4 Nächstbest-Algorithmen (greedy algorithms)

4.1 Optimierungsprobleme

- Ein *Optimierungsproblem* ist charakterisiert durch eine (häufig sehr große) Menge möglicher *Lösungen* und eine *Bewertungsfunktion*, die jeder Lösung einen Wert (z. B. eine natürliche oder reelle Zahl) zuordnet.

- Bei einem *Maximierungsproblem* wird eine Lösung mit maximalem Wert gesucht, bei einem *Minimierungsproblem* eine Lösung mit minimalem Wert.

Beispiele

- Finde in einem Straßennetz den kürzesten Weg von A nach B.

- Problem des Handlungsreisenden (traveling salesman problem):
  Finde eine kürzeste Rundreise durch N Orte $A_1, \ldots, A_N$.

- Blocksatz: Verteile eine Folge von Wörtern so auf Zeilen einer bestimmten Länge, dass der zum Auffüllen der Zeilen zusätzlich benötigte Zwischenraum möglichst gleichmäßig verteilt ist.
  Problem: Was bedeutet „möglichst gleichmäßig“, d. h. wie lautet eine sinnvolle Bewertungsfunktion?

4.2 Nächstbest-Algorithmen

- Ein *Nächstbest-Algorithmus* konstruiert schrittweise eine Lösung eines Optimierungsproblems, indem er in jedem Schritt die nächstbeste Erweiterung der bis jetzt konstruierten Teillösung wählt.

- Wenn das so jeweils gewählte *lokale Optimum* auch ein *globales Optimum* darstellt, findet der Algorithmus tatsächlich eine optimale Lösung.

- Andernfalls stellt die gefundene Lösung nur eine mehr oder weniger gute *Näherungslösung* oder *Approximation* der optimalen Lösung dar, die aber oft mit wesentlich weniger Aufwand als die optimale Lösung ermittelt werden kann.

Beispiele

- Der Handlungsreisende besucht als nächstes immer diejenige bis jetzt noch nicht besuchte Stadt, die der momentan besuchten Stadt am nächsten liegt (lokales Optimum, nearest neighbour algorithm).

  Auf diese Weise findet er für N Orte mit Aufwand $O(N^2)$ eine Rundreise, für die gezeigt werden kann, dass sie höchstens $\frac{\lceil \log_2 N \rceil + 1}{2}$-mal so lang ist wie eine kürzeste Rundreise (globales Optimum).

- Die meisten Textverarbeitungsprogramme füllen jede Ausgabezeile mit möglichst vielen Wörtern (lokales Optimum), ohne darauf zu achten, ob diese Strategie für nachfolgende Zeilen günstig ist oder nicht (globales Optimum), zum Beispiel:

```
Wenn man beim Blocksatz jede Zeile für sich betrachtet und
mit möglichst vielen Wörtern füllt, wird dieser Text bei
einer Zeilenbreite von 60 extrem schlecht umgebrochen.
Schuld ist die URL
https://en.wikipedia.org/wiki/Line_wrap_and_word_wrap, weil
sie sehr lang ist und eigentlich nicht sinnvoll getrennt
werden kann. Daher ist es besser, einen Absatz als Ganzes zu
betrachten und den Zwischenraum möglichst gleichmäßig über
alle Zeilen zu verteilen, wie es z. B. von TeX gemacht wird.
```

Besser wäre:

```
Wenn man beim Blocksatz jede Zeile für sich
betrachtet und mit möglichst vielen Wörtern füllt,
wird dieser Text bei einer Zeilenbreite von 60
extrem schlecht umgebrochen. Schuld ist die URL
https://en.wikipedia.org/wiki/Line_wrap_and_word_wrap, weil
sie sehr lang ist und eigentlich nicht sinnvoll getrennt
werden kann. Daher ist es besser, einen Absatz als Ganzes zu
betrachten und den Zwischenraum möglichst gleichmäßig über
alle Zeilen zu verteilen, wie es z. B. von TeX gemacht wird.
```

4.3 Huffman-Kodierung

4.3.1 Idee

❒ Buchstaben in natürlichsprachlichen Texten sind nicht gleichverteilt.
Für deutsche Texte gilt z. B. laut Wikipedia:

Buchstabe	*Häufigkeit*	*Buchstabe*	*Häufigkeit*	*Buchstabe*	*Häufigkeit*
E	17,40 %	U	4,35 %	K	1,21 %
N	9,78 %	L	3,44 %	Z	1,13 %
I	7,55 %	C	3,06 %	P	0,79 %
S	7,27 %	G	3,01 %	V	0,67 %
R	7,00 %	M	2,53 %	ß	0,31 %
A	6,51 %	O	2,51 %	J	0,27 %
T	6,15 %	B	1,89 %	Y	0,04 %
D	5,08 %	W	1,89 %	X	0,03 %
H	4,76 %	F	1,66 %	Q	0,02 %

❒ Wenn man häufige Buchstaben durch kürzere *Kodewörter* darstellt als seltene (*Kode variabler Länge*), brauchen Texte weniger Platz, als wenn alle Kodewörter gleich lang sind (*Kode fester Länge*), d. h. man erreicht eine *verlustfreie Kompression* (auch als *Entropiekodierung* bezeichnet).

- Um den kodierten Text wieder eindeutig dekodieren zu können, muss der Kode jedoch *präfixfrei* sein, d. h. kein Kodewort darf ein Präfix eines anderen Kodeworts sein.

- Beispiel:
  - Um die o. g. 27 Buchstaben mit jeweils gleich vielen Bits zu kodieren, benötigt man 5 Bit pro Buchstabe.
  - Damit benötigt ein Text mit 10 000 Zeichen 50 000 Bit.
  - Mit einem für die obige Verteilung optimalen *Präfixkode* (vgl. § 4.3.2) bräuchte man im Durchschnitt nur etwa 41 000 Bit.
  - Mit einem Präfixkode, der optimal für die tatsächliche Buchstabenverteilung im betrachteten Text ist, kann man u. U. noch eine deutlich höhere Kompression erzielen.

- Nach dem gleichen Prinzip lassen sich auch binäre Daten komprimieren, indem man häufig auftretende Bitfolgen (z. B. Bytes oder Maschinenwörter) durch weniger Bits kodiert als seltene.

- Dies wird z. B. als letzter Schritt bei der (ansonsten normalerweise verlustbehafteten) JPEG-Komprimierung eingesetzt.

4.3.2 Binäre Präfixkodes

- Ein *binärer Kode* ordnet jedem Zeichen des Eingabealphabets ein aus Nullen und Einsen bestehendes *Kodewort* zu.

- Bei einem *Präfixkode* darf kein Kodewort ein Präfix eines anderen Kodeworts sein.

- Ein binärer Präfixkode kann durch einen binären Baum dargestellt werden, in dem jedem Knoten wie folgt ein Kodewort zugeordnet ist:
  - Das Kodewort des Wurzelknotens ist die leere Zeichenkette.
  - Das Kodewort des linken bzw. rechten Nachfolgers eines Knotens entsteht durch Anhängen von 0 bzw. 1 an das Kodewort dieses Knotens.
  - Jedes Zeichen des Eingabealphabets entspricht einem Blattknoten mit dem entsprechenden Kodewort.

- Kodierung einer Zeichenfolge

  Für jedes Zeichen der Zeichenfolge:
  Gib das zugehörige Kodewort aus.

- Dekodierung einer Bitfolge
  1. Setze einen Zeiger auf den Wurzelknoten des Kodebaums.
  2. Für jedes Bit der Bitfolge:
     1. Wenn das Bit gleich 0 bzw. 1 ist, setze den Zeiger auf den linken bzw. rechten Nachfolger des aktuellen Knotens.
     2. Wenn dieser Knoten ein Blattknoten ist:
        Gib das zugehörige Zeichen des Eingabealphabets aus
        und setze den Zeiger wieder auf den Wurzelknoten.

- Beispiel
  - Präfixkode als Baum
  - Kodierung einer Zeichenfolge
  - Dekodierung einer Bitfolge

4.3.3 Optimale binäre Präfixkodes

- Gegeben sei ein Eingabealphabet Σ und eine zu kodierende Zeichenfolge $s \in \Sigma^*$.
- Für jedes Zeichen $c \in \Sigma$ sei $\mathrm{f}_s(c)$ die Häufigkeit von c in s.
- Für einen Kode bzw. Kodebaum T sei $\mathrm{d}_T(c)$ die Länge des Kodeworts von c in diesem Kode, d. h. die Tiefe des zugehörigen Blattknotens im Baum T.
- Wenn die Zeichenfolge s mit dem Kode T kodiert wird, ergibt sich eine Bitfolge der Länge $\mathrm{L}_T(s) = \sum_{c \in \Sigma} \mathrm{f}_s(c)\,\mathrm{d}_T(c)$.
- Ein Kode T ist *optimal* für die Zeichenfolge s, wenn $\mathrm{L}_T(s)$ minimal ist, d. h. wenn $\mathrm{L}_T(s) \leq \mathrm{L}_{T'}(s)$ für alle Kodes T'.
- Frage: Wie findet man einen solchen optimalen Kode?

Lemma 1

- Wenn x und y die Zeichen in Σ mit der kleinsten Häufigkeit sind, dann gibt es einen optimalen Präfixkode, in dem sich die Kodewörter von x und y nur in ihrem letzten Bit unterscheiden, d. h. im Kodebaum sind die zu x und y gehörenden Blattknoten Geschwister.

Beweisidee

- Überführe irgendeinen optimalen Kodebaum T in einen ebenfalls optimalen Kodebaum T'', der die behauptete Eigenschaft besitzt.

Beweis

- Sei T irgendein optimaler Kodebaum.
- Betrachte in diesem Baum zwei Geschwisterblätter a und b mit maximaler Tiefe, d. h. $\mathrm{d}_T(a) = \mathrm{d}_T(b) \geq \mathrm{d}_T(z)$ für alle $z \in \Sigma$.
- O. B. d. A. gilt: $\mathrm{f}_s(a) \leq \mathrm{f}_s(b)$ und $\mathrm{f}_s(x) \leq \mathrm{f}_s(y)$.
  (Andernfalls vertausche a und b bzw. x und y.)

- Da x und y die Zeichen in Σ mit der kleinsten Häufigkeit sind, folgt:
  $f_s(x) \leq f_s(a)$ und $f_s(y) \leq f_s(b)$.

- Sei T' der Kodebaum, der aus T durch Vertauschen der Blattknoten a und x entsteht.

- Somit gilt:

  $$L_T(s) - L_{T'}(s) = \sum_{c \in \Sigma} f_s(c)\, d_T(c) - \sum_{c \in \Sigma} f_s(c)\, d_{T'}(c) =$$

  $$f_s(x)\, d_T(x) + f_s(a)\, d_T(a) - f_s(x)\, d_{T'}(x) - f_s(a)\, d_{T'}(a) =$$

  $$f_s(x)\, d_T(x) + f_s(a)\, d_T(a) - f_s(x)\, d_T(a) - f_s(a)\, d_T(x) =$$

  $$(f_s(a) - f_s(x))\,(d_T(a) - d_T(x)) \geq 0$$

  wegen $f_s(a) \geq f_s(x)$ und $d_T(a) \geq d_T(x)$.

- Daraus folgt: $L_T(s) \geq L_{T'}(s)$.

- Da T optimal ist, gilt jedoch auch: $L_T(s) \leq L_{T'}(s)$.

- Also gilt: $L_T(s) = L_{T'}(s)$, d. h. auch T' ist ein optimaler Kode.

- Sei nun T'' der Kodebaum, der aus T' durch Vertauschen der Blattknoten b und y entsteht.

- Für diesen gilt analog: $L_{T''}(s) = L_{T'}(s) = L_T(s)$, d. h. T'' ist ein optimaler Kodebaum.

- Da a und b im ursprünglichen Kodebaum T Geschwisterblätter sind, sind x und y im resultierenden Kodebaum T'' ebenfalls Geschwisterblätter, sodass T'' tatsächlich die behauptete Eigenschaft besitzt.

Lemma 2

- x und y seien wieder die Zeichen in Σ mit der kleinsten Häufigkeit.
- Das Alphabet Σ' sei definiert als $\Sigma' = \Sigma \setminus \{ x, y \} \cup \{ z \}$,
  wobei $z \notin \Sigma$ ein künstliches neues Zeichen mit Häufigkeit $f_{s'}(z) = f_s(x) + f_s(y)$ sei,
  d. h. gedanklich ersetzt man jedes Auftreten von x und y in s durch z.
- Sei T' ein optimaler Kodebaum für die so modifizierte Zeichenfolge s'
  und T der Baum, der aus T' entsteht, wenn man den Blattknoten z durch einen
  inneren Knoten ersetzt, der die Blattknoten x und y als Nachfolger besitzt.
- Dann ist T ein optimaler Kodebaum für die Zeichenfolge s.

Beweis:

- Aufgrund der Konstruktion von T aus T' gilt:
  - $d_T(x) = d_T(y) = d_{T'}(z) + 1$
  - $d_T(c) = d_{T'}(c)$ für alle $c \in \Sigma \setminus \{ x, y \} = \Sigma' \setminus \{ z \}$

- Daraus folgt:

$$L_T(s) - L_{T'}(s') = \sum_{c \in \Sigma} f_s(c)\, d_T(c) - \sum_{c \in \Sigma'} f_{s'}(c)\, d_{T'}(c) =$$

$$f_s(x)\, d_T(x) + f_s(y)\, d_T(y) - f_{s'}(z)\, d_{T'}(z) =$$

$$(f_s(x) + f_s(y))\, (d_{T'}(z) + 1) - f_{s'}(z)\, d_{T'}(z) =$$

$$f_{s'}(z)\, (d_{T'}(z) + 1) - f_{s'}(z)\, d_{T'}(z) = f_{s'}(z)$$

- Also gilt: $L_T(s) = L_{T'}(s') + f_{s'}(z)$

- Sei nun $\tilde{T}$ ein optimaler Kodebaum für s,
der o. B. d. A. die Eigenschaft von Lemma 1 erfüllt,
und $\tilde{T}'$ der Baum, der aus $\tilde{T}$ entsteht, wenn man die Blattknoten x und y und ihren gemeinsamen Vorgänger durch einen neuen Knoten z mit Häufigkeit $f_{s'}(z) = f_s(x) + f_s(y)$ ersetzt.

- Dann gilt entsprechend aufgrund der Konstruktion von $\tilde{T}'$ aus $\tilde{T}$:
$L_{\tilde{T}}(s) = L_{\tilde{T}'}(s') + f_{s'}(z)$.

- Da $\tilde{T}$ und T' jeweils optimal sind, folgt daraus:
$L_{\tilde{T}}(s) \leq L_T(s) = L_{T'}(s') + f_{s'}(z) \leq L_{\tilde{T}'}(s') + f_{s'}(z) = L_{\tilde{T}}(s)$

- Also gilt: $L_T(s) = L_{\tilde{T}}(s)$, d. h. T ist ein optimaler Kodebaum.

Rekursiver Algorithmus zur Bestimmung eines optimalen Kodes

1 Wenn das Alphabet Σ genau zwei Zeichen x und y enthält:

  Ordne x das Kodewort 0 und y das Kodewort 1 zu.

2 Andernfalls (d. h. wenn Σ mehr als zwei Zeichen enthält):

  1 Ermittle die Zeichen x und y mit der kleinsten Häufigkeit.

  2 Erzeuge ein künstliches neues Zeichen z mit Häufigkeit $f(z) = f(x) + f(y)$.

  3 Führe den Algorithmus rekursiv für das Alphabet $\Sigma' = \Sigma \setminus \{ x, y \} \cup \{ z \}$ aus.

  4 Ordne x bzw. y das Kodewort von z gefolgt von 0 bzw. 1 zu
  und übernimm die Kodewörter für alle anderen Zeichen aus $\Sigma \setminus \{ x, y \} = \Sigma' \setminus \{ z \}$.

Anmerkungen

- Die Korrektheit des Algorithmus folgt direkt aus Lemma 2.
- Problem: Wie findet man effizient die Zeichen mit der kleinsten Häufigkeit?
- Die Implementierung ist aufwendig, weil für jeden rekursiven Aufruf ein neues Alphabet Σ' und eine modifizierte Häufigkeitsverteilung konstruiert werden muss.

Iterativer Nächstbest-Algorithmus mit Vorrangwarteschlange

1 Für jedes Zeichen $c \in \Sigma$:

  Erzeuge einen zugehörigen Blattknoten
  und füge ihn mit Priorität $\mathrm{f}(c)$ in eine Minimum-Vorrangwarteschlange ein.

2 Solange die Warteschlange mindestens zwei Elemente enthält:

  1 Entnimm die Knoten x und y mit minimaler Priorität/Häufigkeit.

  2 Erzeuge einen inneren Knoten mit x und y als Nachfolgern
  und füge ihn mit Priorität $\mathrm{f}(x) + \mathrm{f}(y)$ in die Warteschlange ein.

3 Entnimm den verbleibenden Knoten aus der Warteschlange,
der den Wurzelknoten des gesamten Kodebaums darstellt.

4 Weise jedem Zeichen $c \in \Sigma$ sein Kodewort anhand dieses Baums zu (vgl. § 4.3.2).

Laufzeit des iterativen Algorithmus

❒ Bei geeigneter Implementierung der Vorrangwarteschlange (vgl. Kapitel 3) hat der Algorithmus die Laufzeit $O(N \log_2 N)$, wenn N die Größe des Alphabets Σ bezeichnet.

Beispiel: $\mathrm{f}(A) = 1$, $\mathrm{f}(B) = 2$, $\mathrm{f}(C) = 3$, $\mathrm{f}(D) = 4$, $\mathrm{f}(E) = 5$, $\mathrm{f}(F) = 6$

4.4 Das Problem der stabilen Ehen (stable marriage problem)

4.4.1 Problemstellung

- Gegeben seien N Männer und N Frauen.
- Die *Präferenzliste* einer Person ist eine Permutation der Personen des anderen Geschlechts.
- Eine Person P *bevorzugt* eine Person Q des anderen Geschlechts gegenüber einer weiteren Person R des anderen Geschlechts (in Zeichen: $Q \underset{P}{<} R$), wenn Q in der Präferenzliste von P vor R steht.
- Eine *Zuordnung* ist eine bijektive Funktion, die jeder Person P eineindeutig einen Partner P° des anderen Geschlechts zuordnet.
- Eine Zuordnung ist *labil*, wenn es einen Mann M und eine Frau F gibt, die zusammen *die Stabilität der Zuordnung gefährden*, d. h. die sich beide verbessern würden, wenn sie ein Paar wären:
  - $F \underset{M}{<} M^\circ$, d. h. M bevorzugt F gegenüber der ihm zugeordneten Frau M°
  - $M \underset{F}{<} F^\circ$, d. h. F bevorzugt M gegenüber dem ihr zugeordneten Mann F°

  Andernfalls ist die Zuordnung *stabil*.
- Gesucht ist eine stabile Zuordnung.

4.4.2 Lösung mit Nächstbest-Algorithmus

Herrenwahl

1 Am Anfang sind alle Personen solo

2 Solange es einen Mann M gibt, der noch oder wieder solo ist:

   1 M fragt die erste Frau F auf seiner Präferenzliste, die er noch nicht gefragt hat

   2 Wenn F noch solo ist:

      M und F bilden ein (vorläufiges) Paar, d. h. beide sind dann nicht mehr solo

   3 Andernfalls hat F bereits einen (vorläufigen) Mann $\tilde{M}$

      Wenn $M \underset{F}{<} \tilde{M}$:

      1 F trennt sich von $\tilde{M}$, d. h. anschließend ist $\tilde{M}$ wieder solo

      2 M und F bilden ein neues (vorläufiges) Paar

Damenwahl

Analog mit vertauschten Rollen

4.4.3 Beispiel

Anton
1. Hanna
2. Berta
3. Doris
4. Frieda

Christian
1. Hanna
2. Berta
3. Frieda
4. Doris

Emil
1. Doris
2. Frieda
3. Berta
4. Hanna

Gustav
1. Berta
2. Hanna
3. Frieda
4. Doris

Berta
1. Christian
2. Anton
3. Gustav
4. Emil

Doris
1. Anton
2. Christian
3. Gustav
4. Emil

Frieda
1. Gustav
2. Anton
3. Christian
4. Emil

Hanna
1. Emil
2. Christian
3. Anton
4. Gustav

4.4.4 Korrektheit und weitere Eigenschaften

- Die folgenden Aussagen gelten für Herrenwahl.
- Wenn man die Rollen von Männern und Frauen vertauscht, erhält man entsprechende Aussagen für Damenwahl.

Behauptung 1

- Es gibt keinen Mann, der von jeder Frau abgewiesen wird.

Beweis

- Wenn ein Mann von einer Frau abgewiesen wird, dann hat diese in diesem Moment einen anderen Mann.
- Sobald eine Frau einmal einen Mann hat, hat sie bis zum Ende immer einen Mann.
- Wenn ein Mann von allen Frauen abgewiesen werden würde, dann wäre er am Ende solo, während jede Frau einen Mann haben müsste (weil sie den fraglichen Mann abgewiesen hat).
- Widerspruch, da es gleich viele Männer und Frauen gibt.

Folgerungen

- Der Algorithmus terminiert:
  Spätestens nach N „Versuchen“ hat jeder Mann eine Frau, die er nicht wieder verliert, d. h. er ist dann dauerhaft nicht mehr solo.

- Der Algorithmus ist wohldefiniert:
  Wenn ein Mann solo ist, dann gibt es noch mindestens eine Frau auf seiner Präferenzliste, die er noch nicht gefragt hat.

- Am Ende hat jeder Mann eine Frau und damit auch jede Frau einen Mann, d. h. der Algorithmus ermittelt wirklich eine Zuordnung.

Behauptung 2

- Der Algorithmus ermittelt eine stabile Zuordnung.

Beweis

- Nach Ausführung des Algorithmus gilt für jeden Mann M und jede Frau F:
  - Wenn $F \underset{M}{\geq} M^{\circ}$ gilt, dann stellen M und F keine Gefahr für die Stabilität der Zuordnung dar.
  - Wenn $F \underset{M}{<} M^{\circ}$ gilt, dann hat M F vor M° gefragt und wurde offenbar (sofort oder später) wegen eines anderen Manns $\tilde{M}$ mit $\tilde{M} \underset{F}{<} M$ abgewiesen.
  - Da sich eine Frau während der Durchführung des Algorithmus nicht mehr „verschlechtern“ kann, gilt außerdem $F^{\circ} \underset{F}{\leq} \tilde{M}$ und somit $F^{\circ} \underset{F}{<} M$.
  - Also stellen M und F auch in diesem Fall keine Gefahr für die Stabilität der Zuordnung dar.
- Also ist die Zuordnung stabil.

Folgerung

- Es gibt immer eine stabile Zuordnung.

Definition

- Für eine Person P sei P^+ der beste und P^- der schlechteste Partner, den P bei einer stabilen Zuordnung bekommen kann, das heißt:
  - Es gibt eine stabile Zuordnung, bei der $P^\circ = P^+$ ist.
  - Es gibt eine (möglicherweise andere) stabile Zuordnung, bei der $P^\circ = P^-$ ist.
  - Für jede stabile Zuordnung gilt: $P^+ \underset{P}{\leq} P^\circ \underset{P}{\leq} P^-$.

Behauptung 3

- Der Algorithmus ordnet jedem Mann M seine optimale Frau M^+ zu.

Beweis durch Widerspruch

- Annahme: Es gibt einen Mann M, dem vom Algorithmus eine Frau M° mit $M^\circ \underset{M}{>} M^+$ zugeordnet wird.
- Das heißt: M wird während der Durchführung des Algorithmus von seiner optimalen Frau M^+ wegen eines anderen Manns $\tilde{M}$ mit $\tilde{M} \underset{M^+}{<} M$ abgewiesen, d. h. $\tilde{M}$ und M^+ bilden in diesem Moment ein Paar.

- Wenn es mehrere Männer gibt, denen das passiert, sei M derjenige, dem dies während der Durchführung des Algorithmus als erstem passiert.
- $\tilde{M}$ wurde bis dahin genau von den Frauen F abgewiesen, für die $F \underset{\tilde{M}}{<} M^+$ gilt.
- Da M der erste Mann ist, der von seiner optimalen Frau abgewiesen wird, wurde $\tilde{M}$ von seiner optimalen Frau $\tilde{M}^+$ noch nicht abgewiesen, d. h. für $\tilde{M}^+$ gilt nicht $\tilde{M}^+ \underset{\tilde{M}}{<} M^+$, sondern vielmehr $\tilde{M}^+ \underset{\tilde{M}}{\geq} M^+$.
- Sei Z irgendeine stabile Zuordnung, bei der M seine optimale Frau M^+ bekommt, und sei $\tilde{F}$ die Frau von $\tilde{M}$ bei dieser Zuordnung.
- Für diese Frau $\tilde{F}$ gilt natürlich: $\tilde{F} \underset{\tilde{M}}{\geq} \tilde{M}^+$.
- Aus $\tilde{F} \underset{\tilde{M}}{\geq} \tilde{M}^+$ und $\tilde{M}^+ \underset{\tilde{M}}{\geq} M^+$ folgt insgesamt: $\tilde{F} \underset{\tilde{M}}{\geq} M^+$.
- Da $\tilde{F}$ in der Zuordnung Z zu $\tilde{M}$ und M^+ zu M gehört, muss $\tilde{F} \neq M^+$ sein und somit $\tilde{F} \underset{\tilde{M}}{>} M^+$ gelten.

- Da außerdem $\tilde{M} \underset{M^+}{<} M$ gilt (siehe oben), stellen $\tilde{M}$ und M^+ eine Gefahr für die Stabilität der Zuordnung Z dar.

- Dies steht im Widerspruch dazu, dass Z eine stabile Zuordnung ist.

Folgerung

- Da der Algorithmus jedem Mann – unabhängig von der Reihenfolge, in der die Männer durchlaufen werden – immer seine optimale Frau zuordnet, spielt die Reihenfolge offenbar keine Rolle.

Behauptung 4

- Der Algorithmus ordnet jeder Frau F den schlechtestmöglichen Mann F^- zu.

Beweis durch Widerspruch

- Annahme: Es gibt eine Frau F, der vom Algorithmus ein Mann $\tilde{M} = F^\circ$ mit $\tilde{M} \underset{F}{<} F^-$ zugeordnet wird.
- Sei Z eine stabile Zuordnung, bei der F und F^- ein Paar bilden, und sei $\tilde{F}$ die Frau von $\tilde{M}$ bei dieser Zuordnung.
- Da der Algorithmus jedem Mann seine optimale Frau zuordnet, gilt: $F = \tilde{M}^\circ = \tilde{M}^+ \underset{\tilde{M}}{\leq} \tilde{F}$.
- Außerdem sind F (Partner von F^- bei der Zuordnung Z) und $\tilde{F}$ (Partner von $\tilde{M}$) verschieden, da F^- und $\tilde{M}$ verschieden sind (siehe oben).
  Deshalb gilt: $F \underset{\tilde{M}}{<} \tilde{F}$.
- Damit stellen F und $\tilde{M}$ eine Gefahr für die Stabilität der Zuordnung Z dar.
- Dies steht im Widerspruch dazu, dass Z eine stabile Zuordnung ist.

4.4.5 Laufzeit

- Jeder der N Männer fragt im schlimmsten Fall jede der N Frauen.
- Daher beträgt die Laufzeit des Algorithmus $O(N^2)$.

4.4.6 Anwendungsbeispiel

- Gegeben seien Projekte $P_1, \ldots, P_K$ mit Mitarbeiterbedarf $N_1, \ldots, N_K$ sowie $N = N_1 + \ldots + N_K$ Mitarbeiter.

- Jeder Mitarbeiter erstellt eine Präferenzliste der Projekte (d. h. eine Permutation von $P_1, \ldots, P_K$).

- Jedes Projekt P_i wird durch N_i „Strohmänner“ P_{ij} ($j = 1, \ldots, N_i$) repräsentiert, die alle die gleiche Präferenzliste der Mitarbeiter (d. h. eine Permutation von $M_1, \ldots, M_N$) haben, die z. B. vom Projektleiter erstellt wird.

- In den Präferenzlisten der Mitarbeiter wird jedes Projekt P_i durch die Folge der zugehörigen Strohmänner ersetzt.

- Dann kann der Algorithmus mit den N Mitarbeitern und den N Strohmännern ausgeführt werden, um eine möglichst gute Zuordnung der Mitarbeiter zu den Projekten zu ermitteln.

- Je nachdem, ob die Mitarbeiter oder die Strohmänner die Rolle der Männer spielen, erhält man entweder eine für die Mitarbeiter oder für die Projekte (bzw. Projektleiter) optimale Zuordnung.