurzer Weg" folgt: $|P_j| \geqslant |P_i| + |P_i \cap P_j| \geqslant |P_i| + 1$.
- Widerspruch.

Idee und Algorithmus

- Suche alle kürzesten verbessernden Pfade.
- Je länger diese werden, um so weniger gibt es davon.
- Alle knotendisjunkten Pfade nutzen jeweils mindestens eine Kante des aktuellen Matchings.
- Damit erhalten wir eine bessere Laufzeit.
- Der Algorithmus bestimmt in Runde i möglichst viele kürzeste verbessernden Pfade der Länge l_i.
- $l_1 < l_2 < l_3 < \ldots$.

Algorithmus:

1. Gegeben $G = (V, W, E)$ bipartiter Graph.
2. Setze $M = \emptyset$
3. Solange es verbessernde Pfade gibt, mache:
   1. Bestimme l als die Länge des kürzesten verbessernden Pfads.
   2. Bestimme eine inklusions-maximale Menge von verbessernden Pfaden $P_1, P_2, \ldots, P_i$ der Länge l.
   3. $M = M \oplus P_1 \oplus P_2 \oplus P_3 \oplus \ldots \oplus P_i$.

Anzahl der Schleifendurchläufe

Lemma (Pfadlänge)

Sei M' ein maximum Matching mit $s = |M'|$
und M ein nicht maximales Matching, dann gibt es einen M verbessernden Pfad der Länge höchstens

$$2 \cdot \left\lfloor \frac{|M|}{s - |M|} \right\rfloor + 1.$$

Beweis:

- Es gibt höchstens $s - |M|$ viele M-verbessernde Pfade.
- Diese Pfade enthalten zusammen höchstens $|M|$ Kanten aus M.
- Damit gibt es einen Pfad, der höchstens $\lfloor \frac{|M|}{s-|M|} \rfloor$ Kanten aus M enthält. D.h. ein kürzester Pfad enthällt micht mehr als die Durchschnittsanzahl von Kanten aus M.
- Dieser Pfad hat eine Länge von:

$$2 \cdot \left\lfloor \frac{|M|}{s - |M|} \right\rfloor + 1.$$

Anzahl der Schleifendurchläufe

M' **maximum Matching** $s = |M'|$ **und** $2 \cdot \left\lfloor \frac{|M|}{s-|M|} \right\rfloor + 1$

- Teile in zwei Phasen auf:
  - Phase 1 aktiv, solange $|M| \leqslant \lfloor s - \sqrt{s} \rfloor$ gilt.
    - Maximale Pfadlänge dann:

$$2 \cdot \left\lfloor \frac{\lfloor s - \sqrt{s} \rfloor}{s - \lfloor s - \sqrt{s} \rfloor} \right\rfloor + 1 \leqslant 2 \cdot \frac{s - \lceil \sqrt{s} \rceil}{\lceil \sqrt{s} \rceil} + 1 \leqslant 2 \cdot \sqrt{s} + 1$$

    - Damit gilt in jeder Runde der ersten Phase: $l \leqslant 2 \cdot \sqrt{s} + 1$.
    - Da l in Zweierschritten wächst, gibt es höchsten $\sqrt{s}$ Iterationen in der ersten Phase.
  - Phase 2 aktiv, falls $|M| > \lfloor s - \sqrt{s} \rfloor$ gilt.
    - Es fehlen $\sqrt{s}$ viele Kanten.
    - Die Phase endet nach maximal $\sqrt{s}$ Iterationen.

Laufzeit

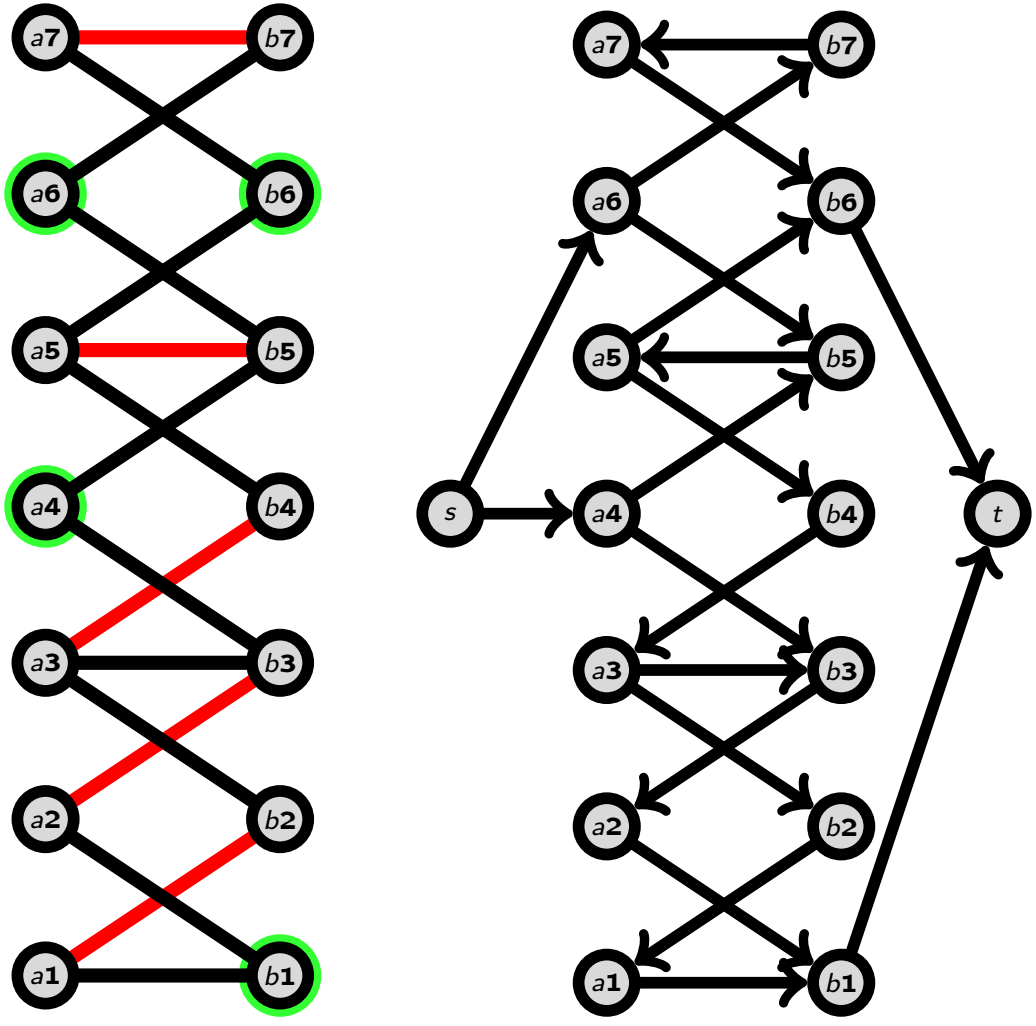
Lemma

Die Bestimmung einer inklusions-maximalen Menge von kürzesten verbessernden Pfaden geht in Zeit $O(m)$

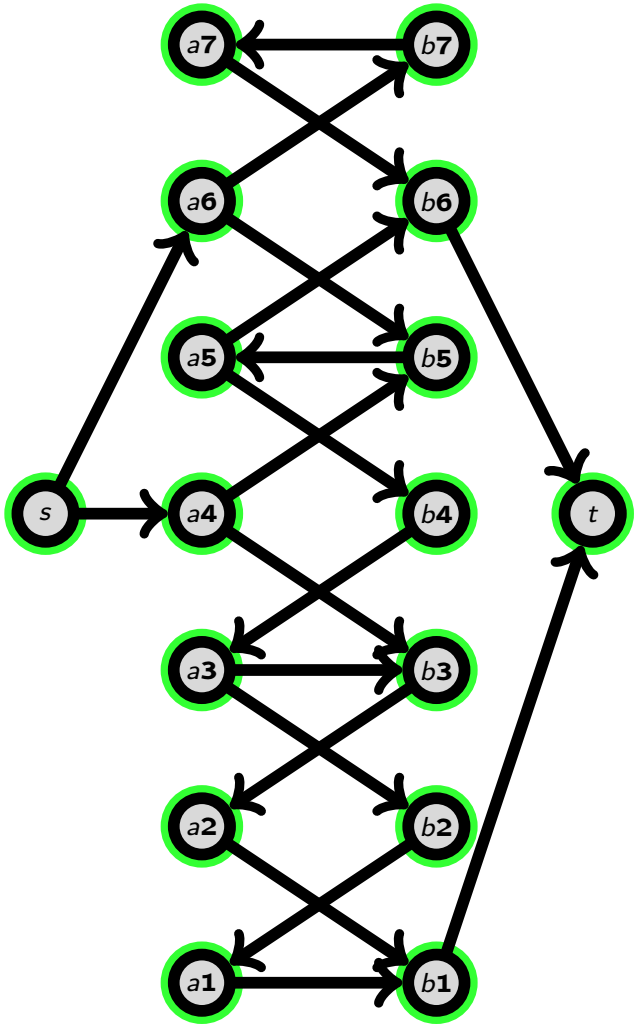
Beweis (Idee wird auch beim gewichteten Matching benutzt.)

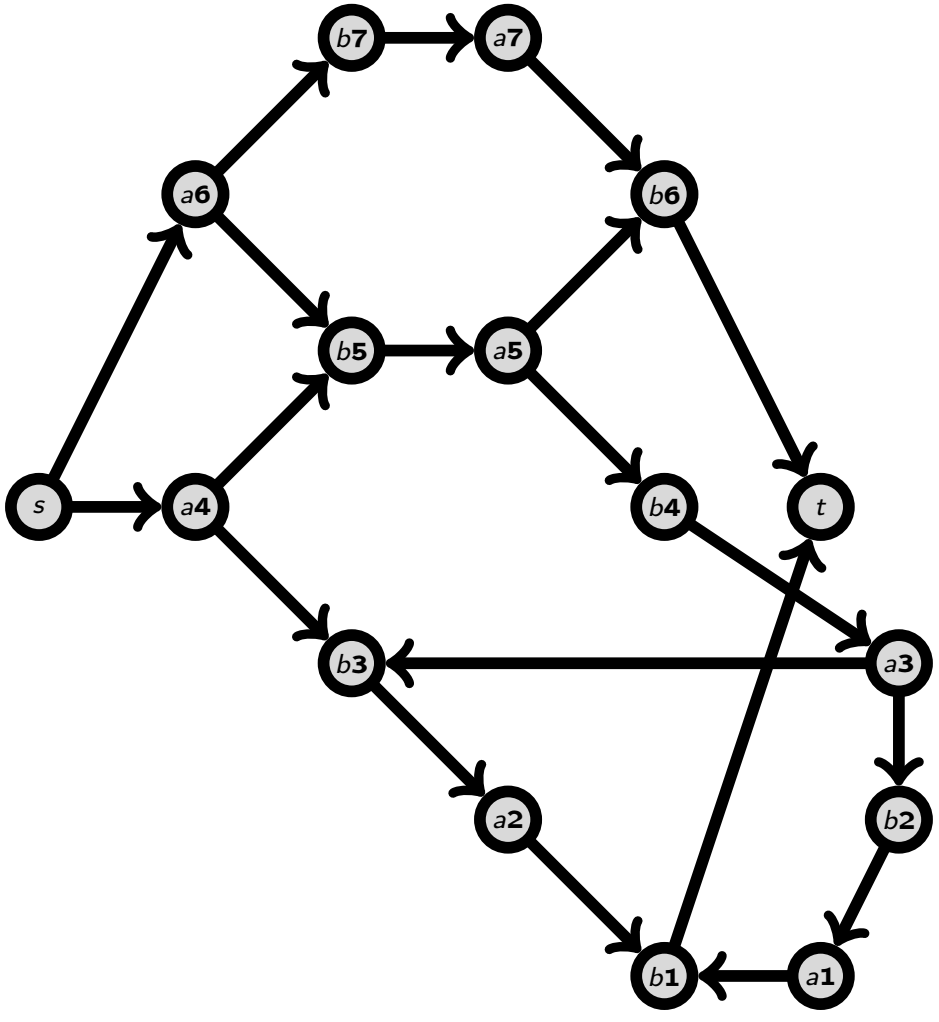
- Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$.
- Damit wird die Suche nach einem verbessernden Pfad eine Suche in G': Von den freien Knoten in V zu den freien Knoten in W.
- Füge Quelle und Senke hinzu, damit entsteht G''.
- Bestimme durch Breitensuche alle Kanten, die nicht auf einem kürzesten Pfad liegen.
- Damit entsteht ein Schichtennetzwerk G'''.
- Damit ergibt sich eine Laufzeit von $O(m)$ durch Tiefensuche auf G'''.

Beispiel (Bestimme G')

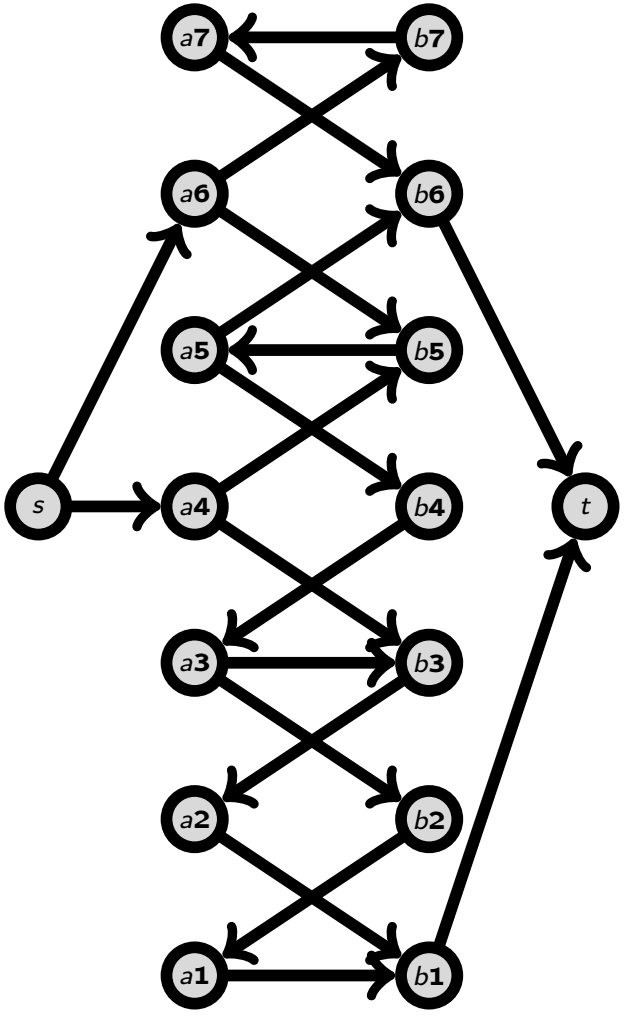


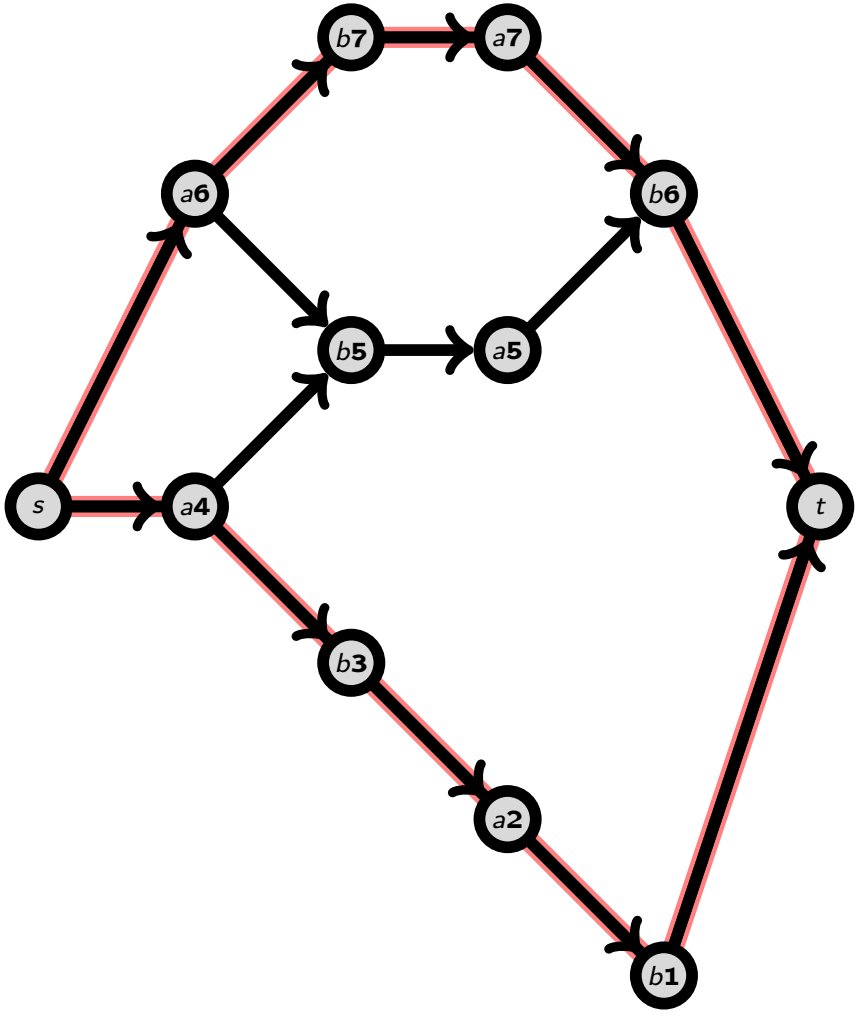
Beispiel (Breitensuche)



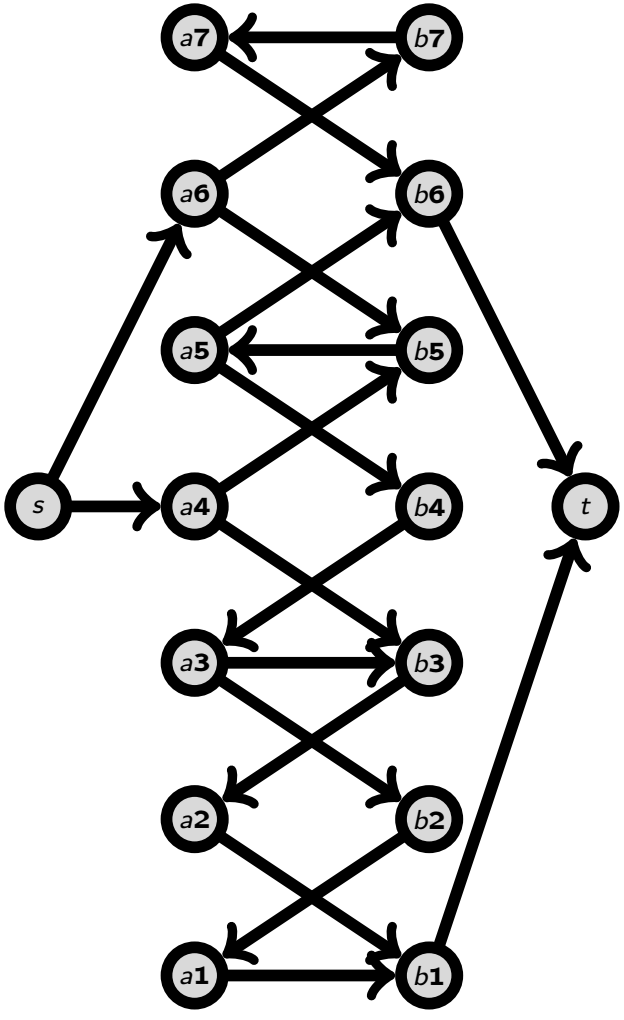


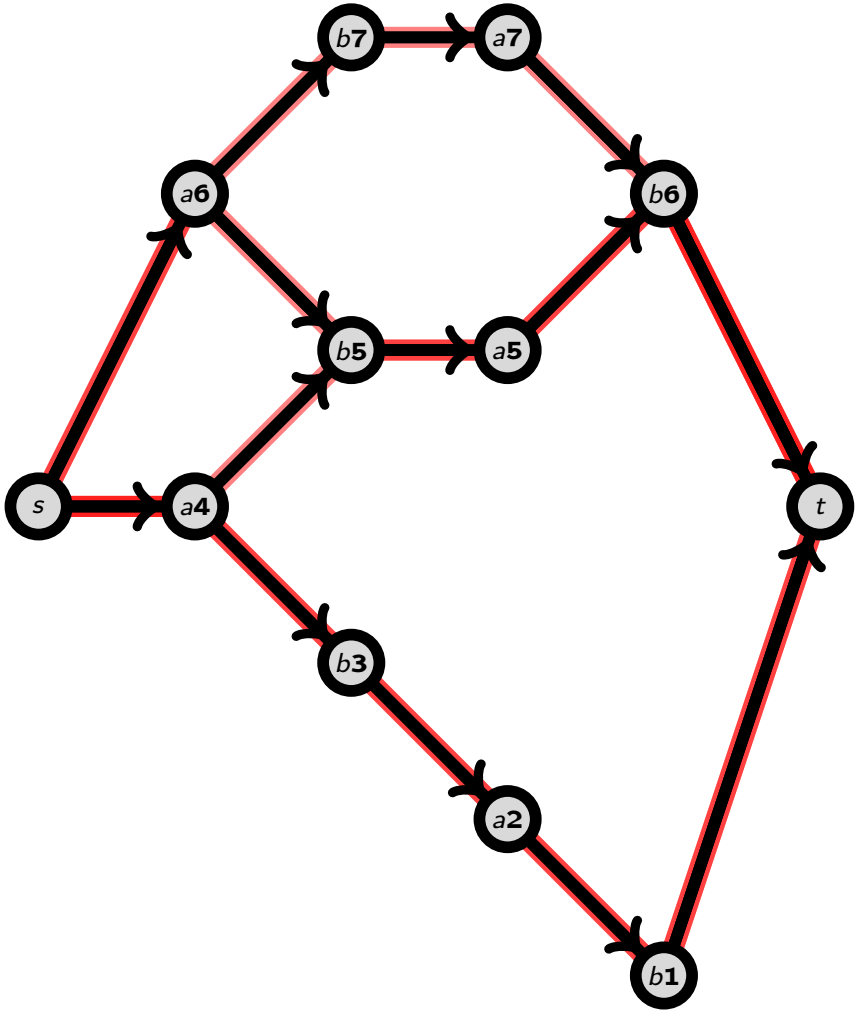
Beispiel (Tiefensuche)



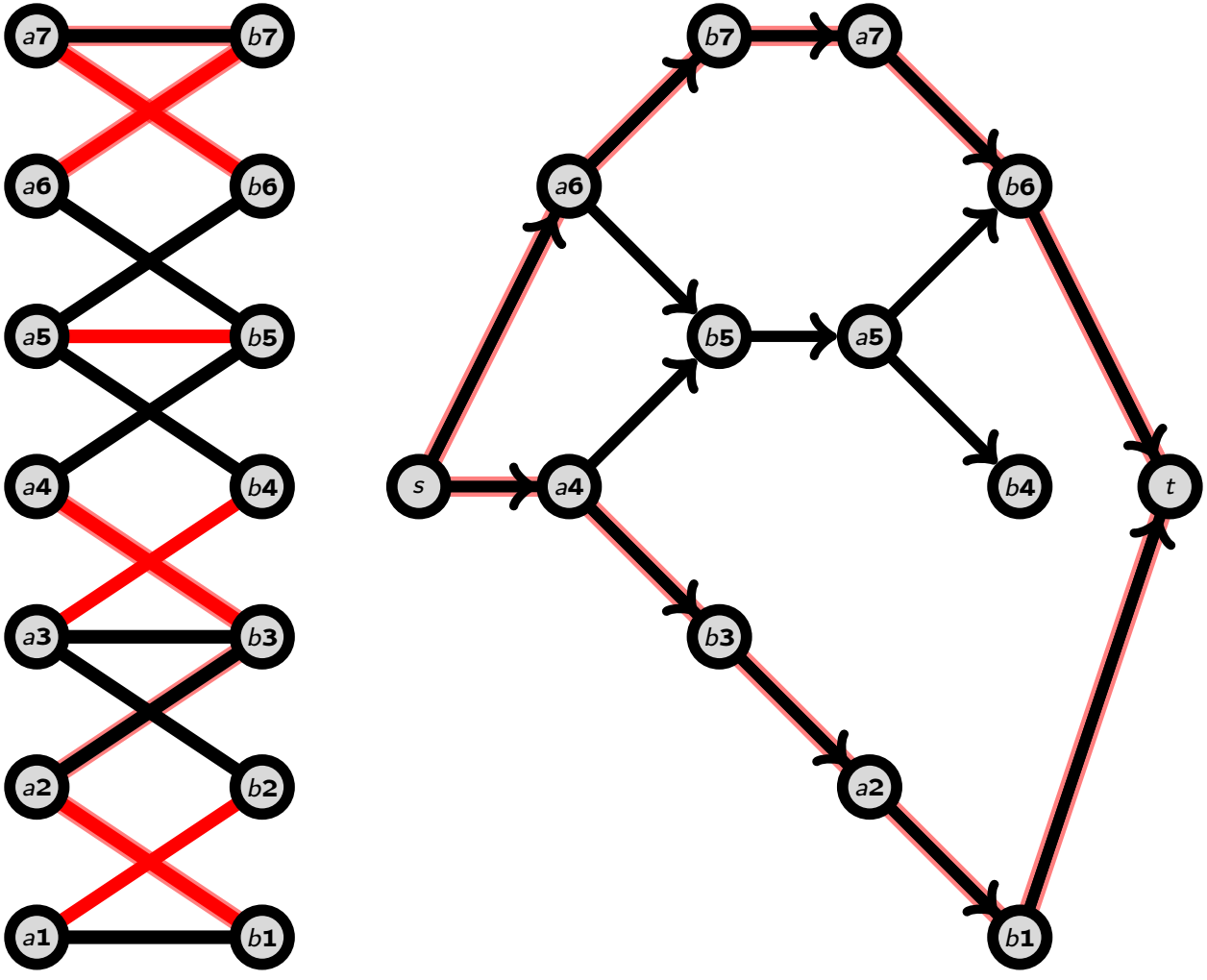


Beispiel (Verbessernde Pfade)





Beispiel

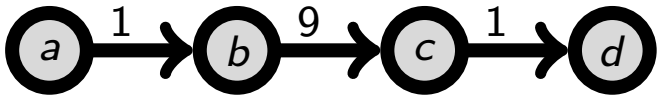

a7
b7
a6
b6
a5
b5
a4
b4
a3
b3
a2
b2
a1
b1
b7
a7
a6
b6
b5
a5
s
a4
b4
t
b3
a2
b1

Laufzeit

Theorem

Das Maximum Matchingproblem auf bipartiten Graphen kann in Zeit $O(m\sqrt{n})$ gelöst werden.

Idee zum Matching mit Kosten



Man kann schnell an dem folgenden Beispiel sehen, daß die Güte eines Matchings durch die geforderte Anzahl der Kanten schwanken kann. In dem Beispiel hat das Matching aus zwei Kanten einen Wert von 2, aber eine einzelne Kante schafft 9. Daher muss zur Bestimmung eines gewichtsmaximales Matching, Matchings jeder Größe betrachtet werden.

Das Verfahren kann nicht effizient werden, wenn zu viele Matchings einer Größe betrachtet werden müssen. Das ist aber nicht notwendig. Sei M ein gewichtsmaximales Matching der Größe x und M' ein beliebiges Matching der Größe $x+1$. Wie schon vorher gemacht, gibt es also einen alternierenden verbessernden Pfad, um aus M ein M' zu bestimmen. Dabei bestimmen die Gewichte des alternierenden verbessernden Pfades, wie sich die Gewichte von M und M' unterschieden.

Wenn unser Algorithmus den alternierenden verbessernden Pfad mit der besten Gewichtsverbesserung bestimmt, so wird dieser ein gewichtsmaximales Matching M' der Größe $x+1$ bestimmen. Damit ist das Vorgehen im Folgenden auch klar.

Matching mit Kosten

$g(E) = \sum_{e \in E} g(e)$

Definition (maximum Matching)

Sei $G = (V, E)$ ungerichteter bipatiter Graph mit Kantenkosten $g : E \mapsto \mathbb{N}$.
$M \subseteq E$ ist gewichtsmaximales Matching:

$$\forall M' \subset E : M' \text{ ist Matching. } : g(M') \leqslant g(M)$$

- Idee suche erweiternden alternierenden Weg mit maximalen Kosten.
- Verwende Idee von der schnellen Wegsuche
- Hier werden nun Wege maximalen Gewichts gesucht.
  - Sei $G = (V, W, E)$ bipartiter Graph und M Matching.
  - Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
    $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
    $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$ und
    $\forall e \in E' : g_M(e) = g(e)$ und
    $\forall e \in E'' : g_M(e) = -g(e)$.
  - Füge Quelle und Senke hinzu, damit entsteht G'.

Idee

$g(E) = \sum_{e \in E} g(e)$

- Sei $G = (V, W, E)$ bipartiter Graph, $g : E \mapsto \mathbb{N}$ und M Matching.
- Erzeuge $G' = (V \cup W, E' \cup E'')$ mit:
  $E' = \{(v, w) \mid \{v, w\} \in E \setminus M \wedge v \in V \wedge w \in W\}$ und
  $E'' = \{(w, v) \mid \{v, w\} \in M \wedge v \in V \wedge w \in W\}$ und
  $\forall e \in E' : g_M(e) = g(e)$ und $\forall e \in E'' : g_M(e) = -g(e)$.
- Damit wird ein ungerichteter M-verbessender Pfad P bestimmt.
- Gewicht von P: $g_M(P) = \sum_{e \in P \setminus M} g(e) - \sum_{e \in P \cap M} g(e)$.
- Damit haben wir:

$$g(M \oplus P) = \sum_{e \in M} g(e) + \sum_{e \in P \setminus M} g(e) - \sum_{e \in P \cap M} g(e) = g(M) + g_M(P)$$

- Damit ergibt sich der folgende Algorithmus:

Algorithmus

$g(E) = \sum_{e \in E} g(e)$

1. Gegeben $G = (V, W, E)$, $g :\mapsto \mathbb{N}$
2. $M = \emptyset$ und $M_{opt} = \emptyset$
3. Solange es verbessernden Pfad P bezüglich M gibt, mache:
   1. $\forall e \in E' : g_M(e) = g(e)$ und $\forall e \in E'' : g_M(e) = -g(e)$
   2. Bestimme verbessernden Pfad P mit maximalen Gewicht bezüglich M und $g_M(e)$:
   3. Setze $M = M \oplus E(P)$
   4. Falls $g(M) > g(M_{opt})$, setze $M_{opt} = M$.
4. Ausgabe: M_{opt}.

Theorem

Der obige Algorithmus bestimmt gewichtsmaximales Matching und hat eine Laufzeit von $O(n^2 \cdot m)$.

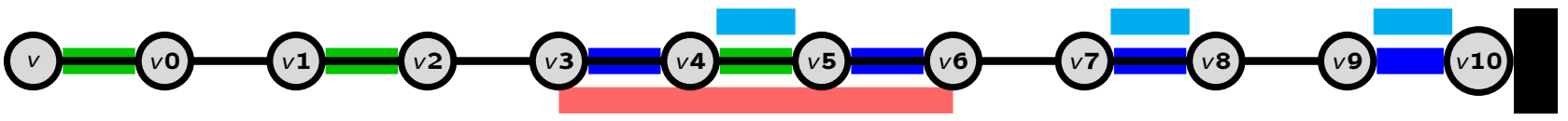
Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i + 1$ folgt.
- Seien M_i die Machtings, die im i-ten Schritt bestimmt wurden.



- Sei M' ein Matching mit $|M'| = i + 1$ und maximalem Gewicht.
- Sei P ein verbessernde Pfad in $M_i \oplus M'$.
- Betrachte $M = M' \oplus P$, es gilt $|M| = i$.
- Es gilt damit: $P \cap M = P \setminus M' = P \cap M_i$.

Beweis

Lemma

Das im i-ten Schritt bestimmte Matching M_i ist das gewichtsmaximale Matching mit Kardinalität i.

Beweis per Induktion:

- Induktionsanfang $i = 0$ ist klar ($M_0 = \emptyset$), Induktionsschritt $i \to i+1$ folgt.
- Seien M_i die Machtings, die im i-ten Schritt bestimmt wurden.
- Sei M' ein Matching mit $|M'| = i+1$ und maximalem Gewicht.
- Sei P ein verbessernde Pfad in $M_i \oplus M'$.
- Betrachte $M = M' \oplus P$, es gilt $|M| = i$.
- Es gilt damit: $P \cap M = P \setminus M' = P \cap M_i$.
- Per Induktion gilt: $g(M_i) \geqslant g(M) = g(M') - g_M(P)$.
- Und wir erhalten $g_M(P) = g_{M_i}(P) \leqslant g_{M_i}(P')$,
  wobei P' der Pfad ist, den der Algorithmus bestimmte.
- Damit: $g(M') = g(M) + g_M(P) \leqslant g(M_i) + g_{M_i}(P') = g(M_{i+1})$.

Laufzeit

- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen verbessernden kostenmaximalen Weg.
- Wir setzen $l(e) = -g_M(e)$ für alle Kanten, damit haben wir:
- Der Algorithmus bestimmt $\lfloor n/2 \rfloor$ mal einen kürzesten Weg.
- Dabei treten auch negative Gewichte auf.
- Wegen des vorherigen Lemmas haben wir keine negativen Kreise bezüglich l.
  - Positive Kreise bezüglich g_M widersprächen der Optimalität von M_i:
- Damit kann nun mit dem Bellmann-Ford-Algorithmus von einer Quelle aus gearbeitet werden.
- Dieser hat Laufzeit $O(nm)$.
- Damit ist die Gesammtlaufzeit $O(n^2 m)$.

Nochmal eine Potentialfunktion

Hier nutzen wir ein Vorgehen, daß schon bei Flüssen mit Kosten auftrat. Eine Potentialfunktion bestimmt das Potential eines Knotens. D.h. wie "gut" ist der Knoten für die Lösung des Problems ist.

Zur Bestimmung der Potentialfunktion wird ein Knoten q gewählt. Für einen Knoten v wird als Potential gewählt: die Kosten eines kürzesten Weges von q nach v. Ein Knoten v addiert sein Potential auf die Gewichte jeder seiner ausgehenden Kanten und subtrahiert die Gewichte jeder seiner eingehenden Kanten. Hier ergeben sich zwei Effekte:

- Das Gewicht eines Weges wird nur um die Potentiale der Endknoten verändert.
- Die Gewichte der Kanten werden nun nicht negativ.

Damit ist der Algorithmus von Dijkstra zur Laufzeitverbesserung anwendbar, denn es können die ursprünglichen kürzesten Wege bestimmt werden.

Idee

$l(e) = -g_M(e)$

- Nutze den Dijkstra-Algorithmus (Laufzeit $O(m + n \log n)$).
- Transformiere dazu die Längen, so dass keine negativen Werte auftreten.
- Beachte: wir haben keine negativen Kreise.
- Verwende dazu eine Potentialfunkion $p : V \mapsto \mathbb{Z}$ mit:

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v)$$

- Die Berechnung so einer Potentialfunktion p im Rahmen der Suche nach dem Matching wird die Ungarische Methode genannt.

Transformation 1

$l(e) = -g_M(e)$

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $p : V \mapsto \mathbb{Z}$. Wenn G keine negativen Kreise enthält, dann setze:

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt für jeden kürzesten Weg P von a nach b:

$$\text{dist}_{l'}(a, b) = p(a) - p(b) + \text{dist}_l(a, b).$$

Beweis:

$$\begin{aligned} \text{dist}_{l'}(a, b) &= \sum_{e \in P} l'(e) \\ &= \sum_{(v,w)=e \in P} l(e) - p(w) + p(v) \\ &= p(a) - p(b) + \sum_{e \in P} l(e) \end{aligned}$$

Damit können die kürzesten Wege auch mit diesen neuen Kantengewichten bestimmt werden.

Transformation 2

$l(e) = -g_M(e)$

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt:

$$\forall e \in E : l'(e) \geqslant 0.$$

Beweis:

- Für jede Kante $e = (v, w)$ gilt: $\text{dist}_l(q, w) \leqslant \text{dist}_l(q, v) + l(e)$.
- Und damit:

$$\begin{aligned} l'(e) &= l(e) - p(w) + p(v) \\ &= l(e) - \text{dist}_l(q, w) + \text{dist}_l(q, v) \\ &\geqslant l(e) - (\text{dist}_l(q, v) + l(e)) + \text{dist}_l(q, v) = 0 \end{aligned}$$

Transformation 3

$l(e) = -g_M(e)$

Lemma

Sei $G = (V, E)$ Graph mit Kantengewichten $l : E \mapsto \mathbb{Z}$ und $q \in V$. Wenn G keine negativen Kreise enthält, dann setze weiter $p : V \mapsto \mathbb{Z}$ mit $p(v) = \text{dist}_l(q, v)$ und

$$\forall (v, w) = e \in E : l'(e) = l(e) - p(w) + p(v).$$

Dann gilt für die Kanten e eines kürzsten Weg von q aus:

$$l'(e) = 0.$$

Beweis:

- Wegen $l'(e) = l(e) - p(w) + p(v)$ gilt für ein Kante $e = (q, v)$: $l'(e) = 0$.
- Sei P ein kürzester Weg von q nach v.
- Dieser hat im Vergleich zu den ursprünglichen Kosten den aditiven Term $p(q) - p(v)$.
- Damit $p(q) - p(v) = \text{dist}_l(q) - \text{dist}_l(v) = -\text{dist}_l(v)$.
- Und weiter. $\text{dist}_{l'}(v) = \text{dist}_l(v) - \text{dist}_l(v) = 0$.

Ungarische Methode

$l(e) = -g_M(e)$

1. Gegeben $G = (V, W, E)$, $g : E \mapsto \mathbb{N}$
2. $M = \emptyset$, $M_{opt} = \emptyset$ und für alle $v \in V$ setze $p(v) = 0$.
3. Für alle $w \in W$ setze $p(w) = \min_{e \in E} l(e)$.
4. Wiederhole die folgenden Schritte:
   1. Füge Knoten q zu V hinzu und Kanten (q, v) zu allen Knoten aus $v \in V \setminus V(M)$.
   2. Bestimme Kantenlängen l_M mit Hilfe von g und M.
   3. Bestimme $p(v) = \text{dist}_{l_M}(q, v)$ für alle erreichbaren Knoten v.
   4. Setze $l'(e) = l_M(e) + p(v) - p(w)$ für alle Kanten $e = (v, w)$.
   5. Falls es Pfad P mit $l'(P) = 0$ von q zu einem Knoten aus $W \setminus V(M)$ gibt, so setze $M = M \oplus E(P)$
   6. Falls es keinen solchen Pfad P gibt, so gebe M_{opt} aus und terminiere.
   7. Falls $g(M) > g(M_{opt})$, setze $M_{opt} = M$.

Laufzeit

$l(e) = -g_M(e)$

Theorem

Der obige Algorithmus bestimmt gewichtsmaximales Matching und hat eine Laufzeit von $O(n \cdot (m + n \log n))$.

Bemerkungen dazu (Nach Konstruktion gelten die vorherigen Ausagen):

- Beim Übergang von M_i nach M_{i+1} werden nur Kanten e mit Kosten $l'(e) = 0$ gedreht.
- Damit vergrößert sich nicht der von q aus erreichbare Teil.
- Sei $e = (v, w)$ so eine Kante vor der Drehung.
- Für die gedreht Kante $e' = (w, v)$ gilt:

$$l'_{M_{i+1}}(e') = l_{M_{i+1}}(e') - p(w) + p(v) = -l_{M_i}(e') - p(w) + p(v) = -l'_{M_i}(e) = 0.$$

- Damit können wir im Schritt 5.4 den Dijkstra-Algorithmus anwenden.
- Weiterhin können wir die Potentiale in Schritt 5.2 mit Breitesuche bestimmen.
- Damit $O(n)$ Iterationen mit Laufzeit $O(m + n \log n)$ plus $O(n + m)$.

Alle bisherigen Verfahren waren für bipatite Graphen. Der Beweis der Korrektheit, d.h. das fehlende verbessernde alternierende Pfade ein Kriterium für ein maximum Matching sind, war auch für allgemeine Graphen gültig. Daher stellt sich die Frage, ob man auch auf allgemeinen Graphen effizient verbessernde alternierende Pfade finden kann. Dazu muss das Durchlaufen der ungeraden Kreise bei der Suche nach alternierenden Pfade betrachtet werden:

- Wird bei der Suche ein gematcher Knoten über eine Kanten, die nicht im Matching ist, betreten, so ist die Folgekante eindeutig bestimmt.
- Enthält ein ungerader Kreis der Länge l weniger wie $\lfloor l/2 \rfloor$ Matchingkanten, so ist ein Teil des Kreis nicht erreichbar.
- Wird ein ungerader Kreis bei der Suche über eine Kante betreten, die nicht im Matching ist, so ist die Durchlaufrichtung eindeutig.
- Enthält ein ungerader Kreis der Länge l genau $\lfloor l/2 \rfloor$ Matchingkanten und wird bei der Suche über eine Matchingkante betreten, so gibt es zwei Möglichkeiten den Kreis zu durchlaufen.

Nur der letzte Fall ist kritisch. In diesem Fall kann aber der Kreis über jeden Knoten verlassen werden. Daher kann der Kreis durch einen einzelnen Knoten ersetzt werden und damit die Suche vereinfacht werden. Es ergibt sich damit eine effiziente Methode zur Suche nach verbessernden alternierenden Pfaden.

Einleitung

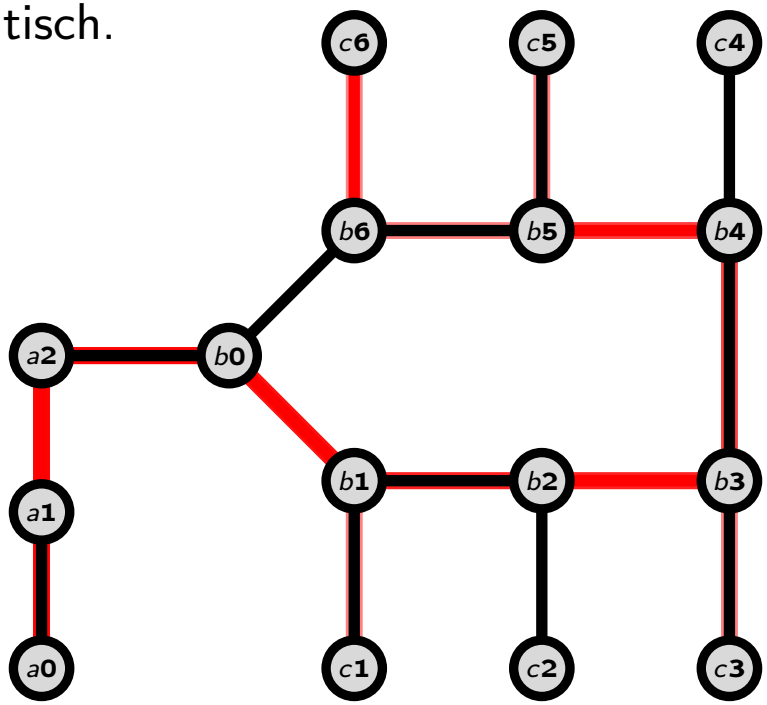
$l(e) = -g_M(e)$

- Was passiert, wenn der obige Algorithmus auf ungerade Kreise stößt?
- Es gibt ggf. mehr Möglichkeiten des Durchlaufs.
- Ausblick und Vorgehen:
  - Untersuche die möglichen Situationen.
  - Erkenne die Situationen, die kritisch für den Algorithmus sind.
  - Passe Algorithmus für diese Situationen an.

Erster unkritischer Fall

$l(e) = -g_M(e)$

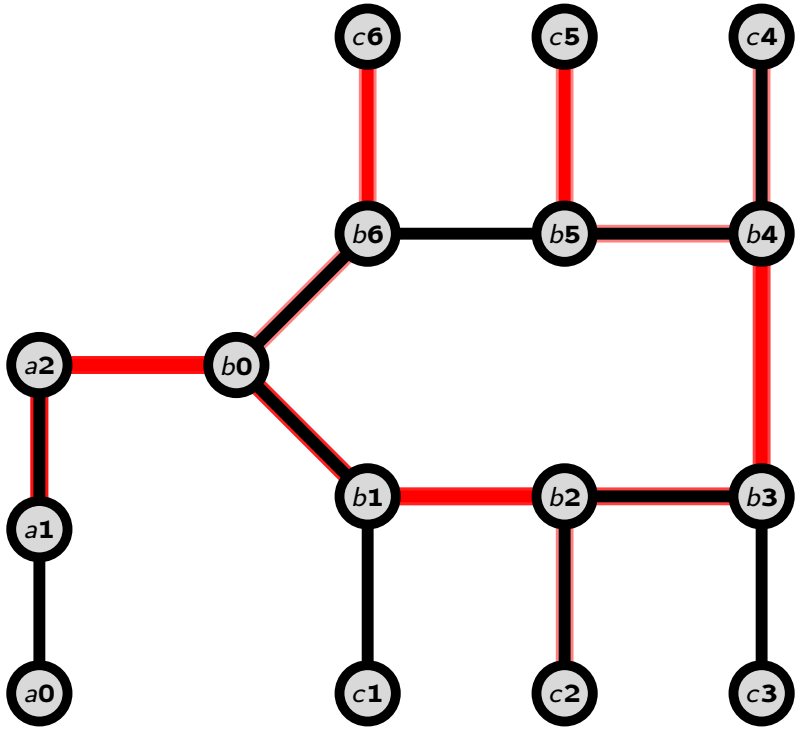
- Die Suche startet an a_0.
- Der Kreis $C = \{b_0, b_1, \ldots, b_6\}$ wird bei b_0 erreicht.
- Wegen $\{b_0, b_1\} \in M$ wird der Kreis in eindeutiger Richtung durchlaufen.
- Damit ist die Kante $\{b_0, b_6\}$ nicht mehr relevant.
- Die Suche sieht keinen Kreis.
- Der Fall ist unkritisch.



Zweiter unkritischer Fall

$l(e) = -g_M(e)$

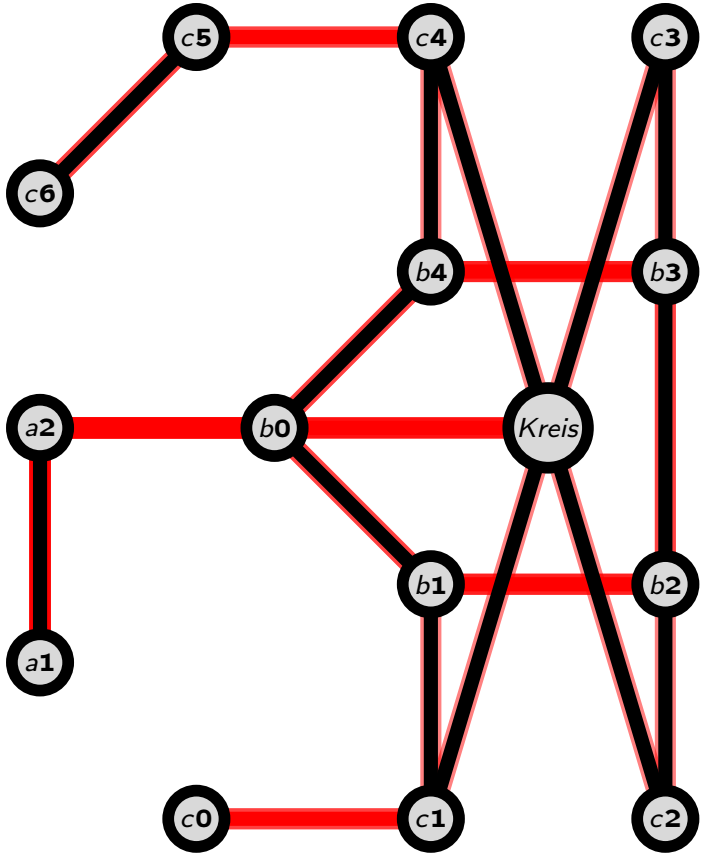
- Die Suche startet an a_1.
- Der Kreis $C = \{b_0, b_1, \ldots, b_6\}$ wird bei b_0 erreicht.
- Es gibt zwei Möglichkeiten, in den Kreis zu laufen.
- Diese sind unabhängig, da b_5 frei in $G|C$ ($\{b_5, b_6\}$ nicht relevant)
- Der Fall ist unkritisch.



Beispiel

$l(e) = -g_M(e)$

- Problem mit ungeraden Kreisen.
- Wenn wir von a_1 starten, dann gibt es zwei Möglichkeiten.
  - über b_0 und b_1
    - nach c_2
    - nach c_6
  - oder über b_0 und b_4
    - nach c_3
    - nach c_0
- Jeder Knoten aus dem Kreis ist erreichbar.
- Die ausgehende Kante bestimmt den Durchlauf durch den Kreis.
- Daher kann der Kreis durch einen Knoten ersetzt werden.



Situation

$l(e) = -g_M(e)$

Definition (Blüte)

Sei $G = (V, E)$ ungerichteter Graph und M Matching in G.
$C = \{v_0, v_1, v_2, \ldots, v_k\}$ heißt Blüte, falls

- $G|C$ ist ein ungerader Kreis (d.h. k ist gerade).
- $\{v_i, v_{i+1}\} \in M$ für $i \in \{1, 3, 5, \ldots, k-1\}$.

v_0 ist der Startknoten der Blüte C.
Definiere $S(G, C) = (V', E' \cup E'')$ mit:

- $V' = (V \setminus C) \dot\cup \{c\}$
- $E' = \{\{v, w\} \mid \{v, w\} \in E \wedge v, w \in V'\}$
- $E'' = \{\{v, c\} \mid \exists w \in C : \{v, w\} \in E\}$

Aufbau der Idee:

$l(e) = -g_M(e)$

- Starte mit einer normalen Suche nach verbessernden Pfaden.
- Falls eine Blüte C in ihrem Startknoten betreten wird, so mache:
  - Schrumpfe C zu einem Knoten,
  - d.h. setze Suche auf $S(G, C)$ fort.
  - Falls verbessernder Pfad gefunden wird, so übertrage diesen auf G.

Initialisiere:

1. Eingabe: $G = (V, E)$ und M Matching.
2. Setze: $G' = (V, E')$ mit $E' = \{(v, w) \mid \{v, w\} \in E\}$.
3. Für alle $\{v, w\} \in M$ setze: $P(v) = w$ und $P(w) = v$.
4. Setze: $Z(v) =$ even für alle $v \in V$ mit $v \cap \cup_{e \in E} e \neq \emptyset$
5. Setze: $Z(v) =$ away für alle $v \in V$ mit $v \cap \cup_{e \in E} e = \emptyset$

Algorithmus (Edmonds)

$l(e) = -g_M(e)$

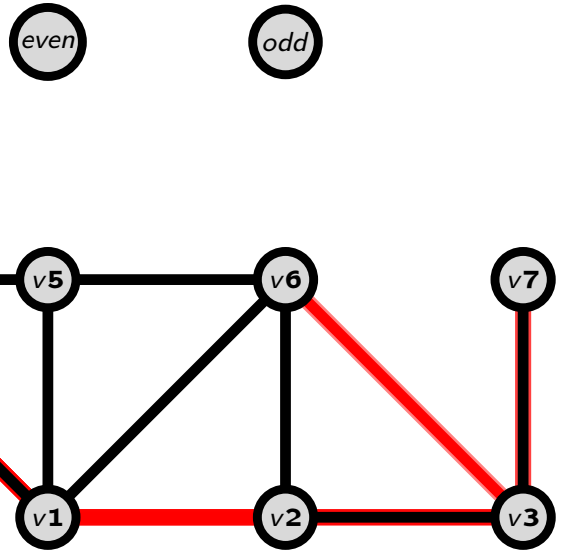
1. Wiederhole: Wähle $(v, v') = e \in E'$ mit: $Z(v) =$ even und e ist noch nicht betrachtet worden.
   1. Falls $Z(v') =$ odd, führe nichts aus (Fall 0).
   2. Falls $Z(v') =$ away, führe Fall 1 aus.
   3. Falls $Z(v') =$ even und v und v' im selben Baum, führe Fall 2 aus.
   4. Falls $Z(v') =$ even und v und v' nicht im selben Baum, führe Fall 3 aus.
2. bis Fall 3 eingetreten oder es gibt keine zu untersuchenden Kanten mehr.

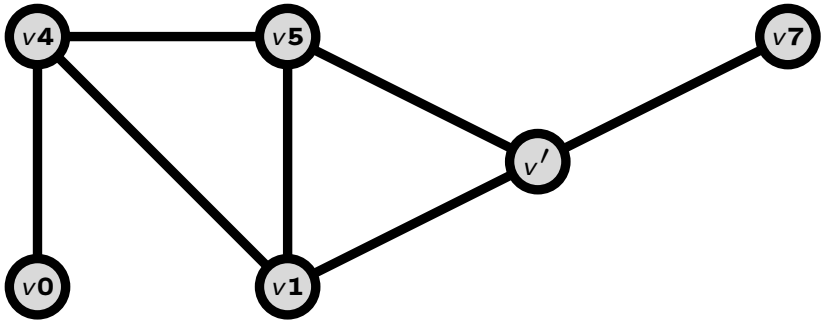
- Fall 1:
  1. $Z(v') =$ odd und $Z(P(v')) =$ even.
  2. $p(v') = v$ und $p(P(v')) = v'$.
- Fall 2:
  1. v'' sei nächster gemeinsamer Vorfahr.
  2. Schrumpfe Blüte $v, v', \ldots, v'', \ldots, v$.
  3. Passe dabei Daten p an.
- Fall 3:
  1. Verbinde v und v'.
  2. Verbessernder Pfad ist gefunden.

Beispiel

$l(e) = -g_M(e)$

- Wähle Kante v_0, v_4.
- Fall 2, Weg: v_0, v_4
- Wähle Kante v_4, v_1.
- Fall 2, Weg: v_0, v_4, v_1
- Wähle Kante v_1, v_2.
- Fall 2, Weg: v_0, v_4, v_1, v_2
- Wähle Kante v_2, v_3.
- Fall 2, Weg: v_0, v_4, v_1, v_2, v_3
- Wähle Kante v_3, v_7.
- Fall 0, v_3, v_7 ist betrachtet worden.
- Wähle Kante v_3, v_6.
- Fall 2, es wird geschrumpft.





Ergebnisse

$l(e) = -g_M(e)$

Theorem

Auf allgemeinen Graphen ist das Matchingproblem in Zeit $O(m\sqrt{n})$ lösbar.

Beweisidee:

- Kombiniere obigen Algorithmus mit der Suche nach kurzen verbessernden Pfaden.
- D.h. adaptiere obigen Algorithmus in eine Breitensuche.
- Verwalte dabei die Pfadlängen korrekt, d.h. beachte die Länge in den Blüten mit.
- Die Pfadlänge in den Blüten ist abhängig von den Kanten, die wir zum Betreten und Verlassen nutzen.

Ergebnisse

$l(e) = -g_M(e)$

Theorem

Auf allgemeinen Graphen ist das gewichtete Matchingproblem in Zeit $O(n^3)$ lösbar.

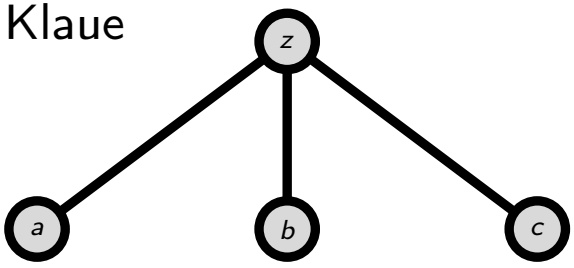
Beweisidee:

- Suche verbessernde Pfade und
- Suche kostenverbessernde alternierende Kreise.
- Vorgehen wie bei kostenminimalen Flüssen.

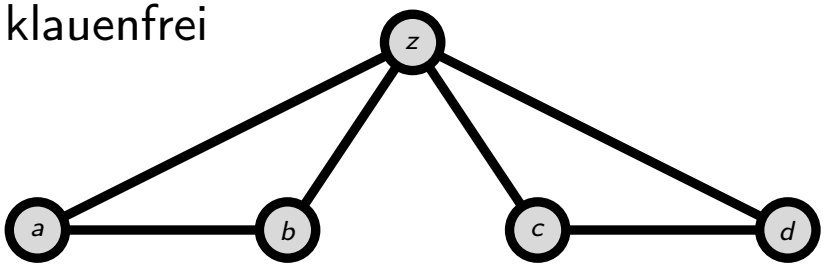
Claw-free

$N(v) = \{w \mid \{w, v\} \in E\}$, $N[v] = N(v) \cup \{v\}$

Klaue



klauenfrei



Definition (Claw-free (klauenfrei))

G heißt klauenfrei, falls kein knoteninduzierter Teilgraph eine Klaue ist.

- Ziel: Bestimme die größte stabile Menge (independent set) auf klauenfreien Graphen.
- Dazu betrachten wir eine Teilklasse der klauenfreien Graphen.
- Auf Kantengraphen entspricht das Bestimmen einer stabilen Menge einem Matching.
- Dann übertragen wir das Vorgehen “verbessernde Pfade” auf klauenfreie Graphen.

Kantengraphen

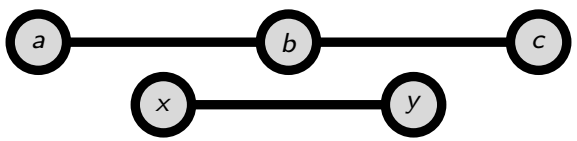
$N(v) = \{w \mid \{w, v\} \in E\}$, $N[v] = N(v) \cup \{v\}$

Definition (Kantengraph)

Sei $G = (V, E)$ ungerichteter Graph. $L(G) = (E, E')$ heißt Kantengraph von G, falls

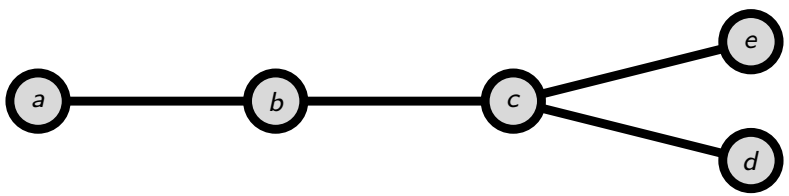
$$E' = \{\{e, e'\} \mid e, e' \in E \wedge e \cap e' \neq \emptyset\}.$$

Ein Graph H heißt Kantengraph, falls es einen Graphen G gibt, mit $L(G) = H$.

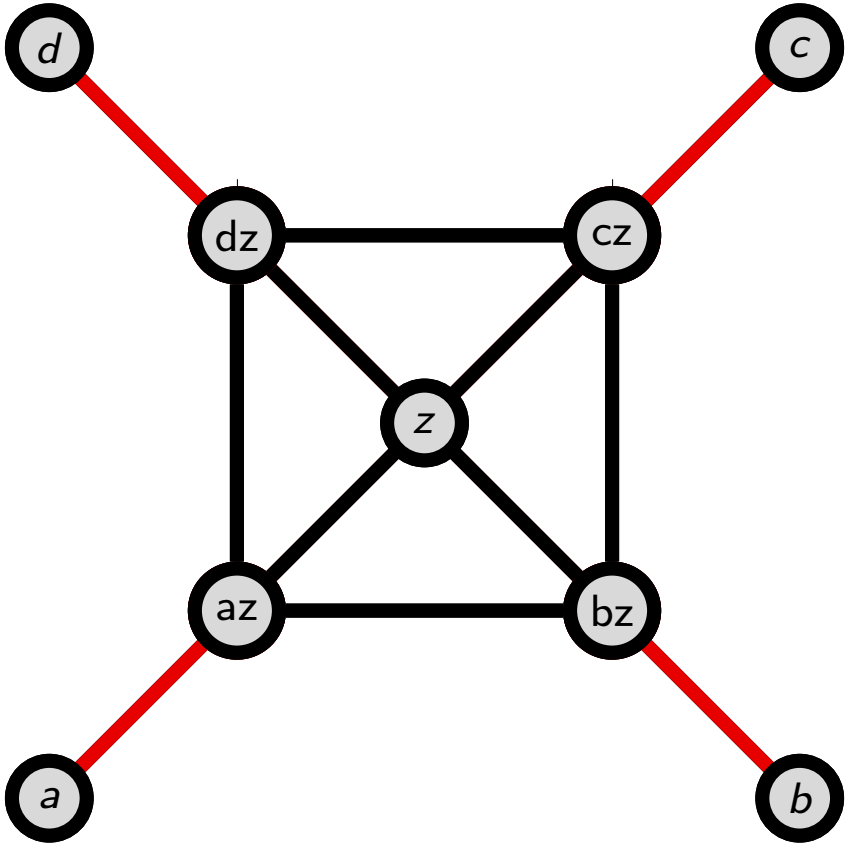


Lemma

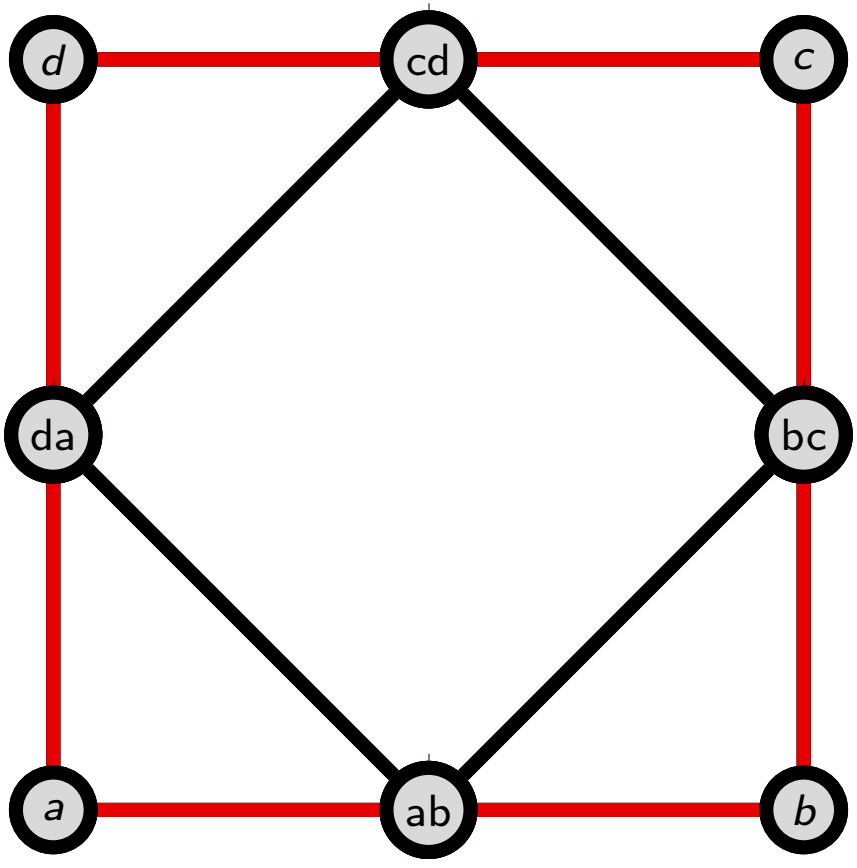
Kantengraphen sind klauenfrei.



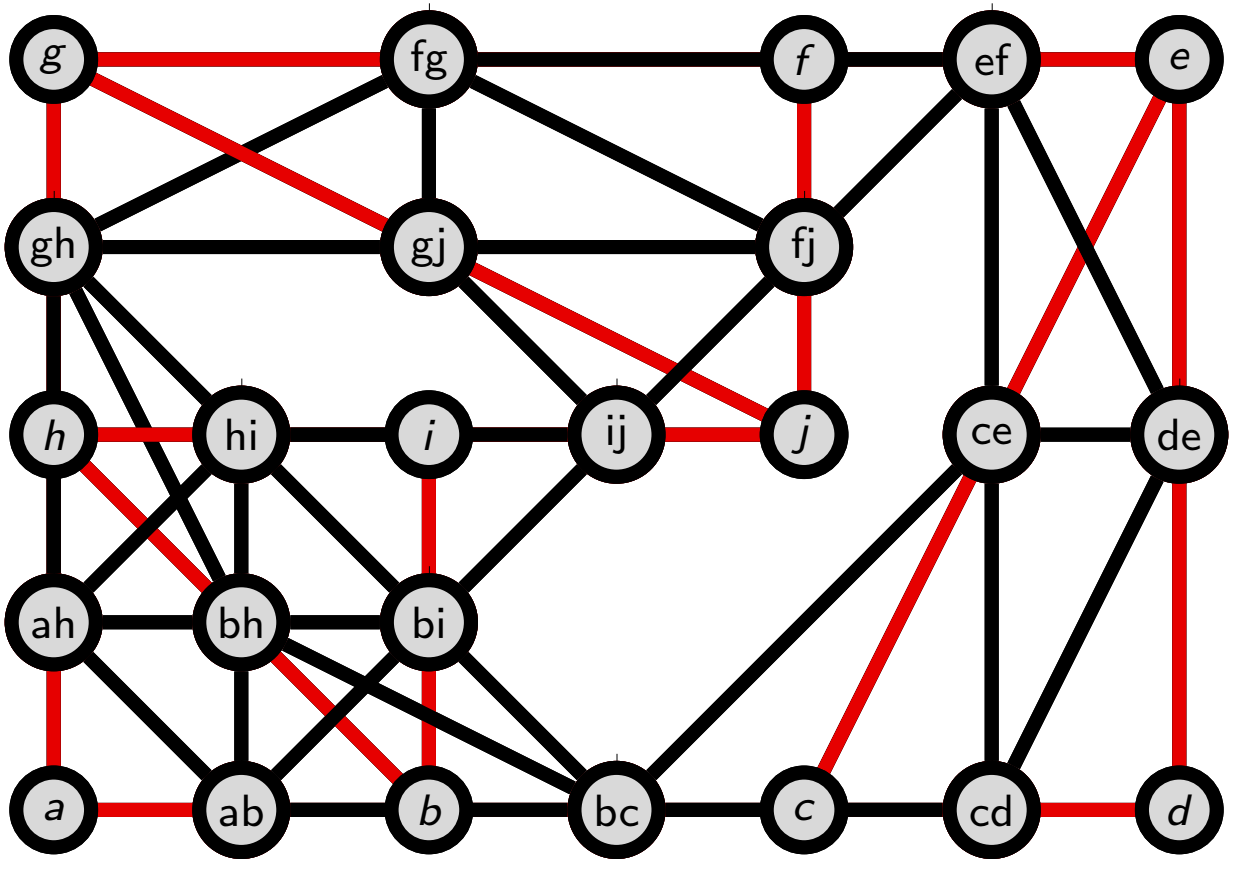
Beispiel 1



Beispiel 2



Beispiel 3

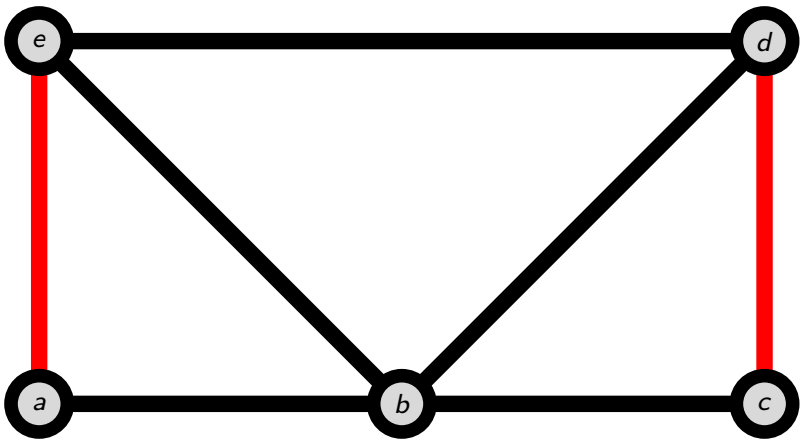


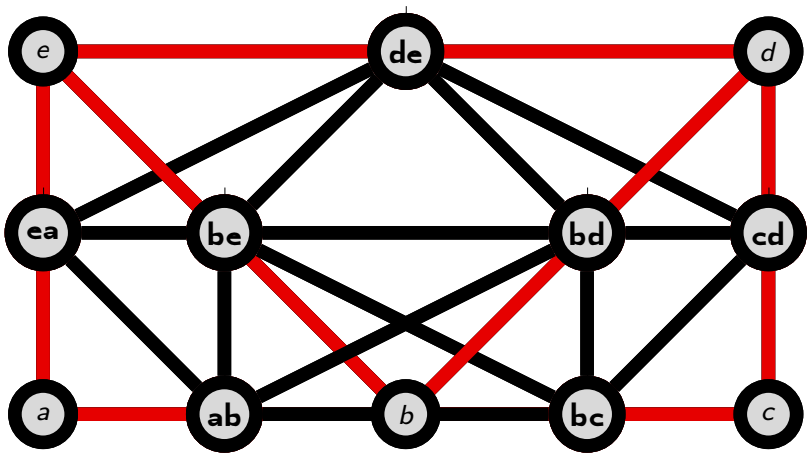
Aussagen

Lemma

Für Kantengraphen G kann, falls H mit $L(H) = G$ bekannt ist, eine größte stabile Menge in Zeit $O(m\sqrt{n})$ bestimmt werden.
(Das Erkennungsproblem für Kantengraphen ist in $\mathcal{P}$.)

Beweis: Übung.





Aussagen

$G|W = (W, \{e \mid e \in E \wedge e \setminus W = \emptyset\})$

Definition (Independent Set (Stabile Menge))

Sei $G = (V, E)$. Die Größe der größten stabilen Menge wird mit $\alpha(G)$ bezeichnet:

$$\alpha(G) = \max_{I \subset V \wedge E(G|I)=\emptyset} |I|$$

Lemma (Claw-free (klauenfrei))

$G = (V, E)$ ist klauenfrei, falls $\forall v \in V : \alpha(G|N[V]) \leqslant 2$.

Beweis: Übung:

Theorem

Für klauenfreie Graphen G ist das Bestimmen von $\alpha(G)$ in $\mathcal{P}$.

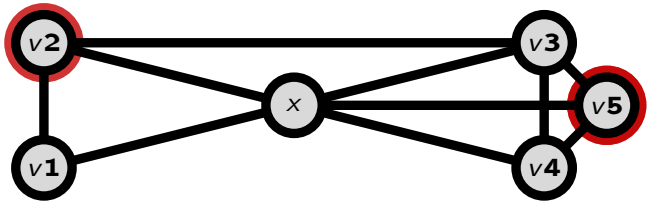
Beschreibung des Vorgehens

Hier betrachten wir nun klauenfreie Graphen. Bei einem klauenfreien Graphen bilden die Nachbarn eines Knotens maximal zwei Cliquen. Daher gilt für einen Knoten, der nicht in einem Independent Set ist: Es sind höchsten zwei Nachbarn in einem Independent Set. Betrachten wir eine Independent Set A und ein größeres Independent Set B. Nun untersuchen wir den durch $A \cup B$ induzierten Graphen. Analog wie beim Matching, muss es hier nun einen speziellen erweiternden Pfad geben. Diesen können wir analog wie beim Matching suchen.

Man beachte dabei, wenn wir eine Suche nach einem Independent Set erweiternden Weg von einem Knoten starten, der nicht in dem bisherigen Independent Set ist, so ist diese Wegsuche eindeutig bestimmt. Daher können wir analog wie beim Matching das Independent Set Problem auf klauenfreien Graphen lösen.

Idee

- Gegeben sei $G = (V, E)$ und S stabile Menge auf G.
- Vergrößere schrittweise S durch eine "alternierende Menge (Pfad)"
- Die Knoten aus S nennen wir "rote Knoten", alle anderen sind "weiß".
- Erste Klassifizierung der weißen Knoten



gebunden: benachbart mit zwei roten Knoten.
beweglich: benachbart mit einem roten Knoten.
frei: benachbart mit keinem roten Knoten.

- Knoten die frei sind, können wir sofort in die stabile Menge aufnehmen.
- D.h. im Folgenden keine freien Knoten.
- Nun definieren wir das Analogon zu den alternierenden Pfaden.

Alternierende Mengen

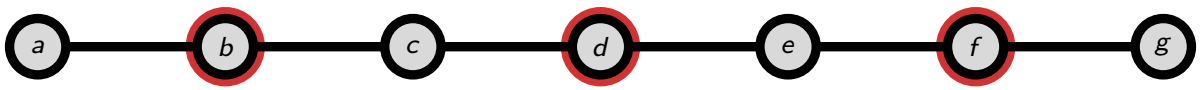
Definition

Gegeben sei $G = (V, E)$ klauenfrei und $S \subset V$ stabile Menge. Eine Knotenmenge $P \subset V$ heißt alternierende Menge, falls die weißen Knoten von P eine stabile Menge in $G|P$ sind.

Lemma

Eine alternierende Menge P auf einem klauenfreien Graphen ist ein Pfad oder ein Kreis. D.h. $\delta(G|V(P)) \leqslant 2$.

Beweis: $V(P) \cap S$ und $V(P) \cap (V \setminus S)$ sind stabile Mengen. Mit $\forall v \in V : \alpha(G|N[V]) \leqslant 2$ folgt die Behauptung.



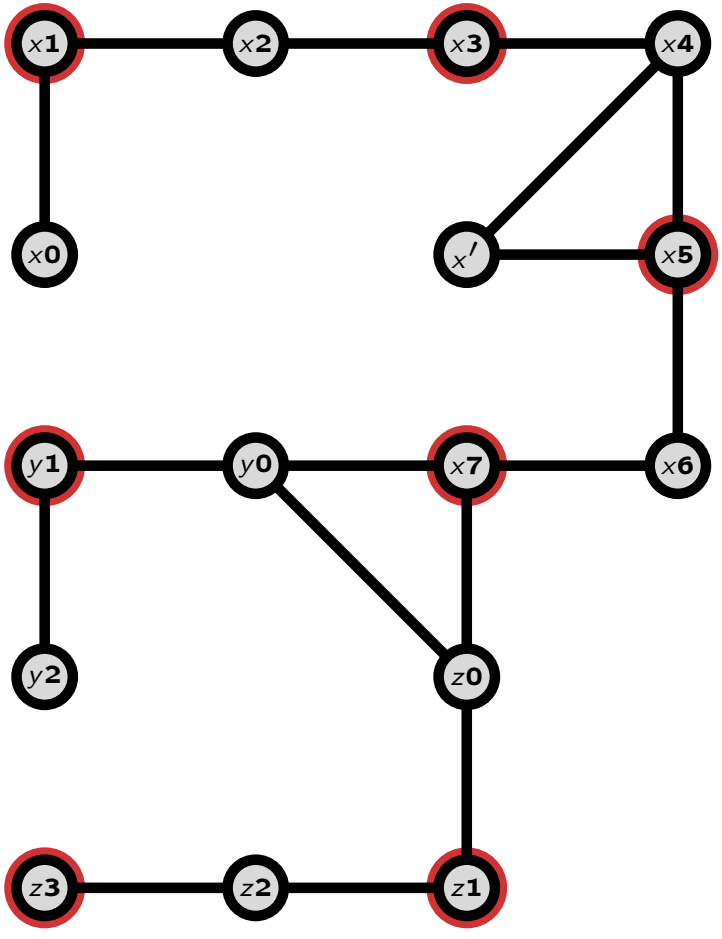
Verbessernde Pfade

Definition (Verbessernder Pfad)

Falls eine alternierende Menge ein Pfad P ist, mit:

- Die Endpunkte sind weiss.
- Die Endpunkte sind nicht verbunden.
- Es gibt keine Kanten zwischen weissen Knoten auf dem Pfad.
  - Hier reicht es aus zu fordern: Keine Kanten zwischen weissen Knoten, die einen Abstand von zwei haben auf P.

Einige hilfreiche Strukturen



- Sei x_0 beweglicher Knoten.
- Dann ist x_1 eindeutig bestimmt.
- Das kann ggf. eindeutig weiter gehen.
- Aufteilung geht nur an einem roten Knoten (x_5).
- Dann gibt es Kante in $N(x_5)$.
- Dann geht der Weg eindeutig weiter: x_6, x_7.
- Andere Aufteilung an x_7.
- Jede Alternative ist möglich.
- Algorithmisch ergibt sich aber eine einfache Verzweigungsstruktur.

Algorithmus

- Eine vollständige Untersuchung ergibt, dass das Suchen nach den verbessernden Pfaden möglich ist.
- Wird hier nicht vorgeführt.
- Ist Graphentheorie.
- Am Ende ergibt sich aber ein ähnlicher Algorithmus:

1. Gegeben $G = (V, E)$ klauenfreier Graph
2. $S = \emptyset$
3. Solange es verbessernden Pfad P gibt, mache:
   1. Setze $S = S \oplus V(P)$
4. Ausgabe: S.

Definition

- Gegeben vollständiger bipartiter Graph (A, B, E) mit $|A| = |B|$.
- Eine Sympathieliste $L(A)$ ist eine totale Ordnung auf A: $a_1 < a_2 < \ldots a_n$.
- Für jedes $a \in A$ gibt es Sympathieliste $L_a(B)$
- Für jedes $b \in B$ gibt es Sympathieliste $L_b(A)$
- Ziel: Bestimme stabile Paarung (perfektes Matching), mit:
  - Kein Paar a, b ist unzufrieden, d.h.
  - $(a, b) \notin M$ und $(a, b') \in M$, $(a', b) \in M$ mit:
  - $b <_{L_a(B)} b'$, d.h. a bevorzugt b vor b' und
  - $a <_{L_b(A)} a'$, d.h. b bevorzugt a vor a'

Beispiel

$$
\begin{array}{llll}
L_{a_1}(B) = (b_2, b_5, b_1, b_3, b_4) & & L_{b_1}(A) = (a_5, a_1, a_4, a_2, a_3) \\
L_{a_2}(B) = (b_1, b_2, b_3, b_4, b_5) & & L_{b_2}(A) = (a_4, a_5, a_2, a_1, a_3) \\
L_{a_3}(B) = (b_2, b_3, b_5, b_4, b_1) & & L_{b_3}(A) = (a_1, a_4, a_2, a_3, a_5) \\
L_{a_4}(B) = (b_1, b_3, b_2, b_4, b_5) & & L_{b_4}(A) = (a_3, a_2, a_4, a_1, a_5) \\
L_{a_5}(B) = (b_5, b_3, b_2, b_1, b_4) & & L_{b_5}(A) = (a_4, a_2, a_3, a_5, a_1)
\end{array}
$$

- Paarung A: $(a_1, b_1)(a_2, b_3)(a_3, b_2)(a_4, b_4)(a_5, b_5)$
- Paarung B: $(a_1, b_1)(a_2, b_4)(a_3, b_5)(a_4, b_3)(a_5, b_2)$
- Paarung A ist nicht stabil: (a_1, b_2) sind unzufrieden.
- Paarung B ist stabil.

Idee des Verfahrens

Hier betrachten wir eine abgeschwächte Version eines Matchingproblems. Die Unterschiede sind:

- Es sind alle Kanten in dem Graphen vorhanden, d.h. alle Paarungen sind möglich.
- Es gibt für jeden Knoten eine Präferenzliste der möglichen Zuordnungen.
- Es wird nicht nach dem besten Matching gesucht, sondern nach einem Matching, wo keine lokale Verbesserung möglich ist.

Hier ergibt sich ein einfacher Greedy Algorithmus. Die noch nicht gematchten Knoten einer Partition fragt einfach anhand der lokalen Präferenzliste nach einem Matchingpartner. Bei einer möglichen Verbesserung des angefragten Knotens wird eine neue Matchingkante gebildet (und ggf. eine schlechtere Matchingkante entfernt). Am Ende entsteht ein Matching, bei dem keine lokale Verbesserung möglich ist.

Algorithmus (Gale und Shapley)

1. Setze $M = \emptyset$
2. Solange es noch einen $a_i \in A$ gibt, mit $a_i \notin \cup_{e \in M} e$, mache
   1. Sei b_j erstes Element in $L_{a_i}(B)$.
   2. Lösche b_j aus $L_{a_i}(B)$.
   3. Falls $b_j \notin \cup_{e \in M} e$ gilt, so setze: $M = M \cup \{a_i, b_j\}$.
   4. Falls $b_j \in \cup_{e \in M} e$ und sei $\{a_k, b_j\} \in M$ dann mache:
      1. Falls $a_i <_{L_{b_j}(A)} a_k$, dann setze: $M = (M \setminus \{\{a_k, b_j\}\}) \cup \{a_i, b_j\}$.

Theorem (Gale und Shapley)

Der Algorithmus terminert nach $O(n^2)$ Runden mit einer stabilen Paarung.

Beispiel

$$
\begin{array}{lcllcl}
L_{a_1}(B) & = & (b_2, b_5, b_1, b_3, b_4) & L_{b_1}(A) & = & (a_5, a_1, a_4, a_2, a_3) \\
L_{a_2}(B) & = & (b_1, b_2, b_3, b_4, b_5) & L_{b_2}(A) & = & (a_4, a_5, a_2, a_1, a_3) \\
L_{a_3}(B) & = & (b_2, b_3, b_5, b_4, b_1) & L_{b_3}(A) & = & (a_1, a_4, a_2, a_3, a_5) \\
L_{a_4}(B) & = & (b_1, b_3, b_2, b_4, b_5) & L_{b_4}(A) & = & (a_3, a_2, a_4, a_1, a_5) \\
L_{a_5}(B) & = & (b_5, b_3, b_2, b_1, b_4) & L_{b_5}(A) & = & (a_4, a_2, a_3, a_5, a_1)
\end{array}
$$

Ablauf des Verfahrens:

- $M_x(y)$: in Runde x wird Matching mit y aufgenommen.
- $F_x(y)$: in Runde x wird Matching mit y angefragt.
- $S_x(y)$: in Runde x wird Matching mit y gelöst.

$$
\begin{array}{lcl}
\text{Partner}(a_1) & = & M_2(b_2), S_7(b_2), M_8(b_5), S_9(b_5), M_{10}(b_1), b_3, b_4 \\
\text{Partner}(a_2) & = & M_3(b_1), S_6(b_1), M_7(b_2), S_{14}(b_2), F_{15}(b_3), M_{16}(b_4), b_5 \\
\text{Partner}(a_3) & = & F_4(b_2), M_5(b_3), S_{11}(b_3), M_{12}(b_5), b_4, b_1 \\
\text{Partner}(a_4) & = & M_6(b_1), S_{10}(b_1), M_{11}(b_3), b_2, b_4, b_5 \\
\text{Partner}(a_5) & = & M_9(b_5), S_{12}(b_5), F_{13}(b_3), M_{14}(b_2), b_1, b_4
\end{array}
$$

Beweis

- Laufzeit:
  - Für jedes Paar $a, b \in A \times B$ wird höchstens eine Anfrage gemacht.
- Korrektheit (Matching ist perfekt):
  - Jedes Element aus B bleibt in $\cup_{e \in M} e$, wenn es einmal aufgenommen wurde.
  - Falls am Ende ein Paar $a, b \notin \cup_{e \in M} e$, dann hat a nie eine Anfrage an b gemacht, Widerspruch.
- Korrektheit (Matching ist stabil):
  - Angenommen (a, b) sind unzufrieden: $(a, b) \notin M$ und $(a, b') \in M$, $(a', b) \in M$ mit: $b <_{L_a(B)} b'$ und $a <_{L_b(A)} a'$.
  - Falls a nie bei b angefragt hat (aber wohl bei b'), gilt $b' <_{L_a(B)} b$ (Widerspruch).
  - Falls a vor a' bei b angefragt hat, gilt $a =_{L_b(B)} a'$ (Widerspruch).
  - Falls a nach a' bei b angefragt hat, hätte a a' verdrängt (Widerspruch).

Literatur

- Cormen, Leiserson, Rives: Introduction to Algorithms, First Edition, MIT Press, 1990.
- Cormen, Leiserson, Rives: Introduction to Algorithms, Second Edition, MIT Press, 2001.
- Ottmann, Widmayer: Algorithmen und Datenstrukturen. BI-Wiss.-Verl. 1990.

Fragen(Matching)

- Wie kann man das bipartite Matching mit Flussalgorithmen lösen?
- Welche Laufzeiten haben die verschiedenen Matchingprobleme?
- Welcher Zusammenhang besteht zwischen verbessernden Pfaden und einem maximum Matching?
- Wie ist die Vorgehensweise, um eine Laufzeit von $O(m\sqrt{n})$ für das Matchingproblem zu erhalten?
- Wie ist die Vorgehensweise, um das Matching auf allgemeinen Graphen zu bestimmen?

Legende

n : Nicht relevant

g : Grundlagen, die implizit genutzt werden

i : Idee des Beweises oder des Vorgehens

s : Struktur des Beweises oder des Vorgehens

w : Vollständiges Wissen