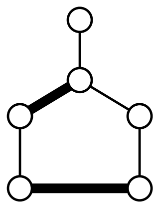
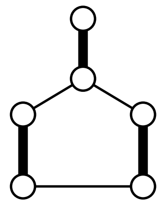
Matchings in Graphen

Es sei ein ungerichteter Graph $G = (V, E)$ gegeben. Ein *Matching* in G ist eine Teilmenge $M \subseteq E$, so dass keine zwei Kanten aus M einen Endpunkt gemeinsam haben. Ein Matching M heißt *maximal*, falls es in G kein größeres Matching M' mit $M' \supset M$ gibt. Ein Matching M heißt *Matching maximaler Kardinalität* (engl. *maximum matching*), falls es in G kein Matching M' mit $|M'| > |M|$ gibt. Wir sind an einem Algorithmus interessiert, der für einen gegebenen Graphen effizient ein Matching maximaler Kardinalität berechnet.



(a) maximales Matching



(b) Matching max. Kardinalität

Matching-Probleme treten in der Praxis meistens als Zuordnungsprobleme auf. Ein Beispiel könnte etwa die Zuordnung von Professoren zu Vorlesungen sein, wenn jeder Professor höchstens eine Vorlesung halten will: die Knoten des Eingabegraphen repräsentieren die Professoren und die Vorlesungen, eine Kante zwischen einem Professor und einer Vorlesung gibt an, dass der Professor die Vorlesung halten kann. Ein Matching maximaler Kardinalität entspricht dann genau einer Zuordnung von Professoren zu Vorlesungen, bei der so viele Vorlesungen angeboten werden können wie möglich.

Für einen Graphen $G = (V, E)$ und ein Matching $M \subseteq E$ nennen wir die Kanten aus M *gematcht* und die Kanten aus $E \setminus M$ *ungematcht*. Ein Knoten heißt *gematcht*, wenn eine seiner inzidenten Kanten gematcht ist. Knoten, die keine inzidente gematchte Kante haben, heißen *frei*. Hilfreich ist weiterhin das Konzept alternierender bzw. augmentierender Pfade bzgl. eines Matchings M:

(i) Ein einfacher Pfad $v_0, v_1, \ldots, v_r$ heißt *alternierend*, falls die Kanten (v_{i-1}, v_i) abwechselnd in M und in $E \setminus M$ sind.

(ii) Ein alternierender Pfad heißt *augmentierend*, falls er an beiden Enden ungematchte Kanten hat und nicht verlängert werden kann, also wenn beide Endknoten des Pfades frei sind.

Beachte, dass ein augmentierender Pfad auch aus einer einzigen ungematchten Kante zwischen zwei freien Knoten bestehen kann.

Man sieht leicht, dass ein Matching M mit Hilfe eines augmentierenden Pfades p zu einem Matching M' vergrößert werden kann: wenn man die gematchten Kanten des Pfades aus M herausnimmt und die ungematchten Kanten des Pfades in M einfügt, erhält man nämlich ein gültiges Matching mit einer zusätzlichen Kante. Diesen Prozess bezeichnen wir auch als *Invertieren* eines augmentierenden Pfades. Das obige Matching (b) lässt sich zum Beispiel aus Matching (a) durch ein derartiges Invertieren eines augmentierenden Pfades erhalten.

Satz 1 *Ein Matching M in $G = (V, E)$ hat genau dann maximale Kardinalität, wenn es keinen augmentierenden Pfad gibt.*

Beweis: Die Behauptung des Satzes ist äquivalent zu: Ein Matching M hat genau dann nicht maximale Kardinalität, wenn es einen augmentierenden Pfad gibt. Wir beweisen diese Behauptung. Die Richtung $\Leftarrow$ ist offensichtlich richtig. Die Richtung $\Rightarrow$ ist noch zu zeigen.

Sei M ein Matching in G, das nicht maximale Kardinalität hat. Sei M' ein Matching in G mit maximaler Kardinalität. Betrachte den Graphen $G' = (V, M \oplus M')$, wobei $M \oplus M' = (M \cup M') \setminus (M \cap M')$ die symmetrische Differenz von M und M' ist. Offensichtlich haben in G' alle Knoten Grad ≤ 2. Also besteht G' aus einer Reihe von einfachen Pfaden und Kreisen, die alle alternierend bzgl. M sind. M' lässt sich aus M durch Invertieren aller dieser Pfade und Kreise erhalten. Da das Invertieren von Kreisen und von nicht-augmentierenden Pfaden das Matching nicht vergrößert, müssen unter den Pfaden auch $|M'| - |M| \geq 1$ augmentierende Pfade sein. $\square$

Aus dem Beweis des Satzes folgt sogar, dass es in G bzgl. M genau $|M'| - |M|$ knotendisjunkte augmentierende Pfade gibt, wobei M' ein Matching maximaler Kardinalität ist. Da sich die $|M|$ gematchten Kanten auf diese $|M'| - |M|$ augmentierenden Pfade verteilen, lässt sich auch ableiten, dass es einen augmentierenden Pfad der Länge höchstens $2 \cdot \lfloor |M|/(|M'| - |M|) \rfloor + 1$ gibt.

Aufgrund des Satzes wissen wir auch, dass wir ein Matching maximaler Kardinalität finden können, indem wir mit einem beliebigen Matching M, zum Beispiel $M = \emptyset$, starten und so lange augmentierende Pfade suchen und diese invertieren, bis es keinen augmentierenden Pfad mehr gibt. Es ist nur noch zu klären, wie die Suche nach augmentierenden Pfaden realisiert werden soll.

Dabei stellt es sich zunächst als sehr günstig heraus, nicht beliebige augmentierende Pfade zu suchen, sondern eine maximale Menge *kürzester* augmentierender Pfade. Sei M das aktuelle Matching und sei ℓ die Länge (Anzahl Kanten) eines kürzesten augmentierenden Pfades bzgl. M. Dann suchen wir eine Menge $p_1, p_2, \ldots, p_r$ von knotendisjunkten augmentierenden Pfaden der Länge ℓ, so dass es keinen weiteren solchen knotendisjunkten Pfad mehr gibt, und invertieren alle diese Pfade $p_1, \ldots, p_r$. Dieses Vorgehen hat den Vorteil, dass wir höchstens $O(\sqrt{|V|})$-mal augmentierende Pfade suchen müssen, wie Hopcroft und Karp gezeigt haben. Lässt sich die Suche nach augmentierenden Pfaden in Zeit $O(|V| + |E|) = O(|E|)$ realisieren, erhält man also einen Matching-Algorithmus mit Gesamtlaufzeit $O(\sqrt{|V|}|E|)$. Im nächsten Abschnitt wird dies für den Fall bipartiter Graphen konkret beschrieben.

1 Matchings in bipartiten Graphen

Ein Graph $G = (V, E)$ heißt *bipartit*, wenn sich seine Knotenmenge V so in disjunkte Teilmengen V_1 und V_2 aufteilen lässt, dass alle Kanten einen Knoten aus V_1 mit einem aus V_2 verbinden. Bei vielen in der Praxis auftretenden Zuordnungsproblemen ist der entstehende Graph tatsächlich bipartit, so auch beim eingangs erwähnten Zuordnungsproblem mit Professoren und Vorlesungen.

In bipartiten Graphen lässt sich die Suche nach einer maximalen Menge knotendisjunkter augmentierender Pfade durch *simultane Breitensuche* und anschließende *Tiefensuche* effizient in linearer Zeit $O(|V| + |E|)$ realisieren (Algorithmus von Hopcroft und Karp, 1973). Genauer sieht der Algorithmus so aus:

- setze $M := \emptyset$

- **while** (true) **do**
  simultane Breitensuche;
  if (es existiert ein augmentierender Pfad)
  then Tiefensuche nach knotendisjunkten kürzesten augmentierenden Pfaden;
  else break;
  fi
  od

- gib M als Matching maximaler Kardinalität aus

Eine Ausführung des Rumpfes der while-Schleife nennen wir *Iteration*. Wir wissen, dass der Algorithmus auch im Worst-Case nur $O(\sqrt{|V|})$ Iterationen ausführt. Jede Iteration besteht aus einer simultanen Breitensuche und, falls das Matching noch nicht maximale Kardinalität hat, einer anschließenden Tiefensuche. Sowohl die simultane Breitensuche als auch die Tiefensuche können in Zeit $O(|V| + |E|)$ ausgeführt werden, so dass sich eine Gesamtlaufzeit von $O(\sqrt{|V|}|E|)$ ergibt.

1.1 Simultane Breitensuche

Bei gegebenem aktuellen Matching M ist es die Aufgabe der simultanen Breitensuche, erstens zu entscheiden, ob es in dem Graphen überhaupt noch augmentierende Pfade gibt, und zweitens den Knoten des Graphen Level-Werte zuzuordnen, mit deren Hilfe die anschließende Tiefensuche eine maximale Menge kürzester knotendisjunkter augmentierender Pfade finden kann.

Jeder Pfad in einem bipartiten Graphen G besucht abwechselnd einen Knoten in V_1 und einen in V_2. Da alle augmentierenden Pfade ungerade Länge haben, beginnen sie jeweils bei einem freien Knoten in V_1 und enden bei einem freien Knoten in V_2. Dabei verwenden sie in Richtung von V_1 nach V_2 jeweils eine ungematchte Kante und in Richtung von V_2 nach V_1 eine gematchte Kante. Wir ordnen jedem Knoten v einen Wert $level[v]$ zu, den wir am Beginn jeder simultanen Breitensuche mit -1 initialisieren und der am Ende der Breitensuche die Länge eines kürzesten alternierenden Pfades von einem freien Knoten aus V_1 zu v repräsentieren soll.

Um die Länge eines kürzesten augmentierenden Pfades zu bestimmen, lässt man eine Breitensuche gleichzeitig bei allen freien Knoten aus V_1 beginnen: aus diesem Grund spricht man von *simultaner* Breitensuche. Dabei initialisiert man die Queue einfach am Anfang nicht mit einem einzigen Startknoten, sondern fügt alle freien Knoten aus V_1 in die Queue ein. Für jeden dieser Knoten v setzt man außerdem $level[v] = 0$.

Die weitere Breitensuche läuft dann in *Phasen* ab, die abwechselnd vom Typ 1 bzw. vom Typ 2 sind:

Typ 1: zu Anfang der Phase bilden die Knoten aus der Queue eine Teilmenge von V_1; in der Phase wird jeder solche Knoten v aus der Queue geholt und alle **ungematchten** Nachbarkanten $e = \{v, w\}$ betrachtet: falls w noch nicht besucht wurde, setzt man $level[w] = level[v] + 1$ und fügt w hinten an die Queue an; am Ende der Phase besteht die Queue nur aus Knoten, die in V_2 sind.

Typ 2: zu Anfang der Phase bilden die Knoten aus der Queue eine Teilmenge von V_2; in der Phase wird jeder solche Knoten v aus der Queue geholt und seine **gematchte**

Nachbarkante $e = \{v, w\}$ betrachtet: falls w noch nicht besucht wurde, setzt man $level[w] = level[v] + 1$ und fügt w hinten an die Queue an; am Ende der Phase besteht die Queue aus Knoten, die in V_1 sind.

Die Breitensuche terminiert, wenn die Queue nach einer Phase vom Typ 1 einen freien Knoten w enthält (dann gibt es einen augmentierenden Pfad der Länge $level[w]$, der bei w endet, und es wird die anschließende Tiefensuche gestartet) oder wenn die Queue leer wird (dann gibt es keinen augmentierenden Pfad und das aktuelle Matching hat bereits maximale Kardinalität).

1.2 Tiefensuche nach augmentierenden Pfaden

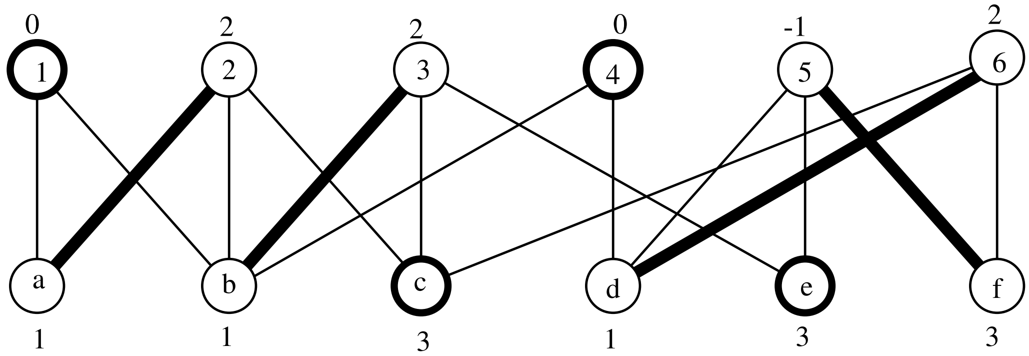
Wenn bei der simultanen Breitensuche nach ℓ Phasen ein freier Knoten aus V_2 erreicht wurde, so wissen wir jetzt, dass es mindestens einen kürzesten augmentierenden Pfad der Länge ℓ gibt, und wollen nun mittels Tiefensuche eine maximale Menge solcher kürzester augmentierender Pfade berechnen. Genauer werden wir dazu nicht eine einzige Tiefensuche verwenden, sondern für jeden freien Knoten v aus V_1 eine eigene Tiefensuche starten, um von v aus (falls möglich) einen augmentierenden Pfad der Länge ℓ zu finden. Bei diesen Tiefensuchen betrachten wir ausschließlich Kanten, die den folgenden Bedingungen genügen (andere Kanten werden ignoriert):

- Ist der aktuelle Knoten $u \in V_1$, so betrachten wir von u aus Kanten $e = \{u, w\}$ mit $e \in E \setminus M$ und $level[w] = level[u] + 1$.
- Ist der aktuelle Knoten $u \in V_2$, so betrachten wir von u aus Kanten $e = \{u, w\}$ mit $e \in M$ und $level[w] = level[u] + 1$.

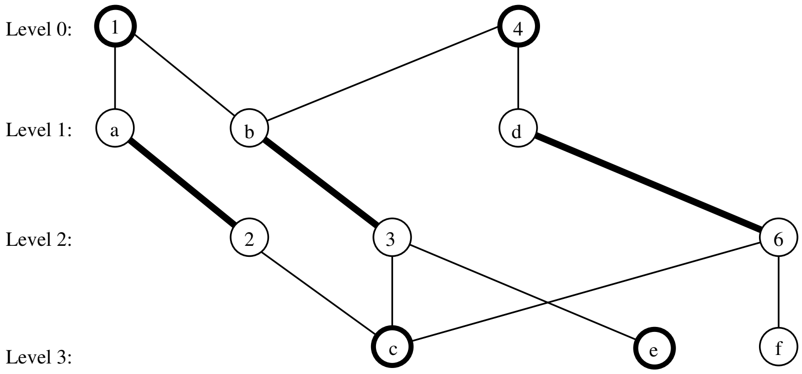
Erreicht eine solche Tiefensuche von einem freien Knoten v aus V_1 einen freien Knoten w aus V_2, so ist der zugehörige Pfad von v nach w ein kürzester augmentierender Pfad. In diesem Fall wird der Pfad sofort invertiert (Kanten, die in M waren, werden aus M herausgenommen, und Kanten, die noch nicht in M waren, werden in M eingefügt), alle Knoten u auf dem Pfad werden durch Setzen von $level[u] = -1$ für weitere Tiefensuchen gesperrt, und die nächste Tiefensuche wird beim nächsten freien Knoten aus V_1 gestartet. Läuft eine Tiefensuche in eine Sackgasse, d.h. es wurde noch kein augmentierender Pfad gefunden und es gibt vom aktuellen Knoten u aus keine den obigen Bedingungen genügende Kante mehr, so wird $level[u] = -1$ gesetzt und die Tiefensuche am Vorgänger von u fortgesetzt. Die Level-Werte der Knoten werden während der Tiefensuchen nur dann verändert, wenn ein augmentierender Pfad oder eine Sackgasse gefunden wurde: in diesem Fall wird durch Setzen des Level-Wertes auf -1 erreicht, dass die betroffenen Knoten für den Rest der Tiefensuche ignoriert werden.

1.3 Beispiel des Ablaufs einer Iteration

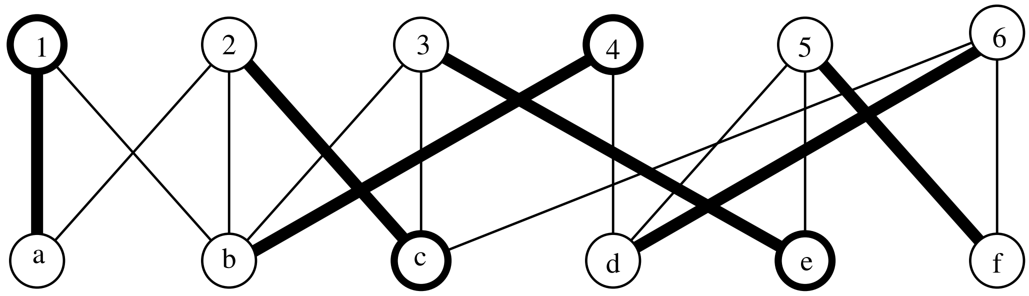
Der bipartite Graph $G = (V_1 \cup V_2, E)$ mit $V_1 = \{1, 2, 3, 4, 5, 6\}$ und $V_2 = \{a, b, c, d, e, f\}$ und das aktuelle Matching M seien wie folgt gegeben (gematchte Kanten sind fett gezeichnet, freie Knoten sind dick umrandet):



Die Zahlen außerhalb der Knoten geben den $level$-Wert an, der sich bei der simultanen Breitensuche ergibt. Nach der dritten Phase der simultanen Breitensuche befinden sich die Knoten c, e und f in der Queue. Da darunter auch zwei freie Knoten (c und e) sind, terminiert die simultane Breitensuche, und es wird nacheinander bei den freien Knoten aus V_1, also bei Knoten 1 und 4, jeweils eine Tiefensuche gestartet. Dabei werden nur die folgenden Kanten betrachtet:

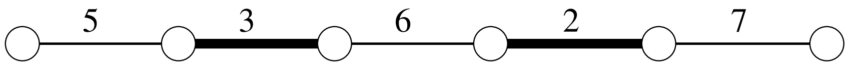


Es könnten dann z.B. die Pfade 1, a, 2, c und 4, b, 3, e gefunden werden, die eine maximale Menge knotendisjunkter kürzester augmentierender Pfade darstellen. (Auch der Pfad 4, b, 3, c alleine wäre übrigens eine solche Menge.) Nach Invertierung dieser beiden Pfade ergibt sich folgendes Matching, das bereits maximale Kardinalität hat.



2 Gewichtete Matchings in bipartiten Graphen

Es sei ein ungerichteter Graph $G = (V, E)$ mit ganzzahligen Kantengewichten $w : E \to \mathbb{Z}$ gegeben. Ein *Matching* in G ist eine Teilmenge $M \subseteq E$, so dass keine zwei Kanten aus M einen Endpunkt gemeinsam haben. Ein Matching M heißt *Matching maximaler Kardinalität* falls es in G kein Matching M' mit $|M'| > |M|$ gibt. Das Gewicht $w(M)$ eines Matchings M ist die Summe der Gewichte aller Kanten in M. Das Gewicht $w(p)$ eines augmentierenden Pfades p bezüglich M ist die Summe der Gewichte aller Kanten aus $p \setminus M$ abzüglich der Summe der Gewichte aller Kanten aus $p \cap M$. Das Gewicht des folgenden augmentierenden Pfades wäre also $5 + 6 + 7 - 3 - 2 = 13$.



Mit dieser Definition ändert sich das Gewicht eines Matchings durch Invertieren eines augmentierenden Pfades p genau um $w(p)$.

Wir haben bereits den Algorithmus von Hopcroft und Karp kennengelernt, der in bipartiten Graphen effizient in Zeit $O(\sqrt{|V|}|E|)$ ein Matching maximaler Kardinalität berechnet. Oft gibt es jedoch in einem gegebenen Graphen verschiedene Matchings maximaler Kardinalität, von denen man nicht ein beliebiges, sondern eines mit minimalem oder maximalem Gewicht haben möchte. Im folgenden betrachten wir den Fall, dass ein Matching maximaler Kardinalität und minimalen Gewichts gesucht ist.

2.1 Inkrementelle Berechnung

Die Grundidee zur Lösung dieses Problems ist, nacheinander für $k = 1, 2, \ldots$ jeweils ein Matching M_k zu berechnen, das genau k Kanten enthält und unter allen Matchings mit k Kanten minimales Gewicht hat. Der folgende Satz zeigt, wie M_{k+1} aus M_k berechnet werden kann.

Satz 2 *Sei M_k ein Matching in G mit k Kanten, das unter allen Matchings in G, die genau k Kanten enthalten, minimales Gewicht hat. Sei p ein augmentierender Pfad in G bzgl. M_k, der unter allen augmentierenden Pfaden bzgl. M_k minimales Gewicht hat. Dann enthält $M_{k+1} := M_k \oplus p$, d.h. das durch Invertieren von p aus M_k entstehende Matching, genau $k + 1$ Kanten und hat unter allen Matchings in G, die genau $k + 1$ Kanten enthalten, minimales Gewicht.*

Beweis: Sei M' ein beliebiges Matching mit $k + 1$ Kanten, das unter allen Matchings mit $k + 1$ Kanten minimales Gewicht hat. Betrachte den Graphen $H = (V, M_k \oplus M')$, der nur die Kanten enthält, die entweder in M_k oder in M' enthalten sind (aber nicht in beiden). Alle Knoten haben in H Grad höchstens 2, und die Zusammenhangskomponenten von H können daher nur die folgende Form haben:

1. alternierende (bzgl. M_k und M') Pfade oder Kreise mit gerader Kantenanzahl

2. augmentierende Pfade bzgl. M_k (d.h. durch Invertieren eines solchen Pfades wird M_k um Eins größer bzw. M' um Eins kleiner)

3. augmentierende Pfade bzgl. M' (d.h. durch Invertieren eines solchen Pfades wird M_k um Eins kleiner bzw. M' um Eins größer)

Auf alternierenden Pfaden oder Kreisen mit gerader Kantenanzahl muss das Gewicht der Kanten aus M_k gleich dem Gewicht der Kanten aus M' sein, da sonst durch Invertieren entweder M_k oder M' billiger gemacht werden könnte (Widerspruch). Weiter muss die Anzahl augmentierender Pfade bzgl. M_k (2.) um genau Eins größer sein als die Anzahl augmentierender Pfade bzgl. M' (3.).

Enthält H keinen augmentierenden Pfad bzgl. M' (3.), so sind wir fertig: H enthält genau einen augmentierenden Pfad p bzgl. M_k, und durch Invertieren von p (oder jedes anderen augmentierenden Pfades mit minimalem Gewicht) erhalten wir aus M_k ein Matching mit $k+1$ Kanten und minimalem Gewicht.

Enthält H augmentierende Pfade bzgl. M' (3.), so muss für jedes Paar eines augmentierenden Pfades p bzgl. M_k und eines augmentierenden Pfades q bzgl. M' gelten: die Summe der Kantengewichte in $(p \cup q) \cap M_k$ ist gleich der Summe der Kantengewichte in $(p \cup q) \cap M'$. (Ansonsten könnte man durch Invertieren von p und q entweder M_k oder M' billiger machen, ein Widerspruch.) Also müssen alle augmentierenden Pfade bzgl. M_k das selbe Gewicht $c \in \mathbb{Z}$ und alle augmentierenden Pfade bzgl. M' das selbe Gewicht $-c$ haben. Auch in diesem Fall folgt sofort, dass man ein Matching mit $k+1$ Kanten und minimalem Gewicht aus M_k durch Invertieren eines einzigen augmentierenden Pfades bzgl. M_k von Gewicht c erhalten kann, also auch durch Invertieren eines beliebigen augmentierenden Pfades mit minimalem Gewicht. $\square$

Der Algorithmus zur Berechnung eines Matchings maximaler Kardinalität und minimalen Gewichts sieht also wie folgt aus:

- setze $M := \emptyset$

- **while** (es existiert augmentierender Pfad bzgl. M) **do**
  suche einen augmentierenden Pfad p mit minimalem Gewicht;
  invertiere p und vergrößere damit M um eine Kante;
  od

- gib M als Matching maximaler Kardinalität und minimalen Gewichts aus

2.2 Berechnung augmentierender Pfade minimalen Gewichts

Nun muss nur noch geklärt werden, wie man in einem bipartiten Graphen einen augmentierenden Pfad mit minimalem Gewicht findet. Wir wollen also unter allen alternierenden Pfaden, die von einem freien Knoten aus V_1 zu einem freien Knoten aus V_2 laufen, einen finden, dessen Gewicht minimal ist. Wir beobachten, dass jeder solche Pfad Kanten aus $E \setminus M$ nur in Richtung von V_1 nach V_2 besucht und Kanten aus M nur in Richtung von V_2 nach V_1. Wir können also die Kanten zu gerichteten Kanten machen. Ferner können wir den Kanten Kosten zuordnen derart, dass das Gewicht eines augmentierenden Pfades genau gleich der Summe der Kosten auf dem Pfad ist: dazu weisen wir jeder Kante $e \in E \setminus M$ die Kosten $c(e) = w(e)$ und jeder Kante $e \in M$ die Kosten $c(e) = -w(e)$ zu. Wenn wir nun noch zwei neue Knoten s und t einfügen, s über gerichtete Kanten mit $c(e) = 0$ mit allen freien Knoten in V_1 verbinden und alle freien Knoten in V_2 über gerichtete Kanten mit $c(e) = 0$ mit t, so lässt sich ein augmentierender Pfad mit minimalem Gewicht offensichtlich durch Suchen eines kürzesten Pfades von s nach t in diesem erweiterten Graphen G' finden. Es bleibt also nur noch zu untersuchen, wie diese Suche nach einem kürzesten Pfad von s nach t in G' erfolgen soll. (In Abschnitt 2.3 finden sich Beispiele dieser Konstruktion.)

2.2.1 Negative Gewichte und Rekalibrierung

Da Kanten in G' auch negatives Gewicht haben können, scheint es so, als ob wir den Algorithmus von Dijkstra nicht einsetzen könnten. Stattdessen wäre der Algorithmus von Bellman und Ford einsetzbar, der kürzeste Pfade in Graphen mit beliebigen Kantengewichten (aber ohne Kreise negativer Länge) in Zeit $O(|V| \cdot |E|)$ berechnet. Die sich ergebende Gesamtlaufzeit für das Finden eines Matchings maximaler Kardinalität und minimalen Gewichts wäre damit $O(|V|^2|E|)$, weil im schlimmsten Fall $|V|/2$ augmentierende Pfade gesucht werden müssen.

Mittels eines geschickten „Rekalibrierungs“-Tricks können wir jedoch die Kosten der Kanten positiv machen und dann doch den Algorithmus von Dijkstra einsetzen. Da der Algorithmus von Dijkstra bei Implementierung mittels Fibonacci-Heaps Worst-Case-Laufzeit $O(|E| + |V| \log |V|)$ hat, ergibt sich damit für das Finden eines Matchings maximaler Kardinalität und minimalen Gewichts (unter allen solchen Matchings) die Gesamtlaufzeit $O(|V|^2 \log |V| + |V| \cdot |E|)$.

Wie funktioniert nun eine solche Rekalibrierung? Betrachten wir zuerst einmal eine beliebige Funktion $F : V \to \mathbb{Z}$ und ersetzen in G' die Kosten $c(u, v)$ jeder Kante (u, v) durch $c'(u, v) := c(u, v) + F(u) - F(v)$. Die Länge jeden Pfades von s nach t ändert sich durch eine solche Rekalibrierung um genau $F(s) - F(t)$. Da diese Änderung nicht vom Pfad abhängt, bleibt ein kürzester Pfad von s nach t auch im Graph mit modifizierten Gewichten ein kürzester Pfad von s nach t. Wenn wir eine Funktion F finden, die alle $c'(u, v)$ positiv macht, so können wir anschließend einen augmentierenden Pfad minimalen Gewichts mit dem Algorithmus von Dijkstra bestimmen.

2.2.2 Wahl der Rekalibrierungsfunktion

Wir benötigen also eine Funktion F, so dass alle $c'(e)$ durch die Rekalibrierung nicht-negativ werden. Eine Möglichkeit ist, $F(u)$ gleich der Länge eines kürzesten Pfades von s nach u zu wählen: damit gilt nämlich für jede Kante (u, v) offensichtlich $F(v) \leq F(u) + c(u, v)$, woraus direkt $c'(u, v) \geq 0$ folgt. Problem dabei ist, dass wir eigentlich die kürzesten Pfade erst mit Hilfe der Rekalibrierung berechnen wollen. Die Lösung des Dilemmas ist, einfach die Informationen aus der Suche nach dem augmentierenden Pfad mit minimalem Gewicht im vorherigen Schritt auszunutzen!

Falls am Anfang, d.h. vor Suche des ersten augmentierenden Pfads, Kanten mit negativem Gewicht existieren, so addiert man einfach auf alle Kantengewichte eine genügend große positive Zahl, um alle Gewichte nicht-negativ zu machen. (Das kann man machen, weil sich dadurch das Gewicht aller Matchings maximaler Kardinalität gleich verändert.) Somit kann die Suche nach dem ersten augmentierenden Pfad bereits mit dem Algorithmus von Dijkstra realisiert werden. Der Graph G' und die Kosten der Kanten ergeben sich dabei wie oben beschrieben. Man lässt den Algorithmus von Dijkstra in G' laufen, bis die Längen der kürzesten

Pfade zu allen Knoten von G' berechnet sind (*single source*-Problem).

Wir bezeichnen mit $c(u, v)$ die aktuellen Kosten der Kanten in G' und mit $d(v)$ die berechnete Länge eines kürzesten Pfades von s nach v. Ferner sei p der kürzeste Pfad in G' von s nach t, den der Algorithmus von Dijkstra berechnet hat. Dann führt der Matching-Algorithmus die folgenden Operationen aus:

- Der p entsprechende augmentierende Pfad in G wird invertiert.
- Die erste und letzte Kante von p wird aus G' gelöscht.
- Für alle Kanten (u, v) von G', die nicht auf p liegen, werden die Kosten $c(u, v)$ durch $c(u, v) + d(u) - d(v) \geq 0$ ersetzt.
- Alle in G' verbleibenden Kanten (u, v) von p (d.h. alle Kanten außer der ersten und der letzten) werden umgedreht (d.h. aus (u, v) wird (v, u)) und erhalten neue Kosten $c(v, u) = 0$.

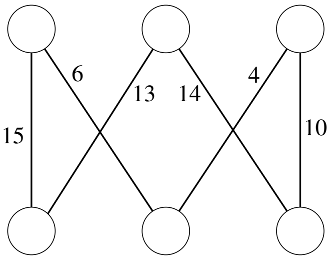
Man überzeugt sich leicht, dass durch diese Modifikationen genau ein Graph G' entsteht, der zur Berechnung des nächsten augmentierenden Pfades mit minimalem Gewicht unter Verwendung des Algorithmus von Dijkstra dienen kann. Der Prozess (Berechnung kürzester Pfade in G' einschließlich eines kürzesten Pfades p von s nach t, Invertieren des p entsprechenden augmentierenden Pfades mit minimalem Gewicht in G, Modifizieren von G' wie oben beschrieben) wird wiederholt, bis irgendwann in G' kein Pfad von s nach t mehr existiert. Dann ist das aktuelle Matching in G genau das gesuchte Matching maximaler Kardinalität, das unter allen solchen Matchings minimales Gewicht hat.

Bemerkung: Da der vorgestellte Algorithmus auch mit negativen Kantengewichten zurecht kommt, kann man auch einfach ein Matching maximaler Kardinalität berechnen, das unter allen solchen Matchings maximales (statt minimales) Gewicht hat. Dazu reicht es aus, zu Beginn des Algorithmus alle Kantengewichte zu negieren.

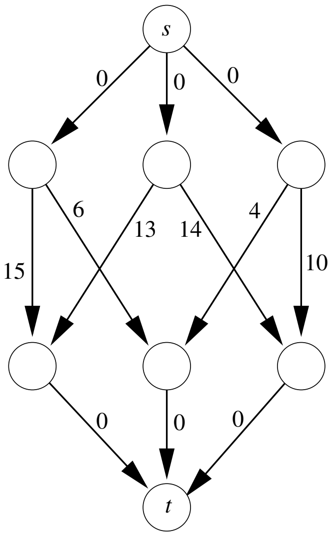
Weitere Informationen: Diese Darstellung des Algorithmus zur Berechnung von Matchings maximaler Kardinalität und minimalen bzw. maximalen Gewichts basiert auf den Seiten 81–85 aus dem Skript "Advanced Algorithms" zu einer Vorlesung von Johan Håstad. Dieses Skript ist als PostScript-Datei im WWW auf Håstad's Homepage `http://www.nada.kth.se/~johanh/` verfügbar.

2.3 Beispiellauf des Algorithmus

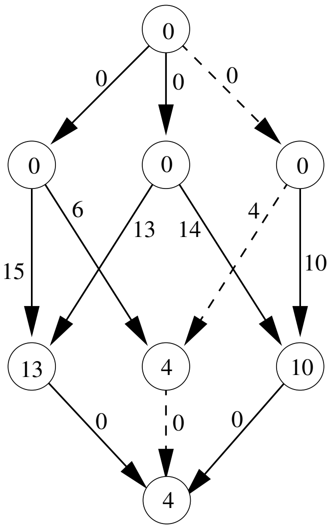
Im folgenden wollen wir den vorgestellten Algorithmus Schritt für Schritt an einem Beispiel durchrechnen.



(1) Eingabegraph G

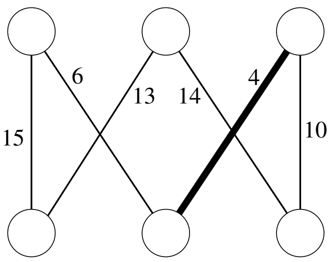


(2) Graph G'

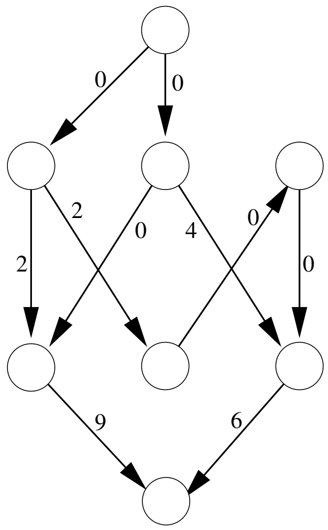


(3) kürzeste Pfade in G'

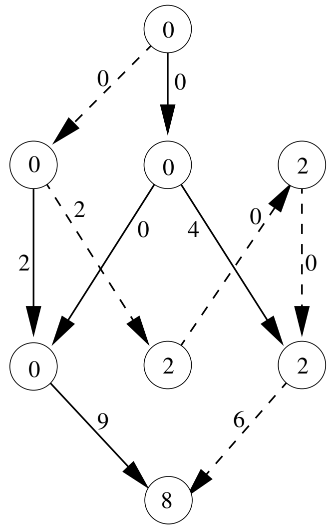
Der Eingabegraph G ist in (1) dargestellt. Die Kantengewichte von G sind alle positiv, so dass anfänglich keine Modifikation der Gewichte nötig ist. Der Graph G' ergibt sich wie in Abschnitt 2.2 beschrieben und ist in (2) dargestellt. Nach der Berechnung kürzester Pfade von s aus in G' mittels des Algorithmus von Dijkstra ergeben sich die in (3) gezeigten Abstände der Knoten von s, und der kürzeste Pfad p von s nach t ist gestrichelt dargestellt. Durch Invertieren des zugehörigen augmentierenden Pfades in G erhalten wir das in (4) dargestellte Matching M_1, das unter allen Matchings in G, die genau eine Kante enthalten, minimales Gewicht hat.



(4) Matching M_1 in G

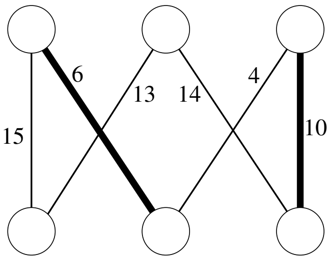


(5) neuer Graph G'

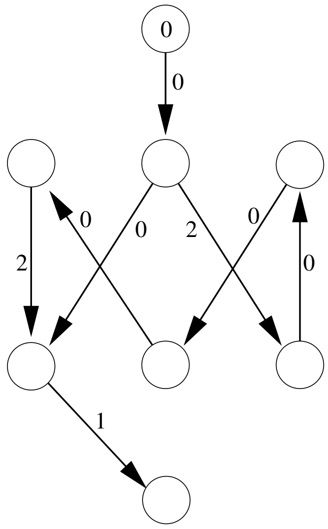


(6) kürzeste Pfade in G'

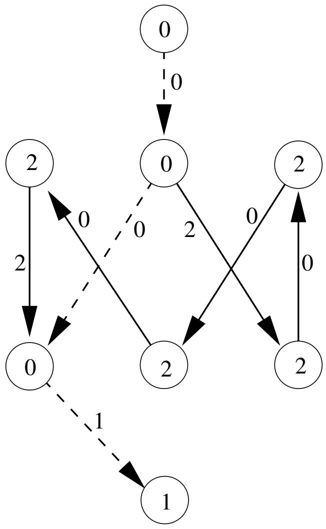
Durch Löschen der ersten und letzten Kante von p, durch Umdrehen der verbleibenden Kante von p und durch Modifikation der Gewichte entsteht aus dem alten Graphen G' in (3) der neue Graph G' in (5). In diesem Graphen wird wieder der Algorithmus von Dijkstra ausgeführt; er liefert diesmal die in (6) dargestellten Abstandswerte sowie einen neuen kürzesten Pfad p (gestrichelt dargestellt) von s nach t. Invertieren des zugehörigen augmentierenden Pfades in G liefert das in (7) dargestellte Matching M_2, das unter allen Matchings in G, die genau zwei Kanten enthalten, minimales Gewicht hat.



(7) Matching M_2 in G

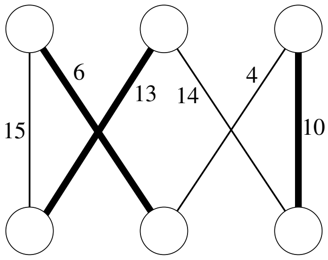


(8) neuer Graph G'



(9) kürzeste Pfade in G'

Wiederum ist der neue Graph G' in (8) abgebildet und das Ergebnis der Berechnung kürzester Pfade in G' in (9). Invertieren des augmentierenden Pfades in G, der dem gestrichelt gezeichneten, kürzesten Pfad in G' entspricht, liefert schließlich das Matching M_3 in (10). Im Graph G', der sich daraus ergibt, ist t von s aus nicht mehr erreichbar. Also hat M_3 maximale Kardinalität und unter allen Matchings maximaler Kardinalität in G minimales Gewicht, nämlich $w(M_3) = 29$.



(10) Matching M_3 in G