(M. Dietzfelbinger, 12.11.2020)

2 Matchings in bipartiten Graphen

2.1 Grundbegriffe

Englischunterricht:

> "*to match*": passend (paarweise) zusammenfügen
> "*matchmaker*": Heiratsvermittler(in)

Beispiel 1: (s. Abb. 1 links und 2 rechts) Wir veranstalten einen Schulausflug mit $2k$ Kindern. Eine Menge von Paaren $E \subseteq \{\{i,j\} \mid i,j \in \{1,\ldots,2k\}, i \neq j\}$ legt fest, wer mit wem im gleichen (Zweier-)Zimmer schlafen kann. Ist es möglich, die Kinder auf genau k Zimmer aufzuteilen?

Beispiel 2 (das Standardbeispiel, s. Abb. 3): U und W sind n-elementige Mengen, die für n Männer und n Frauen stehen. Eine Menge $E \subseteq U \times W$ gibt an, wer mit wem „kann" (was auch immer: im Bus nebeneinander sitzen, Tango tanzen, heiraten). Gibt es eine Möglichkeit, die Personen einander paarweise zuzuordnen, so dass die durch E gegebene Einschränkung eingehalten wird? Technisch: Gibt es eine Teilmenge $M \subseteq E$ mit $|M| = n$ derart, dass jedes $i \in U$ und jedes $j \in W$ in M genau einmal vorkommt? Wenn nicht, finde eine möglichst große Menge von kompatiblen Paaren.

Beispiel 3: Eine Firma beschäftigt n Personen und hat n Tätigkeiten auszuführen. Die Personen sind bei den Jobs unterschiedlich schnell und fehleranfällig.

	Spülen	Putzen	Kochen	Bügeln
Anton	1	2	0	5
Berti	0	3	4	1
Conni	3	1	2	6
Det	2	0	3	4

Wenn Person i Job j ausführt, entsteht Wert w_{ij} (pro Stunde). Wie sollte man die Jobs auf die Personen aufteilen, damit der erzielte Wert in Summe möglichst groß wird? Hier ist nach einem „perfekten Matching" $M \subseteq U \times W$ mit $|M| = n$ gefragt, wobei $U = W = \{1,\ldots,n\}$ gilt, so dass $\sum_{(i,j)\in M} w_{ij}$ möglichst groß ist.

Wir definieren den Begriff „Matching" zunächst für allgemeine Graphen. Damit ist eine Menge von einander nicht berührenden Kanten gemeint. (Passend: Beispiel 1.)

Definition 2.1.1. *$G = (V,E)$ sei ein (ungerichteter) Graph. Ein **Matching** in G ist eine Menge $M \subseteq E$ mit folgender Eigenschaft: wenn (u,v) und (x,w) verschiedene Elemente von M sind, dann ist $\{u,v\} \cap \{x,w\} = \emptyset$.*

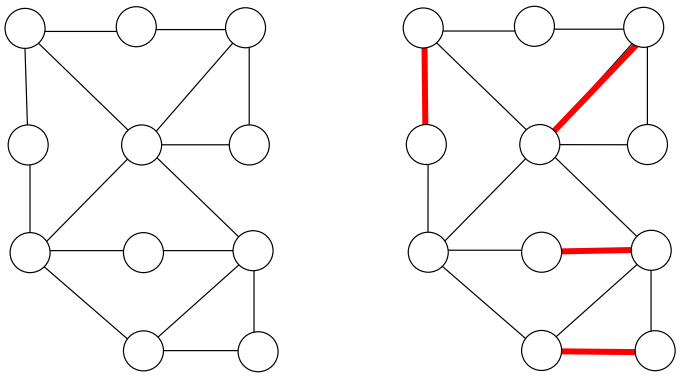

Abbildung 1: Ein Graph und ein (nicht erweiterbares) Matching

Definition 2.1.2. (a) *Ein Matching M in $G = (V, E)$ heißt **nicht erweiterbar** oder **inklusionsmaximal** (engl.: maxim**al** matching), wenn es keine Kante $e \notin M$ gibt, so dass $M' = M \cup \{e\}$ ebenfalls Matching ist.*

(b) *Ein Matching M in $G = (V, E)$ heißt **größtes Matching** oder **maximales Matching** oder **kardinalitätsmaximal** (engl.: maxim**um** matching), wenn es kein Matching M' mit $|M'| > |M|$ gibt.*

(c) *Ein Matching M in $G = (V, E)$ heißt **perfektes Matching**, wenn jeder Knoten $v \in V$ in einer Kante in M vorkommt.*

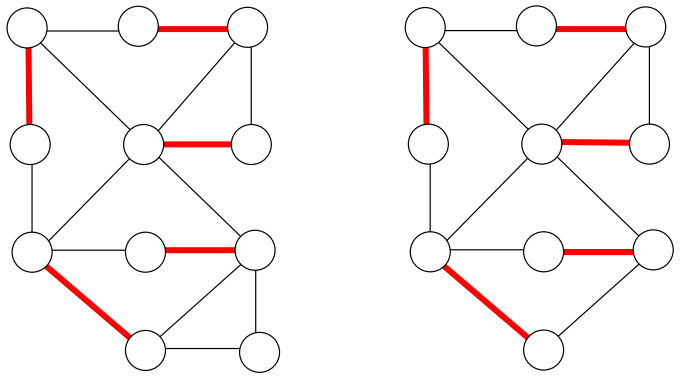

Abbildung 2: Ein (kardinalitäts)maximales Matching und ein perfektes Matching

Die Situation in Beispiel 1 kann als Suche nach einem perfekten Matching in einem allgemeinen Graphen modelliert werden, die in Beispiel 2 als Suche nach einem perfekten oder zumindest (kardinalitäts)maximalen Matching in einem „bipartiten“ Graphen, in Beispiel 3 geht es um ein perfektes Matching in einem vollständigen bipartiten Graphen, das eine gegebene Kostenfunktion maximiert.

Definition 2.1.3. *Ein (ungerichteter) Graph $G = (V, E)$ heißt **bipartiter** oder **paarer** Graph, wenn man V in disjunkte Mengen U und W zerlegen kann, so dass alle Kanten zwischen U und W verlaufen.*

Äquivalent hierzu ist die Bedingung, dass man die Knoten in V so mit zwei Farben („blau“ und „grün“) färben kann, dass jede Kante zwei verschiedenfarbige Endpunkte hat. Es gibt einfache Algorithmen, die einen Graphen G in Zeit $O(|V| + |E|) = O(n + m)$ darauf testen, ob er bipartit ist, und gegebenenfalls eine solche Aufteilung/Färbung berechnen (Übung).

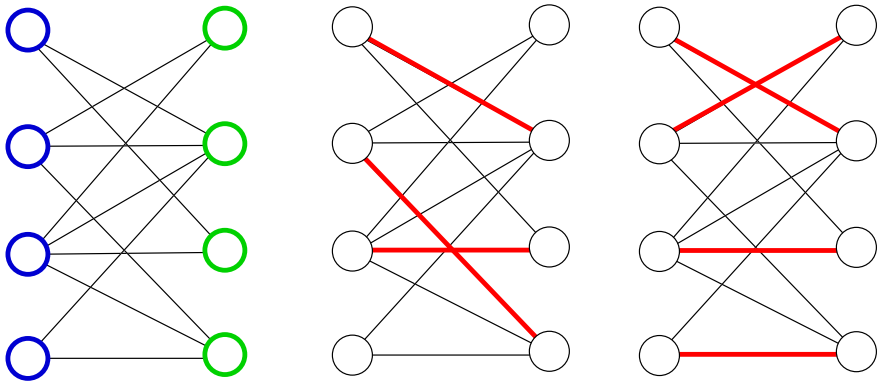

Abbildung 3: Ein bipartiter Graph, mit nicht erweiterbarem Matching, mit perfektem Matching

In diesem Kapitel betrachten wir Algorithmen, die in einem gegebenen Sinn bestmögliche Matchings für bipartite Graphen finden.

2.2 Kostenoptimale Matchings in bipartiten Graphen mit Kantengewichten: Auktionen

(Demange, Gale, Sotomayor, Multi-Item Auctions, Journal of Political Economy, Vol. 94, No. 4, Aug., 1986, pp. 863-872.)

Gegeben ist eine Menge $U = \{1, \ldots, n_\mathrm{b}\}$ (von „Bietern“) und eine Menge $W = \{1, \ldots, n_\mathrm{w}\}$ (von „Waren“), sowie eine „Wertmatrix“

$$(w_{ij})_{1 \le i \le n_\mathrm{b}, 1 \le j \le n_\mathrm{w}}$$

mit Werten $w_{ij} \in \mathbb{N}$. (Wert w_{ij} drückt aus, wie viel Ware j für Bieter i wert ist.) Das Ziel ist, einige der Waren Bietern zuzuordnen, und zwar jedem Bieter höchstens eine, so dass der gesamte realisierte Wert möglichst groß ist. Gesucht ist also ein Matching M im bipartiten Graphen $G = (U \dot\cup W, U \times W)$, so dass $w(M) = \sum_{(i,j) \in M} w_{ij}$ möglichst groß ist.

	Uhr	Fahrrad	Sessel	Reise
Alice	0	3	2	3
Bob	4	0	10	10
Carl	3	9	0	2
Daisy	5	7	0	12

Abbildung 4: Beispielinput für Auktionsmatching. Bieter: $i = 1, 2, 3, 4$ sind Alice, Bob, Carl und Daisy; Waren $j = 1, 2, 3, 4$ sind Uhr, Fahrrad, Sessel, Reise.
Eine mögliche Zuordnung ist $(\text{Alice}, \text{Reise})$, $(\text{Bob}, \text{Sessel})$, $(\text{Carl}, \text{Fahrrad})$, $(\text{Daisy}, \text{Uhr})$; diese hat Wert $3 + 10 + 9 + 5 = 27$. Sie ist nicht optimal.

Man kann offenbar auch das Problem, ein kardinalitätsmaximales Matching M in einem bipartiten Graphen $G = (U, W, E)$ zu finden, in dieser Terminologie formulieren: $w_{ij} \in \{0, 1\}$ für alle i, j, mit $w_{ij} = 1$ genau dann wenn $(i, j) \in E$. Dies stellt den einfachsten Fall dar.

Wir wollen das Problem lösen, indem wir eine Auktion durchführen, und zwar für alle Waren auf einmal. Wie bei einer normalen Auktion gibt es Gebote für die Waren; das aktuelle Höchstgebot für eine Ware j nennen wir den Preis p_j. Das (fiktive) Mindestgebot ist 0: der Preis jeder Ware ist anfangs 0.

Aus der Sicht einer einzelnen Ware j sieht die Auktion ganz normal aus. Die Bieter geben nacheinander Gebote ab. Diese Gebote steigen bei jedem neuen Gebot immer um einen festen Schrittwert $\delta > 0$, so dass das Höchstgebot bei einer Ware nacheinander δ, 2δ, 3δ, ... ist. Der Bieter, der aktuell das Höchstgebot für Ware j abgegeben hat, ist (temporärer) Besitzer, genannt $\text{owner}(j)$. Mit jedem höheren Gebot ändert sich dieser Besitzer. Natürlich dürfen Bieter wiederholt (immer höhere) Gebote für Ware j abgeben. Wenn die Auktion endet, erhält der aktuelle Besitzer $\text{owner}(j)$ die Ware j endgültig (und muss den aktuellen Preis p_j bezahlen).

Aus der Sicht der Bieter sieht es so aus: Jeder darf im Prinzip für jede Ware bieten; wer momentan Besitzer einer Ware ist, darf nicht bieten. Durch die Wahl $w_{ij} = 0$ wird bewirkt, dass sich Bieter i für Ware j nicht interessiert und nie für sie bietet.

Damit es nicht zu sehr durcheinander geht, läuft die Auktion in Runden. In einer Runde darf nur ein Bieter in Aktion treten, und zwar einer, der momentan keine Ware besitzt. Er sucht sich eine Ware aus, die er haben möchte, und gibt sein Gebot ab, sagen wir für Ware j, das um δ höher sein muss als das bisherige Höchstgebot p_j. Bleibt nur noch die Frage, wie ein Bieter entscheidet, für welche Ware er bieten will. Hier stellen wir uns vor, dass Bieter i mit Ware j „Gewinn“ $w_{ij} - (p_j + \delta)$ erzielt. (Er zahlt $p_j + \delta$ und zieht Wert w_{ij} aus dem Besitz.) Nun gibt es zwei Fälle. Wenn der Gewinn für alle Waren negativ ist, steigt Bieter i aus der Auktion aus – er kann

nichts mehr gewinnen. Andernfalls sucht er sich eine Ware j aus, die ihm den größten Gewinn bietet, also eine, für die $w_{ij} - p_j$ am größten ist, und bietet für diese.

Die Auktion endet, wenn kein Bieter mehr bieten möchte, das heißt, wenn jeder Bieter, der keine Ware besitzt, ausgestiegen ist.

Der folgende Algorithmus baut diese Sorte von Auktion nach. Die Bieter, die zur Abgabe eines Gebots bereit sind, halten sich in einer „Wartemenge“ Q auf. (Dabei spielt die Ordnung, in der Bieter aus Q entnommen werden, keine Rolle; manchmal nennt man eine solche Datenstruktur „Halde“.) Wir setzen $\delta = 1/(n_\mathrm{b}+1)$. Der Grund hierfür wird aus der Analyse klar werden.

Algorithm 1: Auktions-Matching

```
1  // Berechnet ein gewichtsmaximales Matching zu (w_ij)_{1≤i≤n_b, 1≤j≤n_w}
2  Initialisierung:
3     Für jede Ware j: setze p_j ← 0 und owner(j) ← NIL
4     Füge alle Bieter in die Wartemenge Q ein
5     Setze δ := 1/(n_b + 1).        // Dann gilt: δ n_b < 1.
6  Bietevorgang:
7     Solange Q nicht leer ist:
8           Entnehme einen Bieter i aus Q
9           Bestimme ein j, das w_ij − p_j maximiert        // bei Gleichheit: beliebig
10          Falls w_ij − p_j > 0:    // falls w_ij ≤ p_j: Bieter i wird inaktiv
11                Füge (bisherigen Besitzer) owner(j) in Q ein
12                owner(j) ← i
13                p_j ← p_j + δ
14    Ausgabe: Menge M aller Paare (owner(j), j) mit owner(j) ≠ NIL
```

Beispiel: Ablauf des Auktionsalgorithmus. Eine Zeile entspricht einer Runde. Jeweils links: Werte w_{ij} und p_j. Jeweils rechts: $w_{ij} - p_j$. Q: in Wartemenge. Hellgrau: $(\mathrm{owner}(j), j)$. Dunkelgrau: in dieser Runde neu zugewiesen.

p_j	0	0	0
Q	1	5	3
Q	1	4	3
Q	0	1	1

p_j	0	0	0
aktiv	1	5	3
Q	1	4	3
Q	0	1	1

p_j	0	0.25	0
	1	5	3
Q	1	4	3
Q	0	1	1

p_j	0	0.25	0
	1	4.75	3
Q	1	3.75	3
aktiv	0	0.75	1

p_j	0	0.25	0.25
	1	5	3
Q	1	4	3
	0	1	1

p_j	0	0.25	0.25
	1	4.75	2.75
aktiv	1	3.75	2.75
	0	0.75	0.75

p_j	0	0.5	0.25
Q	1	5	3
	1	4	3
	0	1	1

p_j	0	0.5	0.25
aktiv	1	4.5	2.75
	1	3.5	2.75
	0	0.5	0.75

p_j	0	0.75	0.25
	1	5	3
Q	1	4	3
	0	1	1

p_j	0	0.75	0.25
	1	4.25	2.75
aktiv	1	3.25	2.75
	0	0.25	0.75

p_j	0	1	0.25
Q	1	5	3
	1	4	3
	0	1	1

p_j	0	1	0.25
aktiv	1	4	2.75
	1	3	2.75
	0	0	0.75

p_j	0	1.25	0.25
	1	5	3
Q	1	4	3
	0	1	1

p_j	0	1.25	0.25
	1	3.75	2.75
aktiv	1	2.75	2.75
	0	-0.25	0.75

p_j	0	1.5	0.25
Q	1	5	3
	1	4	3
	0	1	1

p_j	0	1.5	0.25
aktiv	1	3.5	2.75
	1	2.5	2.75
	0	-0.5	0.75

p_j	0	1.75	0.25
	1	5	3
Q	1	4	3
	0	1	1

p_j	0	1.75	0.25
	1	3.25	2.75
aktiv	1	2.25	2.75
	0	-0.75	0.75

p_j	0	1.75	0.5
	1	5	3
	1	4	3
Q	0	1	1

p_j	0	1.75	0.5
	1	3.25	2.5
	1	2.25	2.5
aktiv	0	-0.75	0.5

p_j	0	1.75	0.75
	1	5	3
Q	1	4	3
	0	1	1

p_j	0	1.75	0.75
	1	3.25	2.25
aktiv	1	2.25	2.25
	0	-0.75	0.25

p_j	0	2	0.75
Q	1	5	3
	1	4	3
	0	1	1

p_j	0	2	0.75
aktiv	1	3	2.25
	1	2	2.25
	0	-1	0.25

p_j	0	2.25	0.75
	1	5	3
Q	1	4	3
	0	1	1

p_j	0	2.25	0.75
	1	2.75	2.25
aktiv	1	1.75	2.25
	0	-1.25	0.25

p_j	0	2.25	1
	1	5	3
	1	4	3
Q	0	1	1

p_j	0	2.25	1
	1	2.75	2
	1	1.75	2
aktiv gibt auf	0	-1.25	0

Q ist leer, Ende. Ausgegebenes Matching: $\{(1,2),(2,3)\}$ mit (optimalem) Wert 8.

Wir überlegen zunächst, dass der Algorithmus das korrekte Ergebnis liefert, *wenn* er eine Ausgabe erzeugt. (Dass nur endlich viele Runden durchgeführt werden, sehen wir weiter unten.)

Die Analyse beruht darauf, dass festgestellt wird, dass alle Bieter am Ende „zufrieden" sind, in folgendem Sinn:

Definition Ein Bieter i heißt „*δ-zufrieden*", wenn gilt:

(i) Wenn i Ware j hat, d. h. $\text{owner}(j) = i$, dann gilt $w_{ij} \geq p_j$ und für alle Waren j' gilt $\delta + (w_{ij} - p_j) \geq w_{ij'} - p_{j'}$;

(ii) Wenn i keine Ware hat, d. h. es gibt kein j mit $\text{owner}(j) = i$, dann gilt für alle Waren j': $w_{ij'} - p_{j'} \leq 0$.

(Intuition: Im ersten Fall darf die Ware j, die Bieter i momentan besitzt, keinen Verlust einbringen und maximiert bis auf eine Abweichung von δ seinen möglichen Gewinn. Im zweiten Fall sind die Preise aller Waren so hoch, dass Bieter i mit ihnen keinen positiven Gewinn machen kann.)

Schleifeninvariante SI:
Bieter, die nicht in der Wartemenge Q sind, sind δ-zufrieden.

Behauptung: SI gilt immer. Natürlich wird der Beweis durch Induktion über Runden des Algorithmus geführt. Am Anfang ist SI erfüllt, weil alle Bieter in Q sind. Nun betrachten wir eine Runde.

Wenn Bieter i aus Q entnommen wird, gibt es zwei Fälle:

1. Fall: Es gibt eine Ware j mit $w_{ij} - p_j > 0$. – Dann wählt der Algorithmus für i eine solche Ware j aus, die zudem $w_{ij} - p_j \geq w_{ij'} - p_{j'}$ für alle Waren j' erfüllt. Die Ungleichungen werden zu[1] $w_{ij} - p_j \geq 0$ und $\delta + w_{ij} - p_j \geq w_{ij'} - p_{j'}$ für alle j' abgeschwächt, weil der Preis p_j um δ steigt (Zeile 13 des Algorithmus). Damit ist (i) für i erfüllt. Solange i Besitzer von j ist, bleibt (i) für i gültig, weil die einzige mögliche Änderung ist, dass die $p_{j'}$ für $j' \neq j$ größer werden.

2. Fall: Für alle Waren j' gilt $w_{ij'} - p_{j'} \leq 0$. – Dann gibt Bieter i auf, und er wird nie mehr in die Wartemenge Q kommen, aber er ist δ-zufrieden, weil für ihn (ii) gilt. Da Preise $p_{j'}$ nur steigen können, bleibt (ii) gültig.

Wenn in der betrachteten Runde Bieter i seine Ware weggenommen wird, dann wandert i nach Q, und SI gilt auch für i.

Damit ist die Behauptung bewiesen.

Es ist offensichtlich, dass der Algorithmus irgendwann einmal anhält. (In jeder Runde steigt entweder ein Bieter (endgültig) aus oder ein Preis erhöht sich um δ. Preis p_j kann nicht größer als $\max_{1 \leq i \leq n_b} w_{ij}$ werden.) Wenn der Algorithmus anhält, muss Q leer sein, und daher sind dann alle Bieter δ-zufrieden. Das folgende Lemma betrachtet diese Situation und liefert die Korrektheit des Algorithmus.

[1] Ein Detail: Preise sind immer Vielfache von δ, und Werte sind ganzzahlig. Wenn also $w_{ij} - p_j > 0$ war und nun $p_j := p_j + \delta$ gesetzt wird, gilt nachher $w_{ij} - p_j \geq 0$.

Lemma 2.2.1. *Sei $M' \subseteq U \times W$ ein beliebiges Matching. Wenn alle Bieter δ-zufrieden sind, dann gilt für das (Algorithmen-)Matching $M = \{(i,j) \mid i = owner(j)\}$:*

$$\delta n_{\mathrm{b}} + \sum_{(i,j)\in M} w_{ij} \geq \sum_{(i,j)\in M'} w_{ij} \ .$$

Bevor wir das Lemma beweisen, bemerken wir, dass daraus die Korrektheit des Algorithmus folgt: Weil alle Gewichte ganzzahlig sind und $\delta n_{\mathrm{b}} < 1$ ist (nach der Wahl von δ), folgt $\sum_{(i,j)\in M} w_{ij} \geq \sum_{(i,j)\in M'} w_{ij}$. Da M' beliebig war, ist das vom Algorithmus gelieferte Matching M optimal.

Beweis von Lemma 2.2.1: Wir bemerken zunächst, dass $p_j > 0$ ist genau dann wenn irgendwann einmal irgendein Bieter Besitzer von j geworden ist. Da Waren nie mehr frei werden, wenn sie einmal gewählt worden sind, heißt dass, dass ein Paar $(\mathrm{owner}(j), j)$ in M vorkommt.

Für Bieter i bezeichne $j_i \in W \cup \{\mathrm{NIL}\}$ die Ware, die ihm in M zugeordnet wird, und $j'_i \in W \cup \{\mathrm{NIL}\}$ die Ware, die ihm in M' zugeordnet wird. Weiter setzen wir $p_{\mathrm{NIL}} = 0$ und $w_{i,\mathrm{NIL}} = 0$. Wir behaupten, dass Folgendes gilt:

$$\delta + (w_{ij_i} - p_{j_i}) \geq w_{ij'_i} - p_{j'_i}. \tag{1}$$

Dies folgt daraus, dass i δ-zufrieden ist, wie man durch Betrachten mehrerer Fälle sieht:

Fall A: $(i, j_i) \in M$ und $(i, j'_i) \in M'$. – Dann folgt (1) aus (i).

Fall B: $j_i = \mathrm{NIL}$ und $(i, j'_i) \in M'$. – Dann ist die linke Seite in (1) gleich δ, und die rechte Seite ist nicht positiv, weil auf i Bedingung (ii) zutrifft.

Fall C: $(i, j_i) \in M$ und $j'_i = \mathrm{NIL}$. – Wegen (i) ist die linke Seite in (1) mindestens δ, die rechte Seite ist 0.

Fall D: $j_i = \mathrm{NIL}$ und $j'_i = \mathrm{NIL}$. – Dann ist die linke Seite in (1) gleich δ, die rechte Seite ist 0.

Durch Summieren von (1) über alle Bieter i erhalten wir:

$$n_{\mathrm{b}}\delta + \sum_{1\leq i\leq n_{\mathrm{b}}} (w_{ij_i} - p_{j_i}) \geq \sum_{1\leq i\leq n_{\mathrm{b}}} (w_{ij'_i} - p_{j'_i}) \tag{2}$$

Wir addieren nun $\sum_{1\leq j\leq n_{\mathrm{w}}} p_j$ zu beiden Seiten von (2) und erhalten Folgendes:

$$n_{\mathrm{b}}\delta + \sum_{1\leq i\leq n_{\mathrm{b}}} w_{ij_i} - \sum_{1\leq i\leq n_{\mathrm{b}}} p_{j_i} + \sum_{1\leq j\leq n_{\mathrm{w}}} p_j \geq \sum_{1\leq i\leq n_{\mathrm{b}}} w_{ij'_i} - \sum_{1\leq i\leq n_{\mathrm{b}}} p_{j'_i} + \sum_{1\leq j\leq n_{\mathrm{w}}} p_j. \tag{3}$$

Auf der linken Seite heben sich die beiden p_j-Summen genau auf, weil die Bedingung $p_j > 0$ gleichbedeutend damit ist, dass es ein i mit $j = j_i$ gibt. Auf der rechten Seite gibt die Summe mit negativem Vorzeichen die Summe der Preise all der Waren an, die in M' vorkommen, die Summe mit positivem Vorzeichen die Summe der Preise aller Waren. Also ist der Gesamtbeitrag der beiden p_j-Summen ≥ 0.

Damit ist Lemma 2.2.1 bewiesen. □

Es fehlt nun noch die Laufzeitanalyse und der Nachweis, dass der Algorithmus anhält. Hierzu bemerken wir Folgendes: In jedem Schleifendurchlauf steigt einer der Preise p_j um δ an oder einer der Bieter „gibt auf“, das heißt, er scheidet dauerhaft aus der Wartemenge Q aus. Kein Wert p_j kann größer als $C := \max_{i,j'} w_{ij'}$ werden. Daher ist die gesamte Anzahl der Schleifendurchläufe nicht größer als

$$n_{\mathrm{b}} + \frac{Cn_{\mathrm{w}}}{\delta} \leq C(n_{\mathrm{b}} + n_{\mathrm{w}})/\delta \leq Cn^2,$$

für $n = n_{\mathrm{b}} + n_{\mathrm{w}}$. Jeder Schleifendurchlauf lässt sich trivial mit Laufzeit $O(n)$ implementieren, was eine Gesamtlaufzeit von $O(Cn^3)$ liefert.

Satz 2.2.2. *Sei n die Gesamtzahl der Knoten im Graphen G und sei C das maximale Kantengewicht. Algorithmus 1 liefert in Zeit $O(Cn^3)$ ein gewichtsmaximales bipartites Matching, für n_{b} Bieter und n_{w} Waren und $n = n_{\mathrm{b}} + n_{\mathrm{w}}$. Wenn die Werte nur 0 und 1 sein können, also ein maximales Matching gesucht wird, ist die Laufzeit $O(n^3)$.*

Bemerkung 2.2.3. (1) Man kann auch größere Werte für δ wählen. Lemma 2.2.1 gilt trotzdem. Das heißt, dass der Wert $w(M) = \sum_{(i,j)\in M} w_{ij}$ additiv nur um δn_{b} vom Optimum abweicht. Dies kann man benutzen, wenn man nicht das exakte Optimum benötigt.

(2) Eine besondere Anwendung gibt es beim Berechnen von maximalen Matchings in einem bipartiten Graphen mit n Knoten und m Kanten. Man setzt $\delta = 1/\lceil\sqrt{n}\rceil$. Wenn der Algorithmus stoppt, ist die Abweichung zwischen $|M|$ und dem Optimum nur $n/\lceil\sqrt{n}\rceil \leq \sqrt{n}$. Die Rechenzeit ist $O(m\sqrt{n})$. Im nächsten Abschnitt werden wir sehen, wie man nun mit $O(\sqrt{n})$ Berechnungen von flussvergrößernden Wegen in einem geeigneten Flussnetzwerk ein kardinalitätsmaximales Matching findet. Der Zeitaufwand hierfür ist nochmals $O(m\sqrt{n})$.

2.3 Kardinalitätsmaximale Matchings in bipartiten Graphen

Im Folgenden ist stets $G = (U \cup W, E)$ mit $E \subseteq U \times W$ ein bipartiter Graph mit Knotenmengen U (links) und W (rechts). Wir möchten in diesem Graphen ein (kardinalitäts)maximales Matching berechnen. Es stellt sich heraus, dass Flussalgorithmen dieses Problem direkt lösen.

Sei $G = (U \cup W, E)$ ein bipartiter Graph, mit disjunkten Mengen U und W.[2] Das zugehörige Flussnetzwerk $\hat{G} = (\hat{V}, \hat{E}, \hat{c})$ entsteht aus G wie folgt: $\hat{V}$ ergibt sich aus $U \cup W$ durch Hinzufügen von zwei neuen Knoten q und s. Die Kantenmenge $\hat{E}$ ist E (von U nach W gerichtet) zusammen mit neuen Kanten (q, u), für $u \in U$, und (w, s), für $w \in W$. Alle Kapazitäten sind 1. Wenn wir nun nur ganzzahlige Flüsse betrachten, dann können an einer Kante nur Flusswert 1 oder Flusswert 0 herrschen.

[2]Oft hat man $U = \{1, \ldots, n_1\}$ und $W = \{1, \ldots, n_2\}$. Man nimmt dann stillschweigend an, dass diese beiden Mengen künstlich disjunkt gemacht worden sind, etwa durch Hinzufügen einer zweiten Komponente zum Index.

Proposition 2.3.1. *Flüsse f in $\hat{G}$ mit ganzzahligen Werten $f(e)$ und Matchings M in G entsprechen einander eineindeutig. Dabei entspricht ein Fluss f mit Wert $w(f)$ einem Matching M mit $|M| = w(f)$. Insbesondere entsprechen maximale Flüsse in $\hat{G}$ genau kardinalitätsmaximalen Matchings in G.*

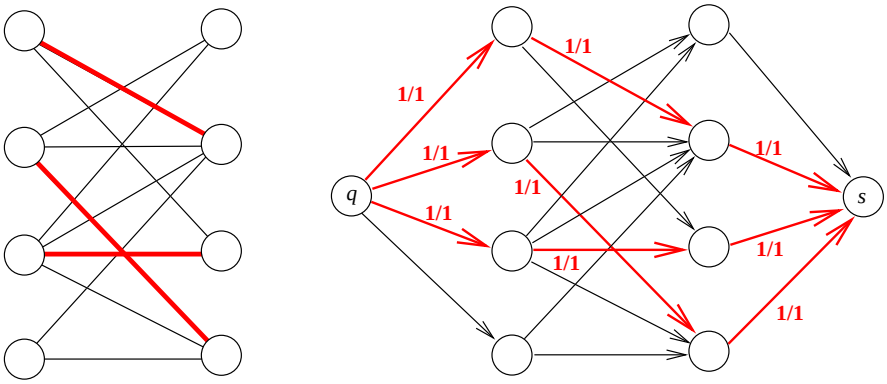

Abbildung 5: Links: Bipartiter Graph mit Matching. Rechts: Entsprechender Flussgraph mit Fluss. Kanten ohne Beschriftung haben Kapazität 1 und Flusswert 0.

Beweis: Wenn ein Matching M gegeben ist, definieren wir einen Fluss $f = f_M$ wie folgt: Für jede Kante $(u, w) \in M$ erhalten die Kanten (q, u), (u, w) und (w, s) Flusswert 1, alle anderen Kanten erhalten Flusswert 0. Es ist klar, dass die Kirchhoff-Gesetze gelten, und es ist klar, dass $w(f_M) = |M|$ ist.

Wenn umgekehrt ein ganzzahliger Fluss f gegeben ist, sieht man unmittelbar, dass dieser 0-1-wertig ist und dass er aus lauter Wegen der Länge 3 von q nach s besteht. Das entsprechende Matching M_f besteht aus den Kanten (u, w) mit $f(u, w) = 1$, für $u \in U$, $w \in W$. □

Damit ergibt sich unmittelbar aus jedem Flussalgorithmus mit der Ganzzahligkeitseigenschaft ein Algorithmus für die Berechnung eines kardinalitätsmaximalen Matchings. (In Frage kommen dafür alle Ford-Fulkerson-Varianten, Edmonds-Karp, Sperrflussvarianten, Dinitz, und alle Versionen der Preflow-Push-Methode.)

2.3.1 Matchingvergrößernde Wege

Definition 2.3.2. *Sei $G = (V, E)$ ein ungerichteter Graph und $M \subseteq E$ ein Matching. Knoten $v \in V$, die nicht auf einer Matchingkante liegen, heißen* ***frei****.*

Ein einfacher Weg $p = (v_0, v_1, \ldots, v_t)$ mit der Eigenschaft, dass sich auf diesem Weg Matching-Kanten mit Nicht-Matching-Kanten abwechseln (d. h. $(v_{i-1}, v_i) \in M \Leftrightarrow (v_i, v_{i+1}) \notin M$) heißt ein ***alternierender Weg (aW)****.*

Ein alternierender Weg $p = (v_0, v_1, \ldots, v_t)$ heißt ***matchingvergrößernder Weg (mvW)****, wenn t ungerade ist und v_0 und v_t frei sind.*

Lemma 2.3.3. *Sei G ein beliebiger Graph. Ein Matching M in G ist kardinalitätsmaximal genau dann wenn es zu G und M keinen matchingvergrößernden Weg gibt.*

Beweis:[3] „⇒“: Wenn M Matching ist und $p = (v_0, v_1, \ldots, v_t)$ ein mvW, dann erhalten wir aus M ein neues, um eine Kante größeres Matching M', indem wir (v_{2i-1}, v_{2i}), $1 \leq i \leq (t-1)/2$, entfernen und (v_{2i}, v_{2i+1}), $0 \leq i \leq (t-1)/2$, hinzufügen.

„⇐“: Sei M ein Matching, das nicht maximal ist. Wähle ein Matching M' mit $|M'| > |M|$. Wir betrachten die symmetrische Differenz $M \oplus M' = (M \cup M') - (M \cap M')$. Diese Menge bildet einen Teilgraphen von G, in dem jeder Knoten Grad 0, 1 oder 2 hat. Daraus folgt, dass es sich um knotendisjunkte Wege und Kreise handelt, auf denen sich M- und M'-Kanten abwechseln. (S. Abb. 6.) Da $M \oplus M'$ mehr M'-Kanten

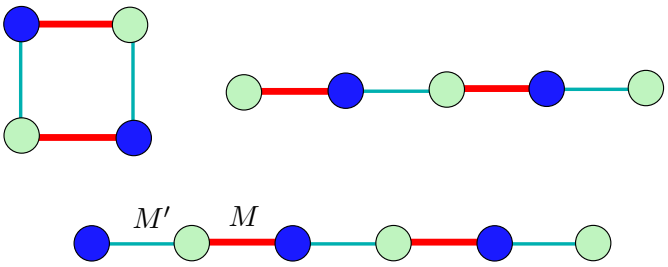

Abbildung 6: Schematische Darstellung von $M \oplus M'$. Kanten in M sind rot, Kanten in M' türkis und etwas dünner gezeichnet. Der Teilgraph $(U \cup W, M \oplus M')$ besteht aus Kreisen und einfachen Wegen, wobei mindestens einer dieser Wege mit einer M'-Kante beginnen und einer M'-Kante enden muss (unten). Dies ist ein matchingvergrößernder Weg bzgl. M.

als M-Kanten enthält, aber Kreise gleich viele M- und M'-Kanten enthalten, muss einer der Wege mehr M'-Kanten als M-Kanten enthalten. Das heißt, dass er in einem M-freien Knoten mit einer M'-Kante beginnt und in einem M-freien Knoten mit einer M'-Kante endet, also ein mvW ist. □

Ab jetzt sei $G = (U \cup W, E)$ bipartit. Der Anwendung der Ford-Fulkerson-Methode auf $\hat{G}$ entspricht folgendes Vorgehen: Gegeben M, suchen wir nach matchingvergrößernden Wegen. Wir beginnen an einem beliebigen freien Knoten $u_0 \in U$. Wenn u_0 nicht isoliert ist, können wir entlang einer Kante (u_0, w_1) zu einem Knoten $w_1 \in W$ gehen. Falls w_1 frei ist, haben wir einen mvW gefunden. Falls nicht, gibt es eine eindeutig bestimmte Kante $(w_1, u_1) \in M$. Nun gilt auch u_1 als gefunden. Wir gehen zu einem Nachbarn $w_2 \neq w_1$ von u_1, falls ein solcher existiert, und prüfen wieder, ob dieser frei ist. Falls ja, haben wir einen mvW gefunden. Falls nein, gibt es eine

[3] Im Fall von bipartiten Graphen könnte man auch die Entsprechung aus Prop. 2.3.1 benutzen und argumentieren, dass flussvergrößernde Wege im Restnetzwerk zu $\hat{G}$ und matchingvergrößernde Wege in G einander genau entsprechen. Dann folgt das Lemma aus dem MaxFlow-MinCut-Theorem. Wir geben hier aber einen einfachen direkten Beweis an, der auch für nicht bipartite Graphen gilt.

eindeutig bestimmte Matching-Kante (w_2, u_2), und wir betrachten u_2 als erreicht. Dieses Vorgehen wird iteriert, aber nicht nur entlang eines Weges, sondern von allen erreichten U-Knoten aus. Ein allgemeiner Iterationsschritt sieht also wie folgt aus: Wenn $U' \subseteq U$ die Menge der bisher gefundenen U-Knoten und $W' \subseteq W$ die Menge der bisher gefundenen W-Knoten ist, sucht man eine Kante (u, w), wobei $u \in U'$ und $w \in W - W'$ ein neuer, bisher unerreichter Knoten ist. Wenn w frei ist, bildet der alternierende Weg, entlang dessen man von u_0 zu w gekommen ist, einen matchingvergrößernden Weg; wenn w nicht frei ist, wird der eindeutig bestimmte Knoten $u' \in U$ mit $(w, u') \in M$ zu U' hinzugefügt. (Beachte: $u' \notin U'$, weil u' nicht frei ist, also $u' \neq u_0$, und weil alle anderen Knoten in U' immer gleichzeitig mit dem anderen Ende ihrer Matchingkante gefunden werden.) In dieser Weise fährt man fort, bis entweder ein freier Knoten auf der W-Seite gefunden wird oder bis alle Nicht-Matching-Kanten, die aus gefundenen Knoten $u \in U'$ herausführen, zu Knoten in W' führen. Im letzten

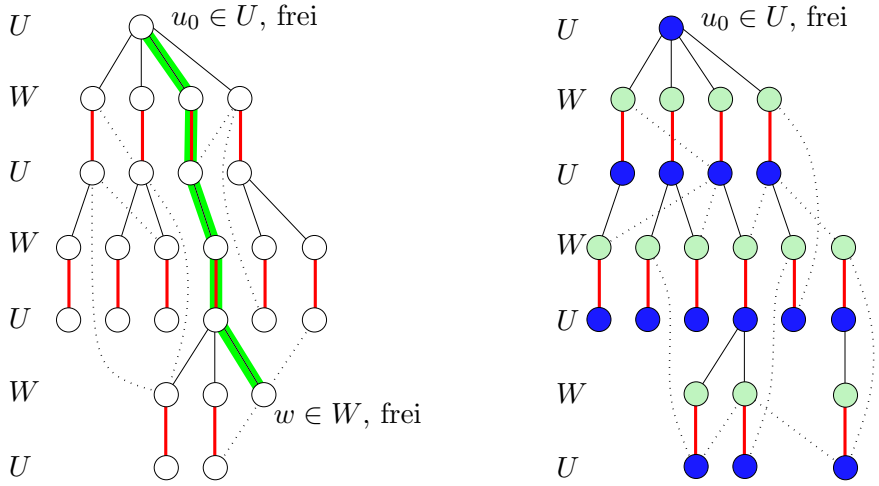

Abbildung 7: Links: Alternierender Baum mit matchingvergrößerndem Weg (grün). Rechts: Alternierender Baum ohne matchingvergrößernden Weg. Baumkanten sind schwarz/dünn (von U nach W) oder rot/dicker (Matchingkanten, von W nach U). Gestrichelte Kanten führen von U-Knoten zu schon entdeckten W-Knoten.

Fall muss man die gleiche Prozedur noch für die anderen freien Knoten in U durchführen. Es entsteht eine Menge von „alternierenden Bäumen“: Die Wurzeln sind frei, auf den Wegen wechseln sich Matchingkanten und Nichtmatchingkanten ab. Sobald ein freies $w \in W$ gefunden wird, kann man aufhören und das Matching vergrößern. Wenn das nie passiert, obwohl von allen freien Knoten in U gestartet wurde, dann gibt es keinen matchingvergrößernden Weg, also ist M maximal.

Bemerkung: Man kann diesen Ansatz („Konstruktion alternierender Bäume“) systematischer organisieren und straffen. Man setzt dazu

$F := \{u \in U \mid u \text{ ist frei}\}$ und

$N := \{u \in U \mid \text{ein Nachbar } w \in W \text{ von } u \text{ ist frei}\}$,

und definiert einen Hilfsgraphen (U, E_M) wie folgt: $(u, u') \in E_M$, wenn es einen Knoten $w \in W$ gibt, so dass $(u, w) \in E - M$ und $(w, u') = (u', w) \in M$. (Diese Definition zieht Vorwärtsgehen von u zu einem neuen Knoten $w \in W$ und das erzwungene Zurückgehen entlang einer eindeutig bestimmten Matchingkante (w, u') zu einem Schritt zusammen.) Wie man leicht sieht, ist dann die Suche nach einem matchingvergrößernden Weg in G äquivalent zur Suche nach einem Weg von F nach N in (U, E_M).

Wir können dann also folgendermaßen vorgehen: Ausgehend von M, berechne (U, E_M) und führe in diesem gerichteten Graphen eine Breitensuche durch, wobei nur Knoten in F als Wurzeln von BFS-Bäumen benutzt werden. (Siehe Vorlesung „Algorithmen und Datenstrukturen“, zu Breitensuche.) Notiere die Vorgänger der erreichten Knoten in ihren Breitensuchbäumen. Sobald ein Knoten $u' \in N$ gefunden wurde, kann der Weg von u' zu der zugehörigen Wurzel $u \in F$ rekonstruiert werden. Dieser Weg wird in einen matchingvergrößernden Weg umgerechnet, und das Matching M kann um eine Kante vergrößert werden.

Ausgehend vom leeren Matching $M_0 = \emptyset$ können höchstens $\frac{1}{2}n$ Matchingvergrößerungen durchgeführt werden. Wenn kein mvW mehr gefunden wird, ist das Matching M maximal. Jede Breitensuche kostet Zeit $O(m)$, für $m = |E|$. Die Gesamtlaufzeit des resultierenden Algorithmus ist $O(nm)$. (Wir kommen später, im Abschnitt über gewichtete Matchings, auf diesen Ansatz zurück.)

2.3.2 Matchings über Sperrflüsse

Die Sperrflussmethode mit der Sperrflussberechnung (z. B. nach Dinitz) führt auf den ersten Blick zu einem Matchingalgorithmus mit Laufzeit $O(n^2m)$ (bzw. $O(n^3)$) bei Knotenzahl n und Kantenzahl m. Im Folgenden wollen wir sehen, dass die Sperrflussmethode in Wirklichkeit zu deutlich besseren Laufzeitschranken führt – sogar besser als $O(nm)$.

Definition 2.3.4. *Ein Flussnetzwerk $G = (V, E, c)$ heißt **0-1-Netzwerk**, wenn alle Kantenkapazitäten den Wert 1 haben. (Nicht vorhandene Kanten haben Kapazität 0.) G heißt **einfaches Netzwerk**, wenn G 0-1-Netzwerk ist und jeder Knoten $v \in V - \{q, s\}$ Eingangsgrad 1 oder Ausgangsgrad 1 hat.*

Beispiel: Das oben zu einem bipartiten Graphen G definierte Flussnetzwerk $\hat{G}$ ist ein einfaches Netzwerk, weil jeder Knoten in U Eingangsgrad 1 hat und jeder Knoten in W Ausgangsgrad 1 hat.

Lemma 2.3.5. *Wenn G ein einfaches Netzwerk ist und f ein ganzzahliger Fluss in G ist, dann ist auch das Restnetzwerk G_f ein einfaches Netzwerk.*

Beweis: (Vgl. Abb. 8.) Wir betrachten einen Knoten v, in den genau eine Kante (u, v) hineinführt. (Knoten v mit Ausgangsgrad 1 behandelt man analog.)

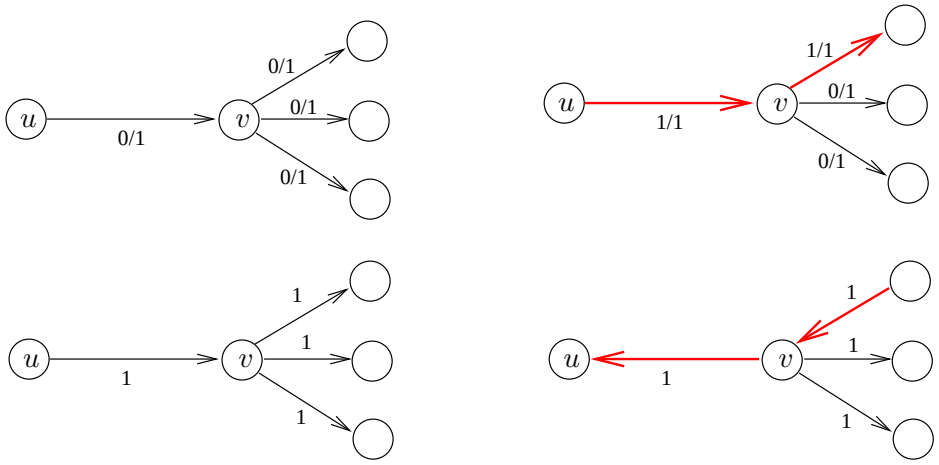

Abbildung 8: Links: Knoten v in einfachem Netzwerk mit $f_{\text{In}}(v) = f_{\text{Out}}(v) = 0$. Auch im Restnetzwerk bleibt der Eingangsgrad 1. Rechts: Knoten in einfachem Netzwerk mit $f_{\text{In}}(v) = f_{\text{Out}}(v) = 1$. Auch im Restnetzwerk ist der Eingangsgrad 1, obgleich zwei Rückwärtskanten auftreten.

1. Fall: $f(u, v) = 0$. In diesem Fall liegt auch auf den Kanten, die aus v hinausführen, Fluss 0. Die Restkapazität auf all diesen Kanten ist also 1; die Kanten in Rückwärtsrichtung haben Restkapazität 0, sind also nicht vorhanden.

2. Fall: $f(u, v) = 1$. In diesem Fall gilt $\text{rest}(u, v) = 0$ und $\text{rest}(v, u) = 1$, in G_f führt also (v, u) aus v hinaus. Es gibt genau eine Kante (v, w) mit $f(v, w) = 1$, für die also $\text{rest}(w, v) = 1$ und $\text{rest}(v, w) = 0$ gilt. Alle anderen Ausgangskanten haben Fluss 0, also auch Restkapazität 1. Also hat auch im Restnetzwerk G_f Knoten v Eingangsgrad 1. □

In einfachen Netzwerken ist es nicht möglich, dass ein ganzzahliger Fluss einen großen Wert hat und gleichzeitig der Abstand zwischen q und s groß ist. Dahinter steckt, dass sich in diesen Netzwerken ganzzahlige Flüsse in knotendisjunkte Flüsse zerlegen lassen. (Diese Tatsache werden wir später auf ein Restnetzwerk anwenden.)

Lemma 2.3.6. *Sei $G = (V, E)$ ein einfaches Netzwerk und sei f ein ganzzahliger Fluss in G mit Wert $w(f)$. Wenn L der Abstand von q nach s in G ist, gilt:*

$$(L - 1) \cdot w(f) \leq |V| - 2.$$

Beweis: Wir betrachten die Kantenmenge $E_{f=1}$, die aus den Kanten e mit $f(e) = 1$ besteht. Weil G einfaches Netzwerk ist, und wegen der Kirchhoff-Regel, gibt es für Knoten $v \in V - \{q, s\}$ nur zwei Fälle: v hat keine Kante in $E_{f=1}$ oder v hat in $E_{f=1}$ Eingangsgrad 1 und Ausgangsgrad 1. Aus solchen Knoten und inzidenten Kanten kann man nur einfache Wege bauen, die bei q beginnen und bei s enden, und Kreise, die weder q noch s berühren. Solche Kreise können wir einfach weglassen

(d. h., wir können den Flusswert auf den Kanten auf 0 setzen), ohne dass sich der Flusswert ändert. Es bleiben $w(f)$ knotendisjunkte Wege von q nach s übrig. Auf jedem dieser Wege liegen außer q und s noch mindestens $L-1$ andere Knoten. Wegen der Disjunktheit gilt: $|V - \{q, s\}| \geq (L-1) \cdot w(f)$, wie behauptet. □

Lemma 2.3.7. *Sei $G = (V, E)$ ein beliebiges Flussnetzwerk und sei f Fluss in G. Sei w^* der maximale Wert eines Flusses in G. Dann existiert im Restnetzwerk G_f ein Fluss f_0 mit Wert $w(f_0) = w^* - w(f)$, so dass $f + f_0$ maximaler Fluss ist.*

Beweis: (Siehe Abb. 9.) Sei f^* ein maximaler Fluss in G, mit $w(f^*) = w^*$. Wir definieren, für $(u, v) \in E$:

$$f_0(u,v) := \begin{cases} f^*(u,v) - f(u,v) & \text{, falls } f(u,v) < f^*(u,v) \\ 0 & \text{, sonst,} \end{cases}$$

und

$$f_0(v,u) := \begin{cases} f(u,v) - f^*(u,v) & \text{, falls } f(u,v) > f^*(u,v) \\ 0 & \text{, sonst,} \end{cases}$$

Man kontrolliert leicht nach, dass f_0 ein Fluss im Restnetzwerk ist, der $f + f_0 = f^*$ und $w(f) + w(f_0) = w(f^*) = w^*$ erfüllt. □

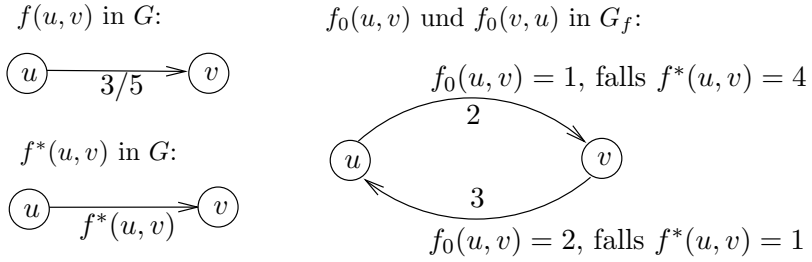

Abbildung 9: Links oben: Kante mit Fluss $f(u, v) = 3$ und Kapazität $c(u, v) = 5$. Links unten: Kante mit „Wunschfluss“ $f^*(u, v)$. Rechts: Kanten im RNW G_f mit Restkapazitäten. Je nachdem ob $f^*(u, v) > f(u, v)$ oder $f^*(u, v) < f(u, v)$ gilt, wird in G_f Fluss $f_0(u, v)$ oder $f_0(v, u)$ auf einen positiven Wert gesetzt.

Wir erinnern uns an die Sperrflussmethode (die hier auf ein einfaches Netzwerk angewendet werden soll und nur ganzzahlige Sperrflüsse benutzt).

Algorithmus 2.3.8 (**(Ganzzahlige) Sperrfluss-Methode**).
INPUT: Flussnetzwerk $G = (V, E, q, s, c)$, c ganzzahlig.

METHODE:
1 $f :=$ der Nullfluss;
2 **repeat**
3 Berechne das Restnetzwerk G_f und das Niveaunetzwerk G'_f;
4 Falls dabei in der ersten BFS s nicht erreicht: **return** f;
5 Konstruiere einen (ganzzahligen) Sperrfluss φ in G'_f;
6 $f := f + \varphi$;

Ohne irgendwelche Modifikationen läuft dieses Verfahren in einfachen Netzwerken besonders schnell.

Satz 2.3.9. *Sei G ein einfaches Netzwerk. Dann gilt, wenn Algorithmus 2.3.8 ausgeführt wird:*

(a) *Das Ergebnis ist ein maximaler Fluss.*

(b) *Der Algorithmus führt maximal $2(\lfloor\sqrt{n}\rfloor + 1)$ Runden aus.*

Beweis: (a) folgt direkt aus der Korrektheit der Sperrflussmethode. Wir beweisen (b). Sei w^* der Wert eines maximalen Flusses. Es ist klar, dass w^* und alle zwischendurch berechneten Flusswerte $w(f)$ ganze Zahlen sind.

Wir teilen die Runden in Algorithmus 2.3.8 in zwei Typen ein, je nach der Situation in Zeile 3. Man unterscheidet, ob für f, den bisher berechneten Fluss, der Wert $w(f)$ noch „weit von w^* entfernt“ ist oder nicht.

Typ 1: $w(f) < w^* - \sqrt{n}$. – Es sei L der Abstand von q nach s im Restnetzwerk G_f, also die Tiefe des Niveaunetzwerks G'_f. Nach Lemma 2.3.7 gibt es einen (ganzzahligen) Fluss f_0 im Restnetzwerk G_f mit Wert $w(f_0) = w^* - w(f)$, also $w(f_0) > \sqrt{n}$. Aus Lemma 2.3.5 wissen wir, dass auch G_f ein einfaches Netzwerk ist. Mit Lemma 2.3.6 folgt $(L-1)w(f_0) < |V| = n$. Mit $w(f_0) > \sqrt{n}$ folgt: $L < \sqrt{n} + 1$. (Intuitiv heißt das: Wenn der Abstand von $w(f)$ zum Zielwert noch „groß“ ist, dann hat das Niveaunetzwerk eher geringe Tiefe.) Im Beweis von Lemma 1.4.4 haben wir gesehen, dass bei der Sperrflussmethode die Tiefe des Niveaunetzwerks in jeder Runde strikt zunimmt. In der ersten Runde ist diese Tiefe mindestens 1. Daher kann es nicht mehr als $\lfloor\sqrt{n}\rfloor + 1$ Runden vom Typ 1 geben.

Typ 2: $w(f) \geq w^* - \sqrt{n}$. – Da mit jeder Runde der Flusswert um mindestens 1 wächst, gibt es maximal $\lfloor\sqrt{n}\rfloor + 1$ Runden vom Typ 2. □

Beobachtung 2.3.10. *In einem einfachen (Niveau-)Netzwerk $G'_f = (V', E')$ kann ein Sperrfluss in Zeit $O(|E'|)$ berechnet werden.*

(Idee: Führe eine Tiefensuche von q aus durch. Sobald s erreicht wird, kann man den gesamten aktiven Weg von q nach s aus dem Graphen streichen, inklusive der ausgehenden Kanten. Auf diese Weise wird jede Kante maximal einmal betrachtet.)

Satz 2.3.11 (Hopcroft-Karp)**.** (*Kardinalitäts-*)*Maximale Matchings in bipartiten Graphen bzw. maximale Flüsse in einfachen Netzwerken können in Zeit* $O(\sqrt{n} \cdot m)$ *berechnet werden, wobei* n *die Knotenzahl und* m *die Kantenzahl ist.*

(Bemerkung: In seiner Originalversion benutzt der Algorithmus von Hopcroft und Karp matchingvergrößernde Wege. In jeder Runde wird eine nicht erweiterbare Menge von kürzestmöglichen matchingvergrößernden Wegen berechnet und das Matching mit allen diesen Wegen erweitert. Genaues Hinsehen zeigt, dass dies genau der Verwendung eines Sperrflusses im Niveaunetzwerk entspricht. Details: Übung.)

2.4 Das Zuordnungsproblem und die Ungarische Methode

Wir verallgemeinern das Problem, ein kardinalitätsmaximales Matching zu finden, auf bipartite Graphen mit Kantengewichten. Zu einem bipartiten Graphen $G = (U \cup W, E)$ mit nichtnegativen Kantengewichten $c\colon E \to \mathbb{R}_0^+$ wollen wir ein Matching $M \subseteq E$ finden, das den Wert $c(M) = \sum_{e \in M} c(e)$ maximiert. Auch für dieses Problem gibt es effiziente Algorithmen, wie wir sehen werden. Der Auktionsalgorithmus aus Abschnitt 2.2 kann benutzt werden, wenn die Gewichte ganzzahlig sind. Allerdings kommt in der Laufzeitschranke das größte Gewicht vor. Wir suchen nach einem Algorithmus, dessen Laufzeit nur von $|U \cup W|$ abhängt.

Die Notation wird einfacher, wenn die disjunkten Mengen U und W die gleiche Größe haben und wenn $E = U \times W$ ist. Dazu fügen wir gegebenenfalls Knoten hinzu, so dass $|U| = |W| = n$ wird, und ergänzen fehlende Kanten mit Gewicht 0. Es entsteht ein vollständiger bipartiter Graph auf der Knotenmenge $V = U \cup W$ mit $2n$ Knoten. In diesem Graphen werden nur noch *perfekte* Matchings gesucht. (Weil die Kosten von „echten“ Kanten nichtnegativ sind, lassen sich maximale nicht-perfekte Matchings durch Hinzunehmen einiger neuer Kanten mit Kosten 0 zu einem maximalen perfekten Matching mit denselben Kosten ergänzen.)

Es ergibt sich das **„Zuordnungsproblem“** (engl.: *assignment problem*).

Max-Zuordnungsproblem:
Seien U und W Mengen mit $|U| = |W| = n$. Gegeben Zahlen („Gewichte“) $c_{uw} = c(u, w) \in \mathbb{R}_0^+$, $u \in U, w \in W$, finde ein perfektes Matching M in $E = U \times W$, so dass

$$c(M) = \sum_{(u,w) \in M} c(u, w)$$

möglichst groß ist.

Beispiel: n Personen sollen n Jobs zugeordnet werden. Wenn Person u Job w ausführt, entsteht (pro Stunde) Wert c_{uw}. Wie soll man Personen und Jobs einander zuordnen, um den gesamten erzeugten Wert zu maximieren?

	Spülen	Putzen	Kochen	Bügeln
Anton	$\underline{1}$	2	0	5
Berti	0	$\underline{3}$	4	1
Conni	3	1	2	$\underline{6}$
Det	2	0	$\underline{3}$	4

(Ist die durch Unterstreichung angedeutete Zuordnung optimal?)

Es ist leicht zu sehen, dass sich das Problem nicht ändert, wenn Kantengewichte auch negativ sein können (ersetze „$c_{uw} \in \mathbb{R}_0^+$“ durch „$c_{uw} \in \mathbb{R}$“). Man addiert einfach $C = -\min\{c_{uw} \mid u \in U, w \in W\}$ zu allen Kantengewichten und erhält eine äquivalente Instanz mit nichtnegativen Einträgen.

Natürlich gibt es auch ein entsprechendes Minimierungsproblem:

Beispiel: n Personen sollen n Jobs zugeordnet werden. Wenn Person u Job w ausführt, fallen (pro Stunde) Kosten c_{uw} an. Wie soll man Personen und Jobs einander zuordnen, um die Gesamtkosten zu *minimieren*?

Min-Zuordnungsproblem:
Seien U und W Mengen mit $|U| = |W| = n$. Gegeben Zahlen $c_{uw} = c(u, w) \in \mathbb{R}$ (ohne Vorzeichenbeschränkung), für $u \in U, w \in W$, finde ein perfektes Matching M in $E = U \times W$, so dass

$$c(M) = \sum_{(u,w)\in M} c_{uw}$$

möglichst klein ist.

Aus einem Minimierungsproblem wird ein äquivalentes Maximierungsproblem, wenn man alle Gewichte mit -1 multipliziert. Ab hier betrachten wir nur noch das Max-Zuordnungsproblem, und nennen es das ***Zuordnungsproblem***. Außerdem legen wir die Bezeichnung $V = U \cup W$ fest.

Der Ansatz für die Lösung ist, mit dem leeren Matching $M = \emptyset$ zu beginnen und in n Runden die Kantenzahl im Matching M zu erhöhen, bis ein perfektes Matching gefunden ist, das das Gesamtgewicht maximiert. Dazu findet man in jeder Runde einen matchingvergrößernden Weg in einem bipartiten **Hilfsgraphen G_ℓ** mit Knotenmenge $U \cup W$ (ohne Kantengewichte!) und vergrößert M entlang dieses Weges, ähnlich wie in Abschnitt 2.3.1. Wenn M nicht perfekt ist, aber G_ℓ keinen matchingvergrößernden Weg mehr hat, kann man den Hilfsgraphen so modifizieren, dass sich das Matching erweitern lässt. Wenn schließlich ein perfektes Matching erreicht ist, erweist sich dieses als optimal. Es sind natürlich noch viele Details zu klären.

Der zentrale Trick liegt darin, den Knoten ***Werte*** oder ***Markierungen*** zu geben:

Definition 2.4.1. *Eine Funktion $\ell\colon V \to \mathbb{R}$ heißt eine* ***zulässige Markierung*** *für den Gewichtssatz $(c_{uw})_{u\in U, w\in W}$, falls*

$$c_{uw} \leq \ell(u) + \ell(w)$$

gilt, für alle $u \in U$, $w \in W$.

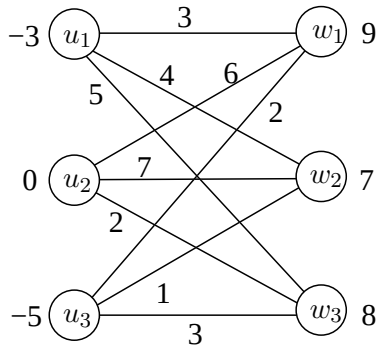

Abbildung 10: Ein vollständiger bipartiter Graph mit Kantengewichten und einer zulässigen Markierung.

Definition 2.4.2. *Sei $\ell\colon V \to \mathbb{R}$ zulässige Markierung. Dann ist der* ***Gleichheitsgraph*** *$G_\ell = (U \cup W, E_\ell)$ durch die Kantenmenge*

$$E_\ell := \{(u, w) \mid c_{uw} = \ell(u) + \ell(w)\}$$

definiert.

Zulässige Markierungen ℓ, der Hilfsgraph G_ℓ und Matchings M in G_ℓ werden verschränkt entwickelt.

Wir können eine kombinatorische Bedingung dafür angeben, dass ein Matching im Gleichheitsgraphen das Zuordnungsproblem löst: *jedes* perfekte Matching in G_ℓ ist geeignet!

Satz 2.4.3 (Kuhn/Munkres)**.** *Sei $\ell\colon V \to \mathbb{R}$ zulässige Markierung und sei $M \subseteq E_\ell$ ein* ***perfektes*** *Matching. Dann ist M optimal, d. h. der Wert $c(M)$ ist größtmöglich unter allen perfekten Matchings über $U \times W$.*

Beweis: Sei M' ein beliebiges perfektes Matching in $E = U \times W$. (Keine Beschränkung

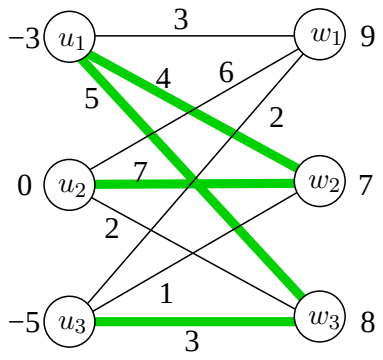

Abbildung 11: Ein vollständiger bipartiter Graph mit Kantengewichten und einer zulässigen Markierung ℓ. Die Kanten des Gleichheitsgraphen sind (grün) hervorgehoben.

auf E_ℓ!) Dann gilt:

$$\begin{aligned} c(M') &= \sum_{(u,w)\in M'} c(u,w) \\ &\overset{(1)}{\leq} \sum_{(u,w)\in M'} (\ell(u)+\ell(w)) \\ &\overset{(2)}{=} \sum_{v\in V} \ell(v) \\ &\overset{(3)}{=} \sum_{(u,w)\in M} (\ell(u)+\ell(w)) \\ &\overset{(4)}{=} \sum_{(u,w)\in M} c(u,w) \\ &= c(M). \end{aligned}$$

((1) gilt, weil ℓ zulässig ist; (2), weil M' perfekt ist; (3) gilt, weil M perfekt ist, und (4), weil $M \subseteq E_\ell$ ist.) □

Damit ist unser Ziel für den Rest dieses Abschnittes klar: wir entwerfen einen Algorithmus, der zugleich eine Markierung ℓ und ein perfektes Matching $M \subseteq E_\ell$ aufbaut.

Wir geben zunächst in Form eines Flussdiagramms den allgemeinen Plan des Algorithmus an (Abb. 12). Die Details werden im Weiteren genauer ausgeführt.

Der Beginn ist einfach: Wir setzen $M = \emptyset$ und $\ell(u) = 0$ für $u \in U$ und $\ell(w) = \max\{c_{uw} \mid u \in U\}$ für $w \in W$. Dann ist klar, dass ℓ zulässig ist und dass $M \subseteq E_\ell$ ist.

Nun nehmen wir an, dass ℓ und $M \subseteq E_\ell$ gegeben sind, wobei M (noch) nicht perfekt ist. Wir entwickeln die Situation weiter, indem wir in $G_\ell = (V, E_\ell)$ nach einem matchingvergrößernden Weg suchen. Dies geschieht systematisch, durch Aufbau eines

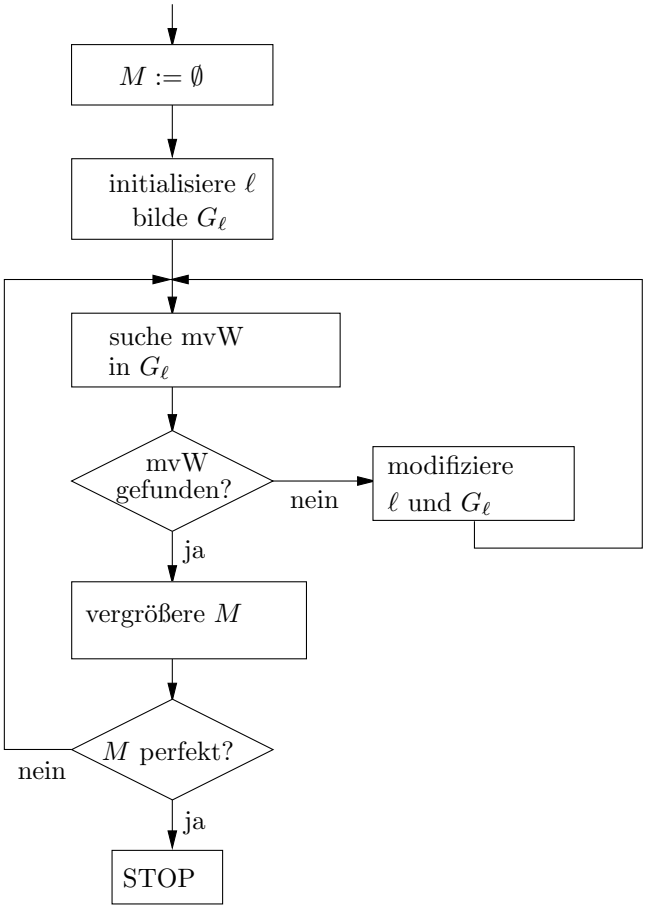

Abbildung 12: Ungarische Methode: Der Plan

alternierenden Baums B im bipartiten Graphen G_ℓ (Abbildungen 13 und 14).

Definition 2.4.4. *Sei* $G_\ell = (V, E_\ell)$ *sowie ein Matching* $M \subseteq E_\ell$ *gegeben. Ein* ***alternierender Baum*** B *besteht aus einem (von* r *weg gerichteten) Baum* B *mit Wurzel* $r \in U$*, wobei Folgendes gilt*:

(i) r *ist frei*;

(ii) *wenn* (u, w) *mit* $u \in U$*,* $w \in W$ *gerichtete Kante in* B *ist, dann ist entweder* w *frei oder es gibt ein* $u' \in U$*, so dass* $(u', w) \in M$ *ist und* (w, u') *Kante in* B *ist.*

Die Menge der Knoten in B*, die in* U *liegen, heißt* S*, die Menge der Knoten in* B*, die in* W *liegen, heißt* T*.*

Wenn G_ℓ und ein nicht perfektes Matching $M \subseteq E_\ell$ gegeben sind, lässt sich ein solcher alternierender Baum ganz leicht aufbauen. Man beginnt mit einem beliebigen

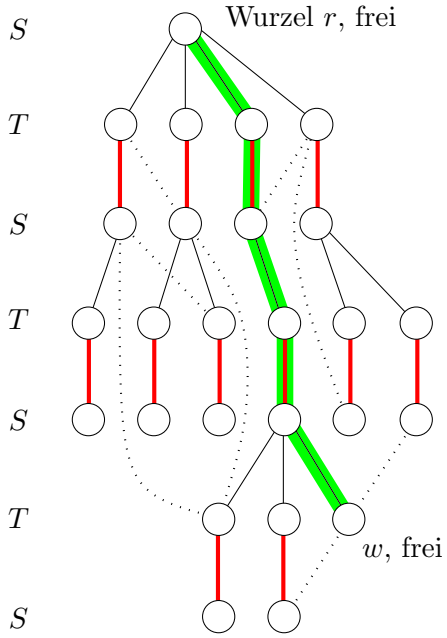

Abbildung 13: Ein (eventuell unvollständiger) alternierender Baum, der einen matchingvergrößernden Weg enthält (grün/schraffiert). Baumkanten sind schwarz/dünn (von S nach T) oder rot/dicker (Matchingkanten, von T nach S). Kanten in E_ℓ, die nicht zum Baum gehören, sind gepunktet dargestellt.

freien Knoten $r \in U$ als Wurzel, und setzt $S := \{r\}$, $T := \emptyset$. Dann iteriert man die folgende Aktion:

> Finde einen Baumknoten $u \in S$, so dass es eine Kante $(u, w) \in E_\ell$ mit $w \in W - T$ gibt. Füge (u, w) zu B hinzu; füge w zu T hinzu.
>
> **1. Fall:** Wenn w frei ist, breche die Konstruktion ab und melde den Weg in B von r zu w als matchingvergrößernden Weg.
>
> **2. Fall:** Wenn w nicht frei ist, dann gibt es einen (eindeutig bestimmten) Knoten $u' \in U$, so dass $(u', w) \in M$ ist. Füge die Kante (w, u') zu B hinzu; füge u' zu S hinzu.

Man beachte, dass in der Konstruktion immer beide Endpunkte einer Matchingkante zusammen entdeckt und in den Baum eingebaut werden. Insbesondere muss der Knoten u' im 2. Fall auch neu sein, weil w neu ist.

Es gibt zwei Möglichkeiten, wie dieser Prozess anhalten kann. Wenn ein freier Knoten $w \in T$ gefunden wird, können wir M anhand des matchingvergrößernden Weges vergrößern (1. Fall, Abb. 13). (Danach wird ein völlig neuer alternierender Baum aufgebaut.) – Der Prozess kann aber auch stoppen, wenn alle $w \in W$, für die es

$u \in S$ und eine Kante $(u, w) \in E_\ell$ gibt, schon in T sind (Abb. 14). In dieser Situation können wir ℓ so verändern, dass der alternierende Baum weitergebaut werden kann. Das wird getan, und nach endlich vielen solchen Modifikationen muss einmal der 1. Fall eintreten.

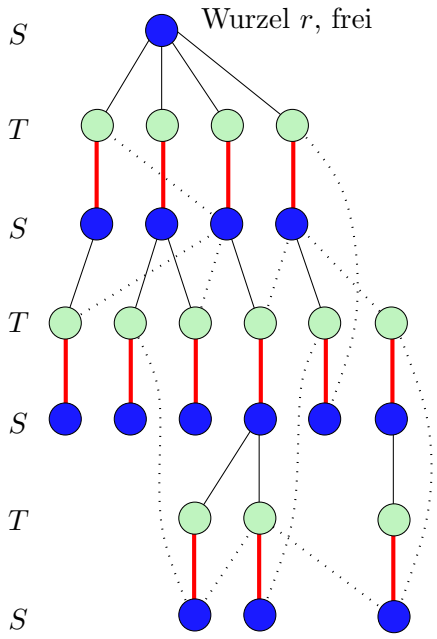

Abbildung 14: Ein alternierender Baum, dessen Konstruktion endet, weil alle Kanten aus S-Knoten zu schon bekannten (im Bild enthaltenen) T-Knoten führen. Die Knotenmenge S ist (dunkel)blau, die Menge T (hell)grün dargestellt. Weil die Wurzel frei ist und alle anderen Knoten paarweise auf Matchingkanten liegen, gilt $|S| = |T| + 1$.

Nehmen wir also an, der Aufbau von B endet, weil alle E_ℓ-Kanten aus S zu Knoten in T führen. Weil die Wurzel r frei ist und alle anderen Knoten in S und T durch die Matchingkanten in B einander paarweise zugeordnet sind, gilt $|S| = |T| + 1$. Insbesondere ist $T \subsetneq W$, und T ist exakt die Menge aller E_ℓ-Nachbarn von Knoten in S.

Bemerkung 2.4.5. In dieser Situation können wir noch Folgendes beobachten:

(a) Wenn $(u, w) \in E_\ell$ eine beliebige Kante ist, dann kann nicht zugleich $u \in S$ und $w \notin T$ sein. Jede Kante in E_ℓ enthält also einen Knoten aus $(U - S) \cup T$ (wir sagen: $(U - S) \cup T$ *überdeckt* E_ℓ).

(b) Wir können in dieser Situation sogar folgern, dass es in $G_\ell = (V, E_\ell)$ kein perfektes Matching geben kann.[4] Wieso? Es gilt $|(U - S) \cup T| = n - |S| + |T| = n - 1$. Daher kann ein Matching M in E_ℓ nicht mehr als $n - 1$ Kanten haben, da $(U - S) \cup T$ erst recht jede der disjunkten Kanten in M trifft.

[4]Es folgt nicht unbedingt, dass M nicht erweiterbar ist. Dies ist hier aber nicht von Bedeutung.

Wenn die Konstruktion des alternierenden Baums B von r aus zu Fall 2 geführt hat, verändern wir ℓ zu einer neuen Markierung (die dann wieder ℓ heißt), wie folgt:

Definiere

$$\alpha := \alpha_\ell(S) := \min\{\ell(u) + \ell(w) - c_{uw} \mid u \in S, w \in W - T\}.$$

Wegen Bemerkung 2.4.5(a) und der Definition des Gleichheitsgraphen sind alle Zahlen, über die hier das Minimum gebildet wird, positiv. Also ist $\alpha > 0$. Weiter gibt es ein Paar $(u, w) \in S \times (W - T)$ mit $\alpha = \ell(u) + \ell(w) - c_{uw}$. Wir definieren:

$$\begin{aligned}
\ell'(u) &:= \ell(u) - \alpha, \text{ für } u \in S;\\
\ell'(w) &:= \ell(w) + \alpha, \text{ für } w \in T;\\
\ell'(v) &:= \ell(v), \text{ für } v \notin S \cup T.
\end{aligned}$$

Das nächste Lemma besagt, dass mit dieser Konstruktion eine neue zulässige Markierung ℓ' entsteht, so dass M und der alternierende Baum B weiter zu ℓ' passen und B erweitert werden kann.

Lemma 2.4.6. *In der eben beschriebenen Situation ist ℓ' wieder zulässig, und es gilt*:

(i) $E_\ell \cap (S \times T) \subseteq E_{\ell'}$.
(*Damit liegen alle Kanten von B in $E_{\ell'}$.*)

(ii) $E_\ell \cap ((U - S) \times (W - T)) \subseteq E_{\ell'}$.
(*Damit liegen auch alle die Kanten von M in $E_{\ell'}$, die B nicht berühren.*)

(iii) *Es gibt mindestens eine Kante $(u, w) \in E_{\ell'}$ mit $u \in S$ und $w \notin T$ (die nicht in E_ℓ war).*

Beweis: Wir betrachten vier Arten von Paaren $(u, w) \in U \times W$.

(a) $u \in S, w \in T$: Es gilt:

$$\ell'(u) + \ell'(w) = (\ell(u) - \alpha) + (\ell(w) + \alpha) = \ell(u) + \ell(w) \geq c_{uw}.$$

Wenn $(u, w) \in E_\ell$, dann gilt auch $\ell'(u) + \ell'(w) = c_{uw}$, also $(u, w) \in E_{\ell'}$.

(b) $u \notin S, w \notin T$: Es gilt trivialerweise $\ell'(u) + \ell'(w) = \ell(u) + \ell(w) \geq c_{uw}$.
Wenn $(u, w) \in E_\ell$ gilt, dann auch $(u, w) \in E_{\ell'}$.

(c) $u \in S, w \notin T$: Nach Definition von α gilt

$$\ell'(u) + \ell'(w) = (\ell(u) - \alpha) + \ell(w) \geq c_{uw}.$$

Für jedes Paar (u, w) mit $\alpha = \ell(u) + \ell(w) - c_{uw}$ gilt $\ell'(u) + \ell'(w) = c_{uw}$. Hiervon gibt es mindestens eines, woraus (iii) folgt.

(d) $u \notin S, w \in T$: Es gilt:

$$\ell'(u) + \ell'(w) = \ell(u) + (\ell(w) + \alpha) > \ell(u) + \ell(w) \geq c_{uw}.$$

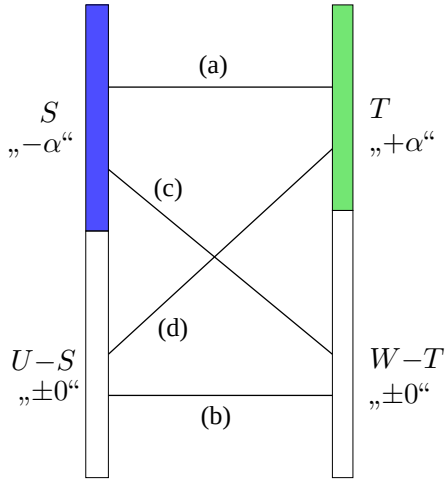

Abbildung 15: Vier Arten von Paaren (u, w) bei der Konstruktion von ℓ' und $E_{\ell'}$. Kanten vom Typ (c) und (d) können nicht in M liegen.

□

Beispiel: In Abb. 16 findet eine Aktualisierung von ℓ statt, die zu einer Änderung des Gleichheitsgraphen führt. Danach kann der alternierende Baum so erweitert werden, dass ein matchingvergrößernder Weg entsteht.

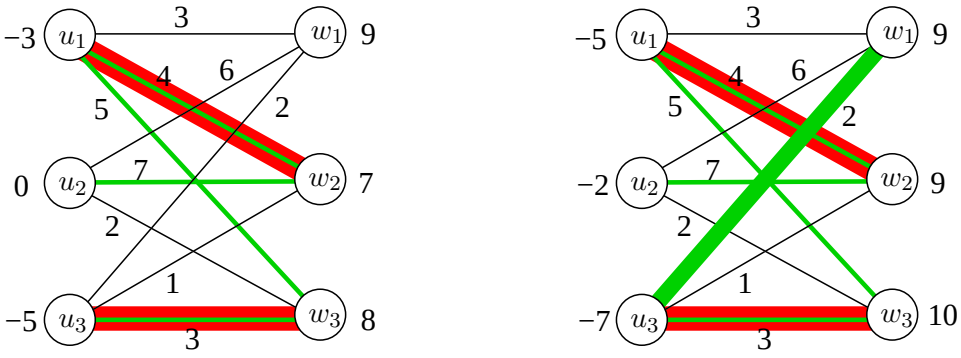

Abbildung 16: Links: Die Konstruktion des alternierenden Baums B von $r = u_2$ aus endet mit $S = \{u_1, u_2, u_3\}$ und $T = \{w_2, w_3\}$. Die Berechnung von α ergibt 2. – Rechts: Nach der Änderung der Markierungen ist (u_3, w_1) neue Kante im Gleichheitsgraphen. Nun findet sich auch ein matchingvergrößernder Weg $(u_2, w_2, u_1, w_3, u_3, w_1)$.

Nach der Änderung von ℓ auf ℓ' benennen wir ℓ' in ℓ um. Aus Lemma 2.4.6 folgt, dass alle Kanten, die für M und für B relevant sind, nach wie vor in E_ℓ sind, dass aber mindestens eine neue Kante $(u, w) \in S \times (W - T)$ dazugekommen ist, die es

erlaubt, B zu vergrößern. Wir bauen B weiter, bis ein matchingvergrößernder Weg gefunden wird oder bis neue, größere Mengen S und T gefunden wurden, über die B nicht hinausgeht. In diesem Fall kann ℓ erneut modifiziert werden, um B wieder zu vergrößern.

Insgesamt führt dies zu Algorithmus 2.4.10[5].

Satz 2.4.7. *Die Ungarische Methode (Algorithmus 2.4.10) löst das Zuordnungsproblem.*

Beweis: Der Algorithmus konstruiert eine zulässige Markierung $\ell\colon V \to \mathbb{R}$ und ein perfektes Matching M in G_ℓ. Nach Satz 2.4.3 ist $c(M)$ maximal. □

Satz 2.4.8. *Die Ungarische Methode (Algorithmus 2.4.10) kann so implementiert werden, dass sie Laufzeit $O(n^4)$ hat.*

Beweis: Es gibt genau n Matchingvergrößerungen, um von 0 auf n Kanten zu kommen. Für eine Matchingvergrößerung muss ein alternierender Baum aufgebaut, eventuell die Funktion ℓ mehrfach verändert und der Baum erweitert werden. Da $|S| = |T|+1$ dadurch wächst, gibt es höchstens n Änderungen von ℓ. Eine solche Änderung und die anschließende Erweiterung von B ist ohne Weiteres in Zeit $O(n^2)$ durchführbar. □

Bemerkung 2.4.9. Mit etwas mehr Sorgfalt kann die Laufzeit sogar auf $O(n^3)$ reduziert werden. Hierzu benötigt man etwas aufwendigere Datenstrukturen, die es erlauben, zu einem Knoten $u \in S$ genau die bisher noch nicht betrachteten Kanten (u, w) des Gleichheitsgraphen zu finden, so dass beim Aufbau und Weiterbau von B nicht wiederholt alle $w \in W$ betrachtet werden müssen. Wir lassen die Details weg.

[5] publiziert von H. Kuhn (1955). Er nannte das Verfahren „Ungarische Methode“, weil sich der Algorithmus auf die Arbeit zweier ungarischer Mathematiker stützte, nämlich Dénes Kőnig and Jenő Egerváry.

Algorithmus 2.4.10 (**Ungarische Methode**)**.**
// Löst das Zuordnungsproblem in vollständigen bipartiten Graphen
INPUT: Kostenmatrix $(c_{uw})_{u\in U, w\in W}$ mit $c_{uw} \in \mathbb{R}$, für $|U| = |W| = n$;

INITIALISIERUNG:
1 $\ell(u) \leftarrow 0$ für alle $u \in U$;
2 $\ell(w) \leftarrow \max\{c_{uw} \mid u \in U\}$ für alle $w \in W$;
3 $M \leftarrow \emptyset$;
ITERATION:
4 **while** $|M| < n$ **repeat**
5 // vergrößere M:
6 wähle freies $r \in U$;
7 $S \leftarrow \{r\}$; $T \leftarrow \emptyset$;
8 stecke r in neue, leere Warteschlange Q;
9 **repeat** // B aufbauen, ggfls. ℓ anpassen (wiederholt)
10 entnehme u aus Q;
11 prüfe Kanten $(u, w) \in E_\ell$ (d. h. mit $\ell(u) + \ell(w) = c_{uw}$) nacheinander:
12 **if** $w \notin T$ **then**
13 füge w zu T hinzu; füge (u, w) zu B hinzu;
14 **if** es gibt eine Kante $(u', w) \in M$
15 **then**
16 füge u' zu S hinzu; füge (w, u') zu B hinzu;
17 füge u' in Q ein;
18 **else** // $w \in W$ ist frei
19 bestimme den mv Weg p von r nach w in B; springe zu Zeile 28;
20 **if** Q ist leer // B kann mit aktuellem E_ℓ nicht erweitert werden
21 **then** // passe ℓ an
22 $\alpha := \min\{\ell(u) + \ell(w) - c_{uw} \mid u \in S, w \in W - T\}$;
23 füge alle $u \in S$ mit $(\exists w \in W - T\colon \ell(u) + \ell(w) - c_{uw} = \alpha)$ in Q ein;
24 **for** $u \in S$ **do** $\ell(u) \leftarrow \ell(u) - \alpha$;
25 **for** $w \in T$ **do** $\ell(w) \leftarrow \ell(w) + \alpha$;
26 // baue B weiter auf: weiter bei Zeile 9
27 // **Ende repeat aus Zeile 9**
28 // hier ist p ein mv Weg in G_ℓ von r zu einem $w \in W$
29 vergrößere M mit p zu M' mit $|M'| = |M| + 1$;
30 $M \leftarrow M'$;
31 // **Ende while aus Zeile 4**
32 Ausgabe: M.

2.5 Stabile Paarungen

Wir stellen uns folgende Situation bei einer Praktikumsbörse vor: n Studierende suchen einen Praktikumsplatz; Firmen bieten n solche Plätze an. Jeder Studierende hat ein eigenes Ranking für die Praktikumsplätze; die Firmen haben für jeden Platz ein Ranking der Studierenden nach deren Eignung für den Platz. Wie soll man eine Zuordnung treffen? Was soll eigentlich das Kriterium für eine „gute“ Zuordnung sein? – Probleme dieser Art stellen sich in vielen Zusammenhängen, zum Beispiel auch bei der Zuteilung von Studienplätzen an Bewerber(innen). Untersucht wurde es im Zusammenhang mit der Verteilung von Assistenzärzten auf Krankenhäuser in den USA. Das Thema ist so wichtig und vielseitig, dass es schon früh ein ganzes Buch darüber gab[6].

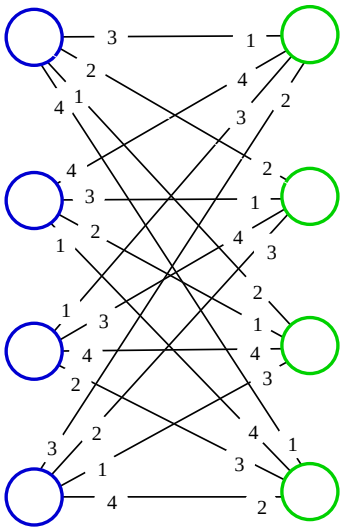

Abbildung 17: Ein vollständiger bipartiter Graph mit Präferenzen an den Knoten

Im etwas allgemeineren Fall hat man einen bipartiten Graphen mit Seiten U und W gegeben; jeder Knoten in U hat eine Rangfolge seiner Nachbarn in W und jeder Knoten in W hat eine Rangfolge seiner Nachbarn in U. Um die Sache nicht zu kompliziert zu machen, befassen wir uns hier nur mit dem vollständigen bipartiten Graphen $(U \cup W, E)$ mit $|U| = |W|$ und $E = U \times W$ und der Suche nach einem *perfekten* Matching $M \subseteq E$, das bestimmte Qualitätskriterien erfüllt. (Siehe Abb. 17. Varianten werden in der Übung betrachtet.) Die traditionelle Terminologie sieht so aus:

$U = \{u_1, \ldots, u_n\}$ ist eine Menge von n Männern;

$W = \{w_1, \ldots, w_n\}$ ist eine Menge von n Frauen;

[6] D. Gusfield und R. W. Irving, The Stable Marriage Problem: Structure and Algorithms. MIT Press, 1989

notfalls kommen alle Paare miteinander aus, aber jede Person hat ein (striktes) Ranking der Personen auf der anderen Seite.

Notation: $r_u \colon W \to \{1, \dots, n\}$ ist das Ranking von $u \in U$, eine injektive Abbildung. (Kleinere Werte bedeuten höhere Wertschätzung.) Analog ist $r_w \colon U \to \{1, \dots, n\}$ das Ranking von $w \in W$.

Wir suchen also ein „gutes“ perfektes Matching. Was soll das sein? Wir möchten erreichen, dass es kein Paar gibt, das in M nicht zusammen ist, aber miteinander zufriedener wäre als es in M ist. (Das ist natürlich nur *ein* mögliches Ziel bei der Herstellung von Zuordnungen.)

Definition 2.5.1. *Sei M ein Matching in $U \times W$. Wir nennen zwei Paare (u, w) und (u', w') in M eine* **Instabilität***, wenn zugleich gilt:*

$r_u(w') < r_u(w)$ *(d. h. u findet w' besser als w),*

$r_{w'}(u) < r_{w'}(u')$ *(d. h. w' findet u besser als u').*

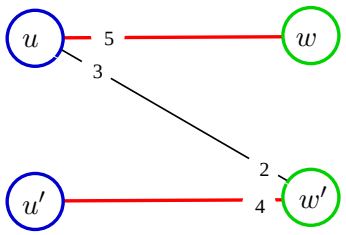

Abbildung 18: Eine Instabilität. Die waagerechten (roten) Kanten sind im Matching, aber die Endpunkte der diagonalen (schwarzen) Kante bevorzugen einander gegenüber den Matching-Partnern

In diesem Falle, so die Vorstellung, würden u und w' ihre Partner verlassen und sich zu einem neuen Paar zusammentun. Aus subjektiver Sicht der beiden ist dies eine Verbesserung, auch wenn das globale Ziel, dass jede(r) eine(n) Partner(in) hat, dadurch zerstört wird.

Definition 2.5.2. *Ein perfektes Matching M heißt eine* ***stabile Paarung****, wenn es keine Instabilität hat.*

Um eine stabile Paarung zu konstruieren, gibt es einen sehr eleganten Algorithmus (von Gale und Shapley[7], 1962), den wir hier betrachten wollen. Er formuliert den Ablauf als eine Folge von Heiratsanträgen, die sich gegenseitig Konkurrenz machen.

[7] D. Gale und L. S. Shapley: „College Admissions and the Stability of Marriage“, American Mathematical Monthly 69, 9–14, 1962. – Zusammen mit Alvin E. Roth erhielt Lloyd S. Shapley 2012 den Nobelpreis in Wirtschaftswissenschaften.

Er ist als Algorithmus 2 wiedergegeben. Er beginnt mit dem leeren Matching und besteht aus Runden. In jeder Runde macht irgendeine Frau w, die momentan nicht verlobt ist, einem Mann u einen Heiratsantrag. Wenn dieser frei ist, wird (u, w) zum Matching hinzugefügt. Wenn u mit w' verlobt ist, kommt es darauf an, ob er w oder w' höher schätzt. Wenn $r_u(w) < r_u(w')$, löst u die Verlobung und verlobt sich mit w (das Paar (u, w') im Matching wird durch (u, w) ersetzt), andernfalls ändert sich nichts. Die einzige Regel ist, dass jede Frau w für sich gesehen die Männer in der Reihenfolge ihres Rankings r_w abarbeitet. Der Algorithmus stoppt, wenn alle Frauen w verlobt sind (wenn das Matching perfekt geworden ist) oder alle Frauen ihre Listen abgearbeitet haben. Man sollte Algorithmus 2 am Graphen aus Abb. 17 durchspielen. Zum Vergleichen findet man das Resultat in Abb. 19.

Algorithm 2: Gale-Shapley: Heiratsantrags-Algorithmus

```
1  // Berechnet eine stabile Paarung M
2  Eingabe: Rankings r_u, u ∈ U der Männer und r_w, w ∈ W, der Frauen.
3  Es wird ein Matching M gebaut, das sich im Lauf der Zeit noch ändert.
4  (u, w) ∈ M heißt: u und w sind „verlobt“.
5  Initialisierung:
6    M := ∅;
7    while ∃w ∈ W(w ist momentan nicht verlobt und
8                 w hat noch nicht allen u einen Antrag gemacht) do
9      wähle ein solches w beliebig;
10     w macht dem ersten Mann u auf ihrer Liste,
11           den sie bisher noch nicht gefragt hat, einen Antrag.
12     1. Fall: u ist frei. – Dann: Verlobung, füge (u, w) zu M hinzu.
13     2. Fall: (u, w') ∈ M, u ist anderweitig verlobt.
14       Fall 2a: r_u(w') < r_u(w): u lehnt ab, keine Änderung.
15       Fall 2b: r_u(w) < r_u(w'): u nimmt an, M := (M − {(u, w')}) ∪ {(u, w)}.
16         (w' ist nicht mehr verlobt.)
17   Ausgabe: M.    „Massenhochzeit“
```

Es stellen sich Fragen nach Korrektheit und Zeitaufwand.

Beobachtungen: (i) Sobald ein Mann $u \in U$ einmal einen Antrag bekommen hat, bleibt er immer verlobt. Seine Partnerinnen können wechseln, aber er wechselt immer nur zu Partnerinnen, die ihm nach seiner Rangliste lieber sind: Einen Wechsel von (u, w') zu (u, w) gibt es nur, wenn $r_u(w') > r_u(w)$ gilt.
(ii) Eine Frau $w \in W$ kann auch wieder frei werden, wenn sie einmal mit einem $u \in U$ verlobt war. Wenn sie später wieder einen Partner u' findet, dann hat dieser aus ihrer Sicht einen schlechteren Rang als der vorherige: $r_w(u') > r_w(u)$. Dies liegt einfach daran, dass sie ihre Anträge in dieser Reihenfolge stellt.

Lemma 2.5.3. *Algorithmus 2 terminiert nach maximal n^2 Schleifendurchläufen.*

Beweis: Nach spätestens n^2 Durchläufen haben alle n Frauen ihre Ranglisten der

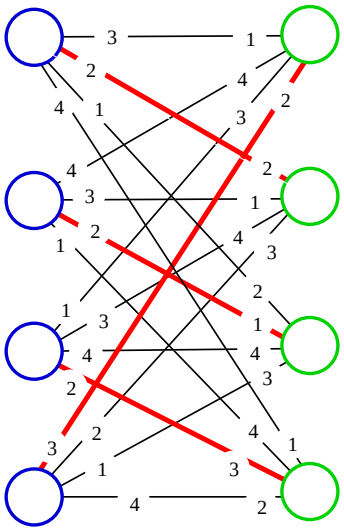

Abbildung 19: Resultat des Gale-Shapley-Algorithmus

Länge n abgearbeitet. □

Es ist auf den ersten Blick überhaupt nicht klar, dass am Ende jede Person eine(n) Partner(in) hat. Hierzu benutzt man ein Kardinalitätsargument.

Lemma 2.5.4. *Wenn Algorithmus 2 terminiert, dann ist M (die verlobten Paare) ein perfektes Matching.*

Beweis: Annahme: Der Algorithmus terminiert, aber M ist nicht perfekt. Weil $|U| = |W|$ gilt, muss es eine Frau w geben, die frei ist. Weil die Schleife beendet ist, hat w jedem Mann einen Antrag gemacht. Nach Beobachtung (i) ist daher jeder Mann verlobt, also ist $|M| = n$, Widerspruch. □

Satz 2.5.5. *Die Ausgabe M von Algorithmus 2 ist eine stabile Paarung.*

Beweis: Wir betrachten zwei beliebige Paare $(u, w), (u', w') \in M$ und zeigen, dass diese keine Instabilität bilden. Wenn $r_u(w') > r_u(w)$ gilt, ist nichts zu zeigen. Also können wir

$$r_u(w') < r_u(w)$$

annehmen. Wegen Beobachtung (i) kann w' niemals u einen Antrag gemacht haben. (Zum Zeitpunkt eines solchen Antrags wäre u entweder frei gewesen oder wäre mit einer Frau w'' verlobt gewesen, für die $r_u(w'') \geq r_u(w) > r_u(w')$ gilt. In beiden Fällen hätte u den Antrag angenommen. Dann müsste aber die endgültige Partnerin w von u bei u mindestens so beliebt sein wie w', d. h., es müsste $r_u(w) \leq r_u(w')$ gelten.) Aus der Tatsache, dass w' bei der Abarbeitung ihrer Liste nicht bis zu u gekommen ist, folgt nach Beobachtung (ii), dass $r_{w'}(u') < r_{w'}(u)$ gilt. Also bilden (u, w) und $(u', w') \in M$ *keine* Instabilität. □

Wir wollen unsere Überlegungen noch ein wenig erweitern. In Zeile 9 des Algorithmus wird die Frau w, die den nächsten Antrag macht, beliebig gewählt. Hat die Reihenfolge, in der die Anträge gemacht werden, Einfluss auf das Ergebnis? Kurioserweise ist das Ergebnis immer das gleiche – der „Nichtdeterminismus“ im Algorithmus bleibt ohne Auswirkungen. Algorithmen dieser Art sind natürlich besonders interessant.

Definition 2.5.6. *$u \in U$ heißt ein **möglicher Partner** für $w \in W$, falls es eine stabile Paarung M mit $(u, w) \in M$ gibt.* (Nach Satz 2.5.5 gibt es eine stabile Paarung, also hat jedes $w \in W$ mindestens einen möglichen Partner.)

$$\text{best}(w) := \text{der mögliche Partner } u \text{ von } w, \text{ der } r_w(u) \text{ minimiert.}$$

(*Der **beste** mögliche Partner von w.*)

$$M^* := \{(\text{best}(w), w) \mid w \in W\}.$$

Satz 2.5.7. *Die Ausgabe M von Algorithmus 2 ist M^*.*

Bevor wir den Satz beweisen, machen wir einige Bemerkungen.
(a) Bemerkenswert ist zunächst, dass der Nichtdeterminismus in Algorithmus 2 keine Auswirkungen auf das Ergebnis hat.
(b) Von der Definition her ist zunächst nicht einmal klar, dass M^* ein Matching ist.
(c) Alle Frauen bekommen gleichzeitig den besten Partner, den sie unter der Einschränkung „stabile Paarung“ (ein globales „soziales Gut“) überhaupt bekommen können.
(d) (Siehe Übung:) Alle Männer bekommen gleichzeitig die schlechteste mögliche Partnerin ...

Beweis von Satz 2.5.7: Indirekt. *Annahme*: Es gibt eine Ablaufreihenfolge $\mathcal{R}$ des Algorithmus von Gale-Shapley, die eine von M^* verschiedene stabile Paarung M erzeugt.

Nach Annahme und der Definition von M^* gibt es in M ein Paar (u, w) mit $u \neq \text{best}(w)$. Weil (u, w) in der stabilen Paarung M vorkommt, ist u möglicher Partner von w. Im Algorithmus ist festgelegt, dass w den Männern ihre Anträge in der Reihenfolge wachsender r_w-Werte (d. h. geringer werdender Wertschätzung) macht. Daraus folgt, dass w im Verlauf von $\mathcal{R}$ ihrem besten möglichen Partner best(w) einen Antrag gemacht hat, dieser sie aber abgewiesen hat oder sich später zugunsten einer anderen Frau von ihr getrennt hat. Damit wissen wir, dass es im Ablauf $\mathcal{R}$ einen Zeitpunkt t_0 gibt, zu dem *zum ersten Mal* einer Frau w_0 und einem möglichen Partner u_0 von w_0 Folgendes passiert:

w_0 macht u_0 einen Antrag, wird aber abgewiesen, da u_0 schon mit einer Frau w' verlobt ist, die er besser als w_0 findet
oder
w_0 ist bisher mit Mann u_0 verlobt, wird aber von diesem verlassen, da u_0 von einer Frau w' einen Antrag bekommt, die er besser als w_0 findet.
In beiden Fällen gilt also:

$$r_{u_0}(w') < r_{u_0}(w_0). \tag{4}$$

Wegen der Minimalität von t_0 und wegen der Anfragereihenfolge bei den Frauen ist u_0 nicht nur irgendein möglicher Partner von w_0, sondern sogar der beste mögliche Partner, d. h. $u_0 = \text{best}(w_0)$.

Weil u_0 möglicher Partner von w_0 ist, gibt es eine stabile Paarung M_0 mit $(u_0, w_0) \in M_0$. In M_0 kommt auch w' in einem Paar vor, etwa $(u', w') \in M_0$.

Wir betrachten nun die Geschichte von w' in $\mathcal{R}$: Unmittelbar nach Zeitpunkt t_0 ist w' mit u_0 verlobt. Nach Wahl von t_0 wurde w' selbst vor t_0 noch nie von einem möglichen Partner zurückgewiesen oder „entlobt“. Nun ist u' ein solcher möglicher Partner von w' (weil $(u', w') \in M_0$ für die stabile Paarung M_0 gilt). Es folgt, dass in $\mathcal{R}$ w' mit u' vor dem Zeitpunkt t_0 nichts zu tun hatte, das bedeutet, dass

$$r_{w'}(u_0) < r_{w'}(u') \tag{5}$$

gelten muss. Die Relationen (4) und (5) zusammen besagen, dass die Paare (u_0, w_0) und (u', w') eine Instabilität in der stabilen Paarung M_0 bilden (siehe Abb. 20), ein Widerspruch. □

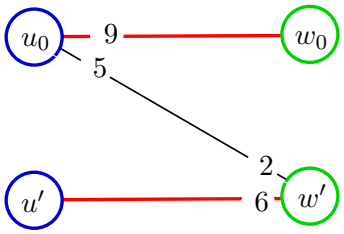

Abbildung 20: Die waagerechten (roten) Kanten gehören zu M_0. Mann u_0 schätzt w' mehr als seine M_0-Partnerin w_0 (nach (4)); Frau w' schätzt u_0 mehr als ihren M_0-Partner u' (nach (5)). Dies ist eine Instabilität. (Die konkret angegebenen Ränge sind nur Beispiele.)