Storyline-Visualisierungen für wissenschaftliche Kollaborationsgraphen

Kolloquium

Tim Herrmann

Gliederung

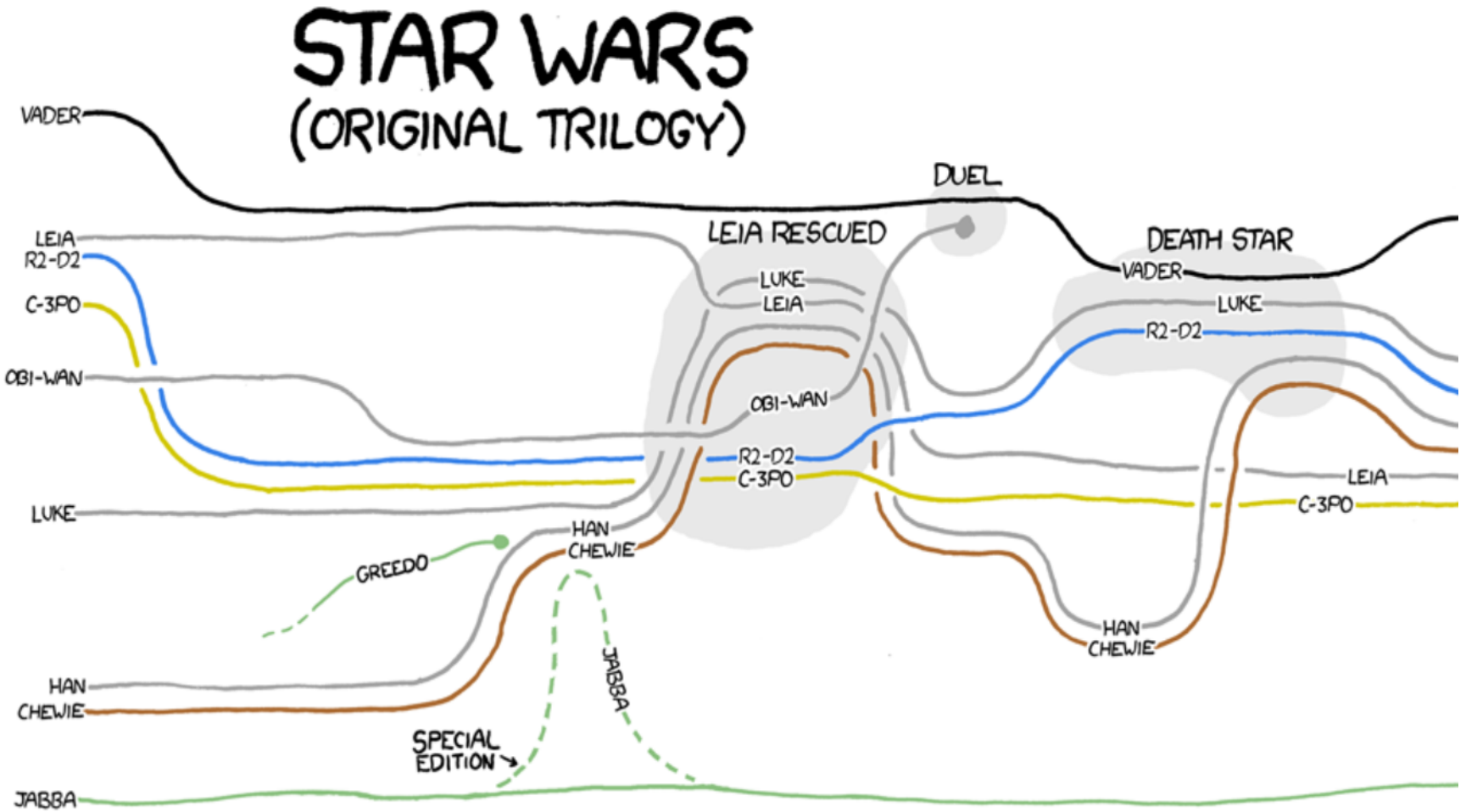
1. Grundlagen
2. Greedy-Algorithmus
3. Auswertung und Fazit

Gliederung

1. **Grundlagen**
   (a) Storyline-Visualisierung
   (b) Problemstellung
2. Greedy-Algorithmus
3. Auswertung und Fazit

Storyline-Visualisierung

auf xkcd wurden mehrere Visualisierungen für verschiedene Geschichten veröffentlicht



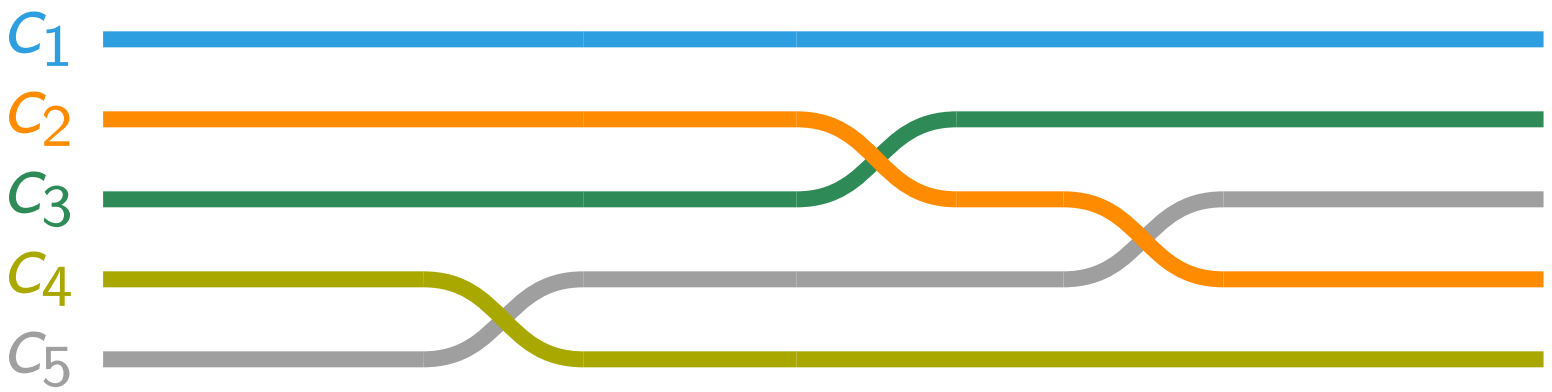
Storyline-Visualisierung

Storyline: $S = (C, M)$

Storyline-Visualisierung

Storyline: $S = (C, M)$ $C = \{c_1, c_2, \ldots, c_k\}$

Menge der Charaktere

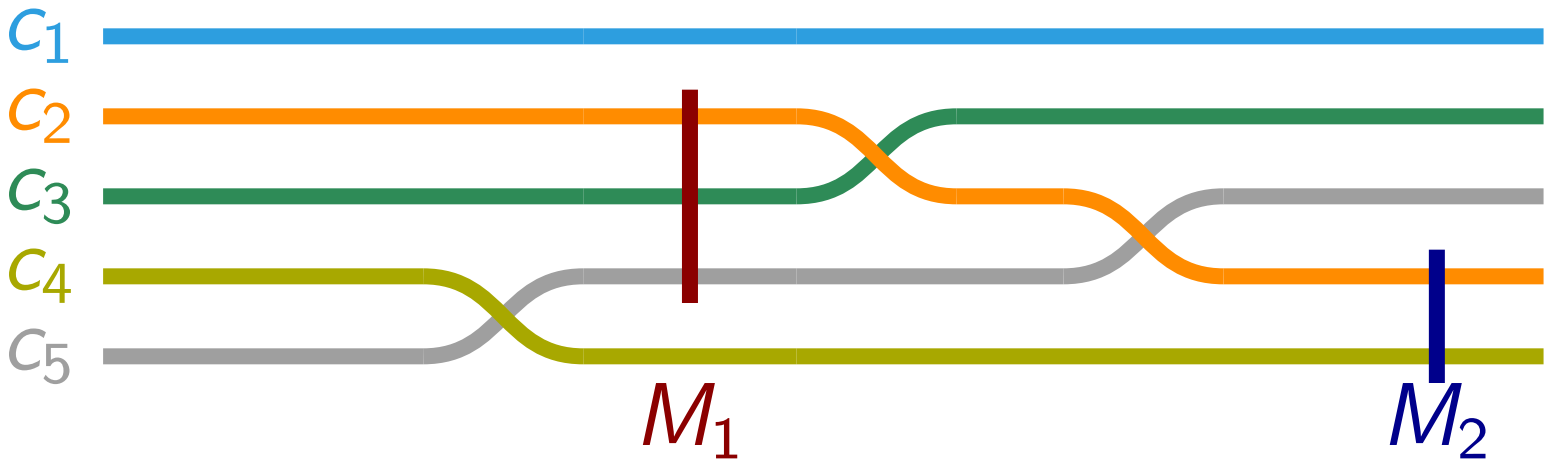


Charaktere werden durch x-monotone Kurven dargestellt

Storyline-Visualisierung

Storyline: $S = (C, M)$ $\quad C = \{c_1, c_2, \ldots, c_k\}$

Abfolge von Treffen $\longrightarrow$ $M = (M_1, M_2, \ldots, M_m)$

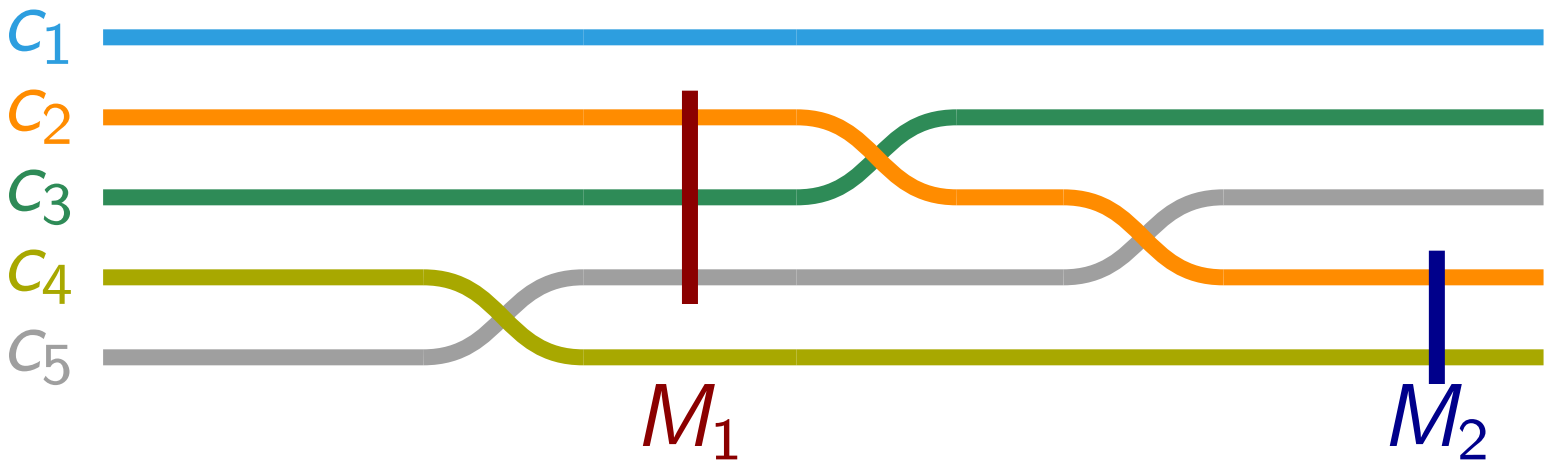


Treffen werden durch vertikale Linien dargestellt

Storyline-Visualisierung

Storyline: $S = (C, M)$ $\quad C = \{c_1, c_2, \ldots, c_k\}$

Abfolge von Treffen $\longrightarrow M = (M_1, M_2, \ldots, M_m)$



Treffen werden durch vertikale Linien dargestellt

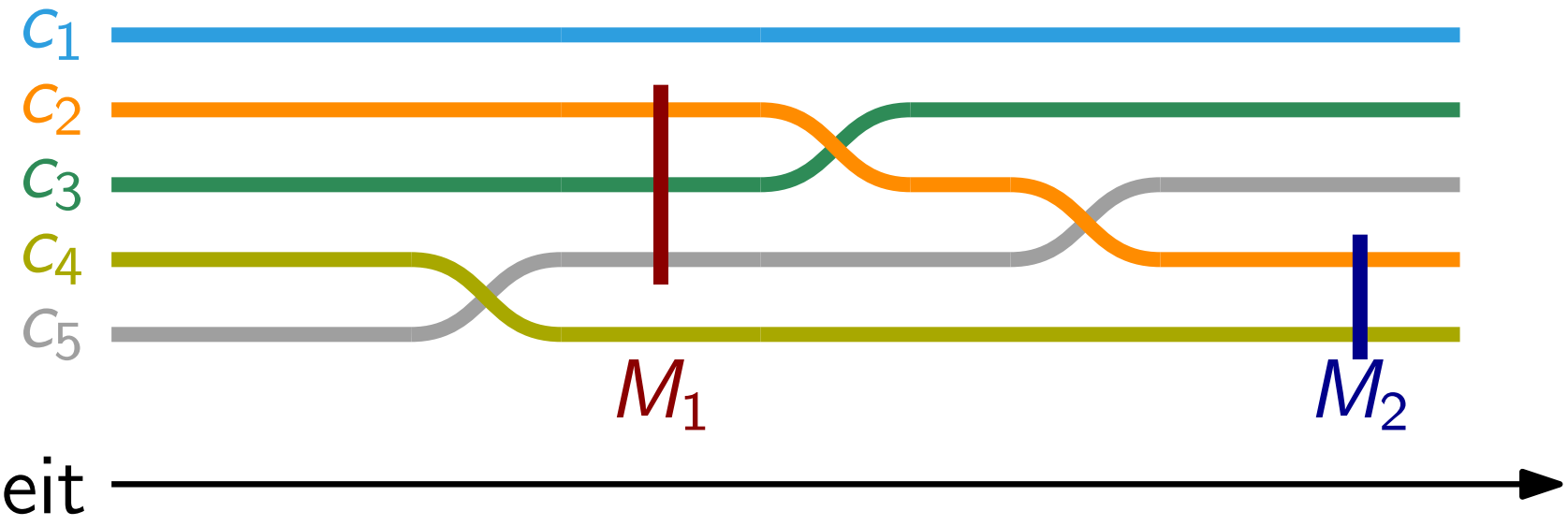
jedes Treffen $M_i \in M$:

- $|M_i| \geq 2$
- $M_i \subseteq C$

Storyline-Visualisierung

Storyline: $S = (C, M)$ $\quad C = \{c_1, c_2, \ldots, c_k\}$

Abfolge von Treffen $\longrightarrow$ $M = (M_1, M_2, \ldots, M_m)$



Treffen werden durch vertikale Linien dargestellt

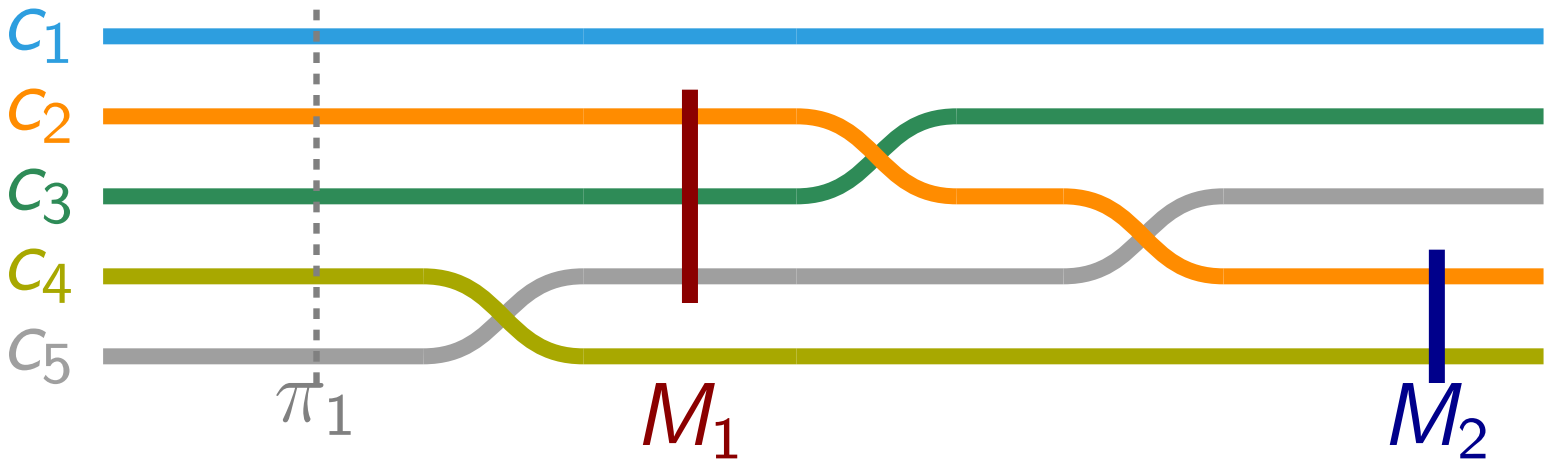
jedes Treffen $M_i \in M$:

- $|M_i| \geq 2$
- $M_i \subseteq C$

Storyline-Visualisierung

Storyline: $S = (C, M)$ $\quad C = \{c_1, c_2, \ldots, c_k\}$

$M = (M_1, M_2, \ldots, M_m)$

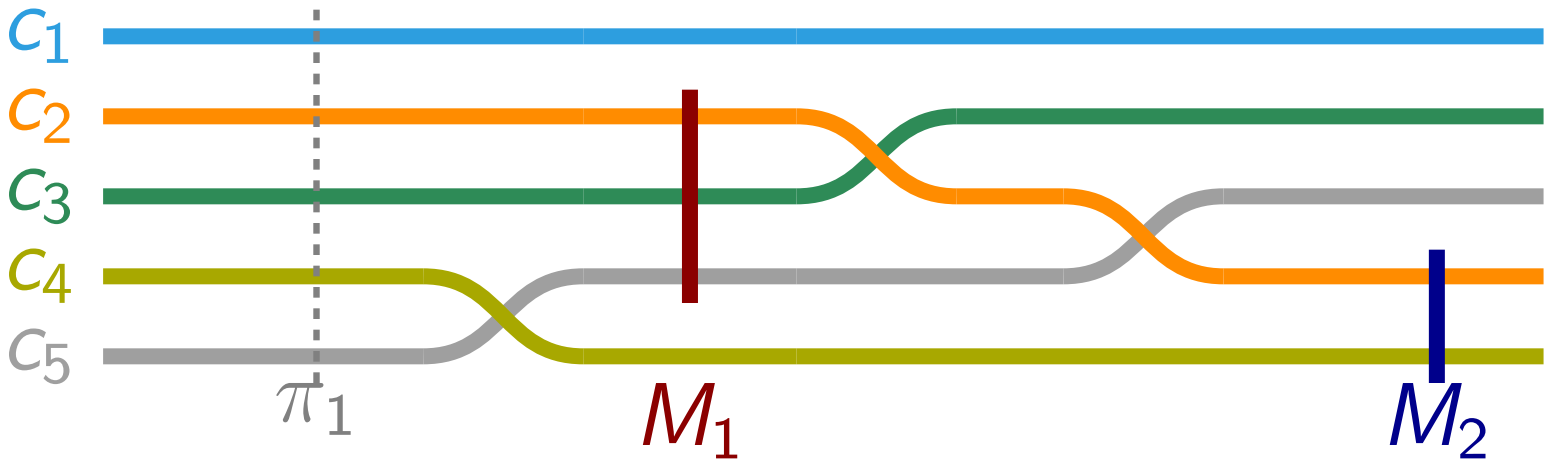


Permutation π:
vertikale Anordnung der Charaktere in der Visualisierung

Storyline-Visualisierung

Storyline: $S = (C, M)$ $\quad C = \{c_1, c_2, \ldots, c_k\}$

$M = (M_1, M_2, \ldots, M_m)$



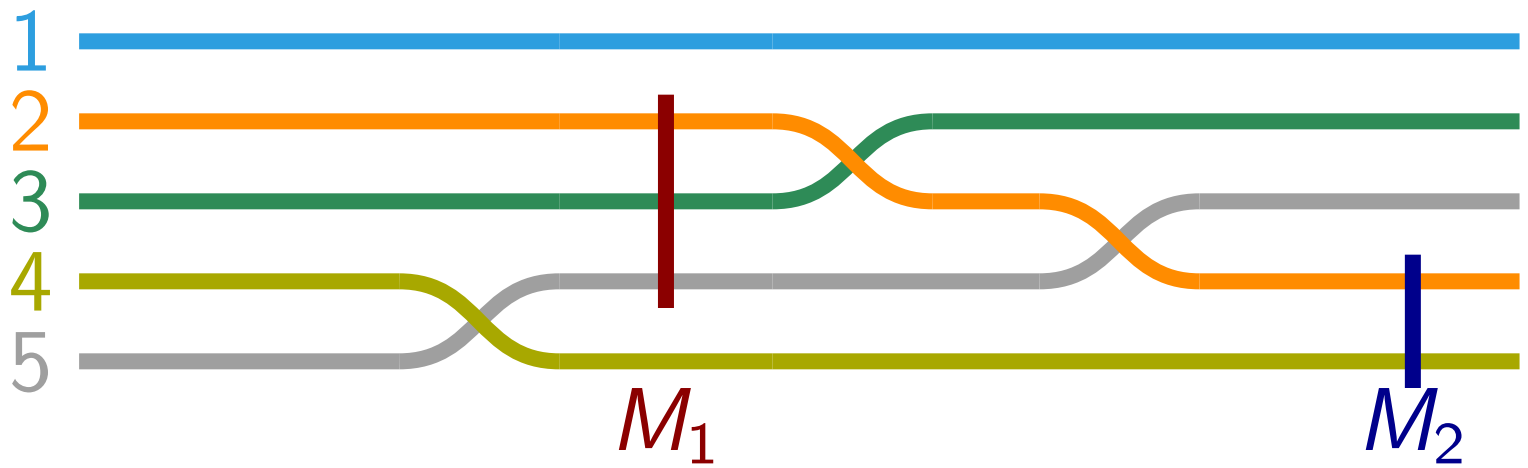
Permutation π:
vertikale Anordnung der Charaktere in der Visualisierung

Treffen $M_i \in M$ können in π stattfinden, falls die Charaktere aus M_i ein Intervall in π bilden

Storyline-Visualisierung

Storyline: $S = (C, M)$ $\quad C = \{1, 2, 3, 4, 5\}$

$M = (\{2, 3, 5\}, \{2, 4\})$

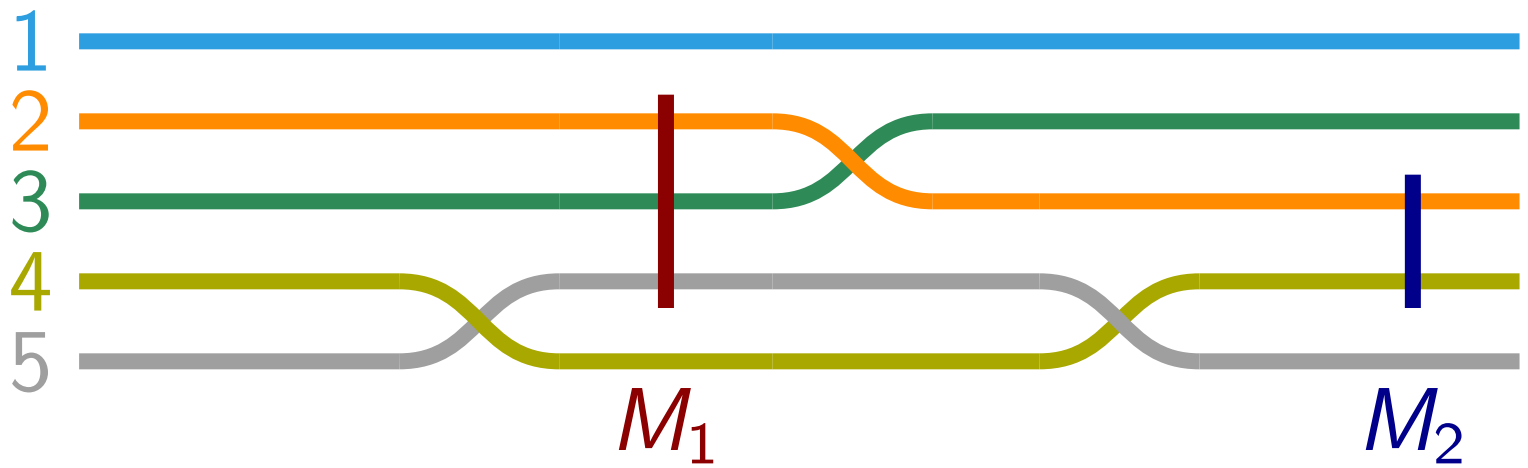


$\Rightarrow$ viele mögliche Visualisierungen möglich

Storyline-Visualisierung

Storyline: $S = (C, M)$ $\quad C = \{1, 2, 3, 4, 5\}$

$M = (\{2, 3, 5\}, \{2, 4\})$

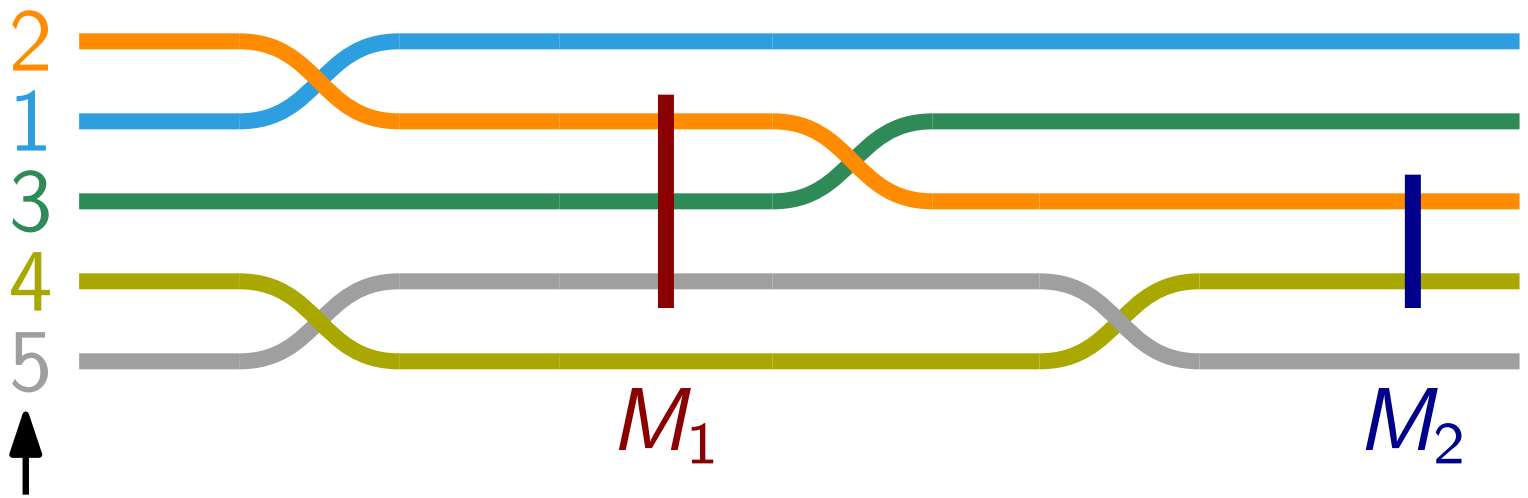


$\Rightarrow$ viele mögliche Visualisierungen möglich

Storyline-Visualisierung

Storyline: $S = (C, M)$ $\quad C = \{1, 2, 3, 4, 5\}$

$M = (\{2, 3, 5\}, \{2, 4\})$



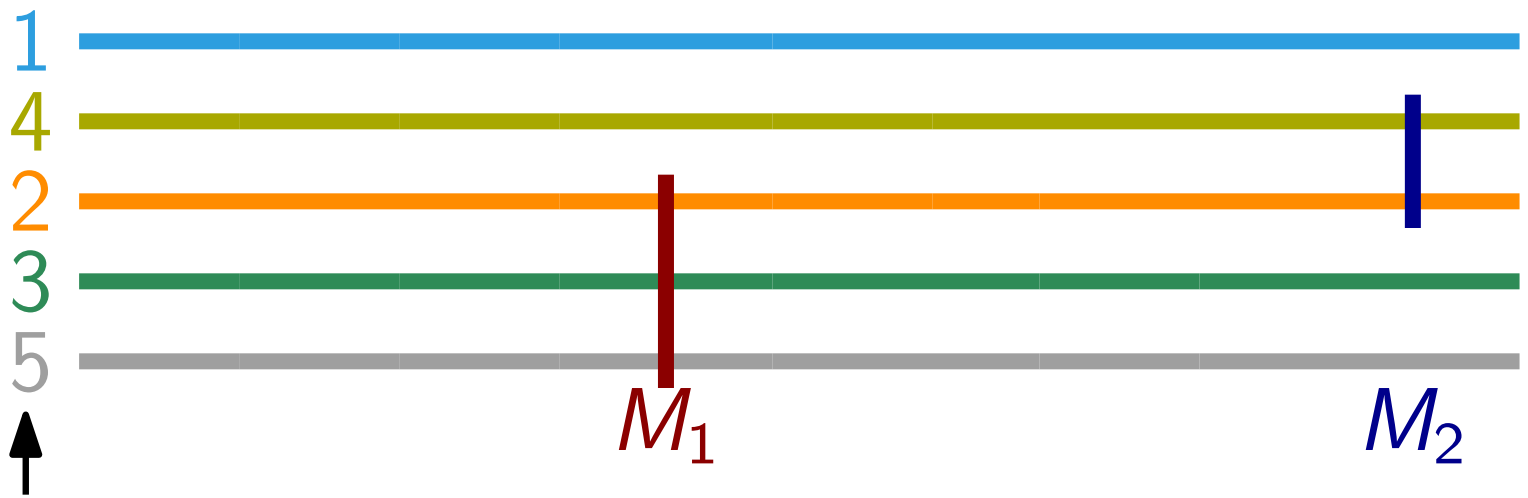
Andere Startpermutation

$\Rightarrow$ viele mögliche Visualisierungen möglich

Storyline-Visualisierung

Storyline: $S = (C, M)$ $\quad C = \{1, 2, 3, 4, 5\}$

$M = (\{2, 3, 5\}, \{2, 4\})$



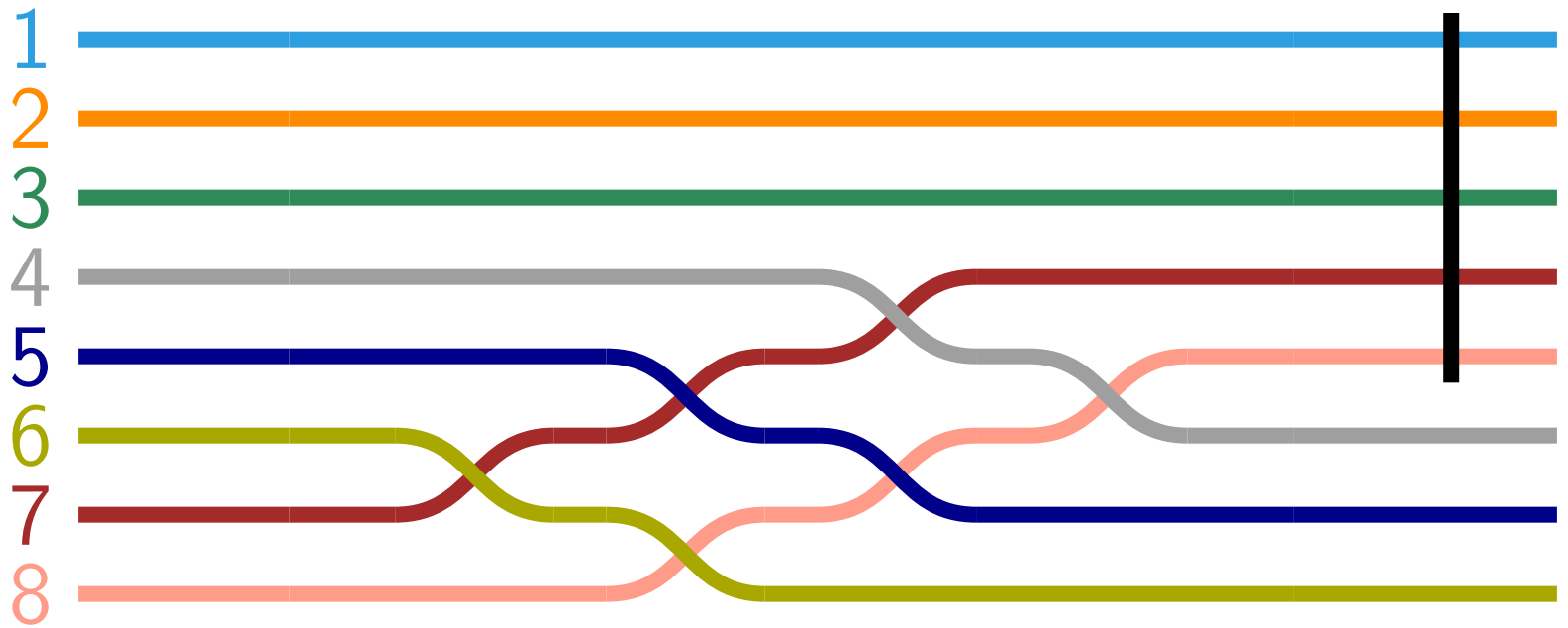
Andere Startpermutation

$\Rightarrow$ viele mögliche Visualisierungen möglich

$\Rightarrow$ häufig soll die Anzahl der Kreuzungen minimiert werden

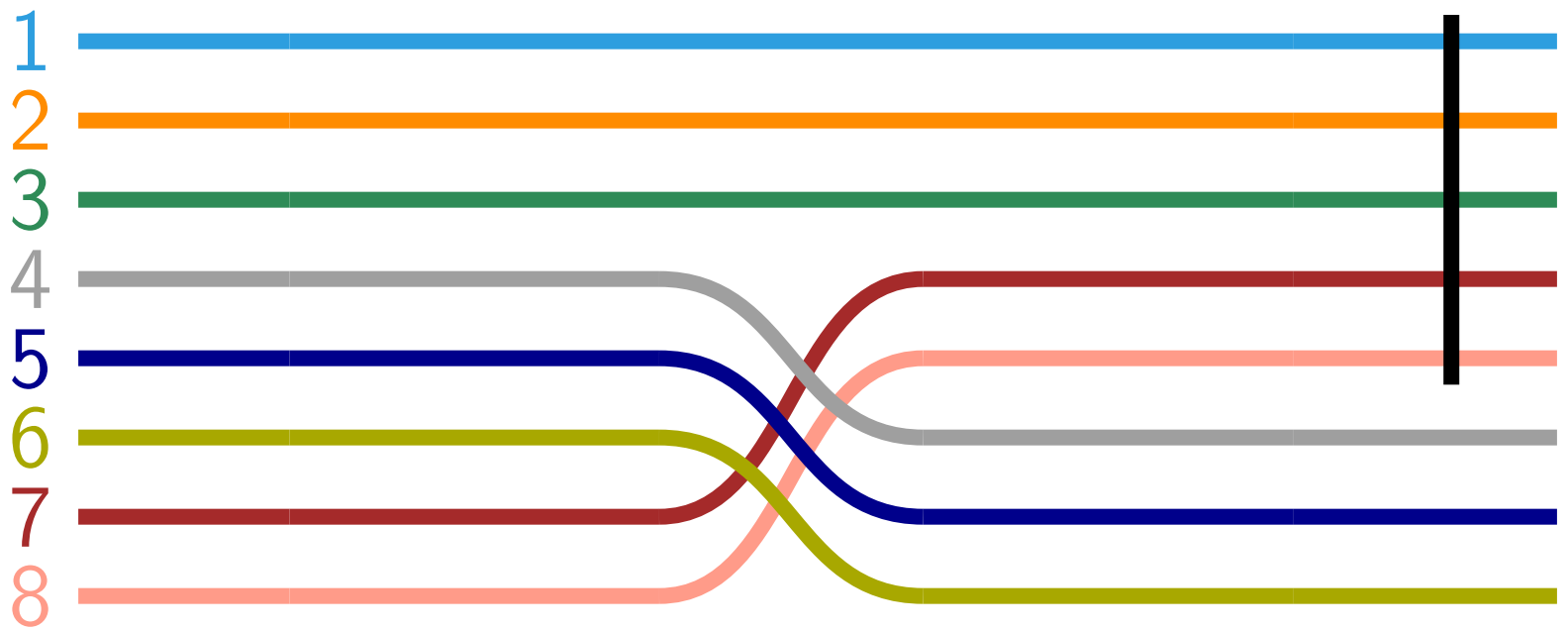
Blockkreuzung

$$C = \{1, 2, 3, 4, 5, 6, 7, 8\} \quad M = (\{1, 2, 3, 7, 8\})$$



Blockkreuzung

$C = \{1, 2, 3, 4, 5, 6, 7, 8\}$ $M = (\{1, 2, 3, 7, 8\})$



Benachbarte Stränge kreuzen sich

$\Rightarrow$ kann Visualisierung lesbarer machen

Problemstellung

Es soll eine Webapplikation erstellt werden, die für eine Anfrage von Autoren eine Storyline-Visualisierung zeichnet, welche die Kollaboration der betrachteten Autoren aufzeigt.

Problemstellung

Es soll eine Webapplikation erstellt werden, die für eine Anfrage von Autoren eine Storyline-Visualisierung zeichnet, welche die Kollaboration der betrachteten Autoren aufzeigt.

Die Visualisierung soll möglichst wenig Blockkreuzungen beinhalten.

Problemstellung

Die dazu notwendigen Informationen werden aus der Publikationsdatenbank `dblp` abgefragt.

Problemstellung

Die dazu notwendigen Informationen werden aus der Publikationsdatenbank `dblp` abgefragt.

Der Greedy-Algorithmus soll eine Storyline-Visualisierung mit minimaler Anzahl an Blockkreuzungen erstellen.

Problemstellung

Die dazu notwendigen Informationen werden aus der Publikationsdatenbank `dblp` abgefragt.

Der Greedy-Algorithmus soll eine Storyline-Visualisierung mit minimaler Anzahl an Blockkreuzungen erstellen.

Für den Greedy-Algorithmus werden weiterhin die Bezeichnungen *Charaktere* und *Treffen* verwendet.

Gliederung

1. Grundlagen
2. **Greedy-Algorithmus**
   (a) Grundidee
   (b) Beispiel
   (c) Laufzeit
   (d) Startpermutationen
3. Auswertung und Fazit

Grundidee

Algorithmus 2: GreedySBCM(Liste C, Liste M)

Eingabe: Liste mit Charakteren C, Liste mit Treffen M

Ausgabe: Startpermutation π, Liste mit Blockkreuzungen B

Grundidee

Algorithmus 2: GreedySBCM(Liste C, Liste M)

Eingabe: Liste mit Charakteren C, Liste mit Treffen M

Ausgabe: Startpermutation π, Liste mit Blockkreuzungen B

1 erstelle zufällige Startpermutation π der Charaktere

2 erstelle leere Liste von Blockkreuzungen B

3 i=1

4 **while** $i \leq M$.length **do**

Grundidee

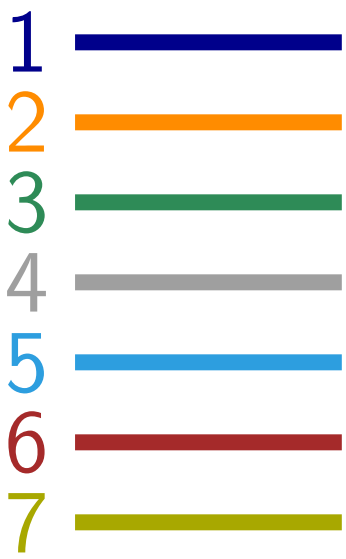
Algorithmus 2: GreedySBCM(Liste C, Liste M)

Eingabe: Liste mit Charakteren C, Liste mit Treffen M
Ausgabe: Startpermutation π, Liste mit Blockkreuzungen B

```
1 erstelle zufällige Startpermutation π der Charaktere
2 erstelle leere Liste von Blockkreuzungen B
3 i=1
4 while i ≤ M.length do
5     if isValidMeeting(π, M_i) then
6         i++
7     else
8         führe scheinbar geeignete Blockkreuzung durch
9 return (π, B)
```

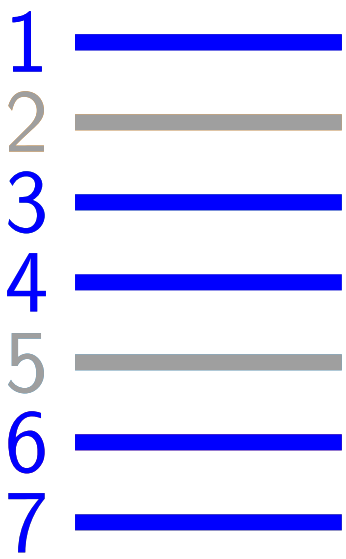
Grundidee

$M_1 = \{1, 3, 4, 6, 7\}$



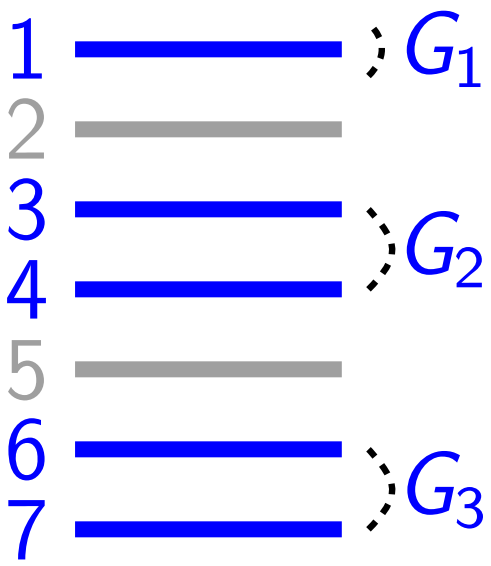
Grundidee

$M_1 = \{1, 3, 4, 6, 7\}$



Grundidee

$M_1 = \{1, 3, 4, 6, 7\}$

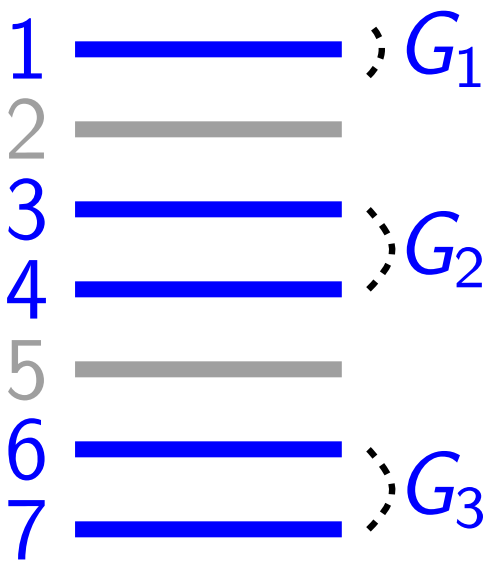


benachbarte Charaktere:

- müssen nicht mehr zusammengeführt werden
- können gruppiert werden

Grundidee

$M_1 = \{1, 3, 4, 6, 7\}$



benachbarte Charaktere:

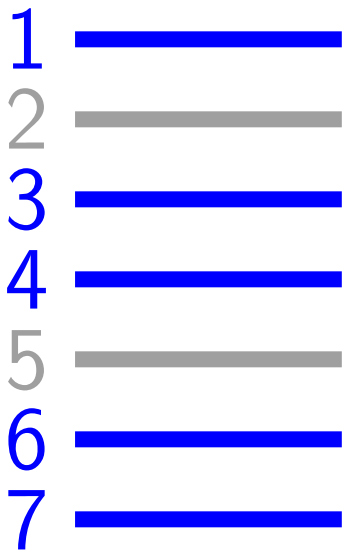
- müssen nicht mehr zusammengeführt werden
- können gruppiert werden

$\Rightarrow$ Werden diese Gruppen mit Blockkreuzungen zueinander geführt, kann das Treffen M_1 stattfinden

Grundidee

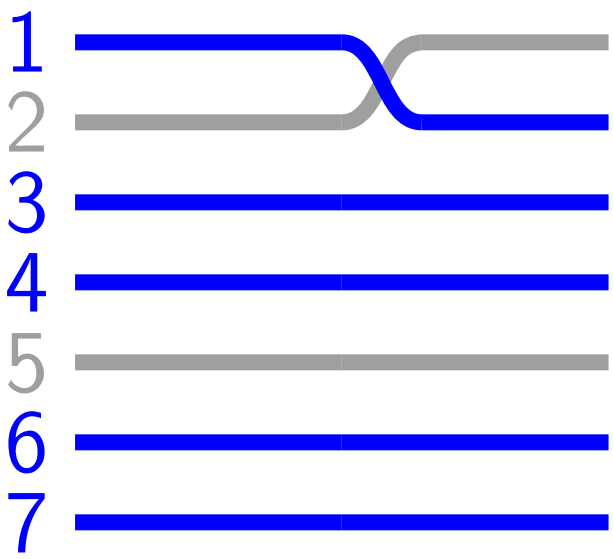
$M_1 = \{1, 3, 4, 6, 7\}$

Lösung des Beispiels:



Grundidee

$M_1 = \{1, 3, 4, 6, 7\}$

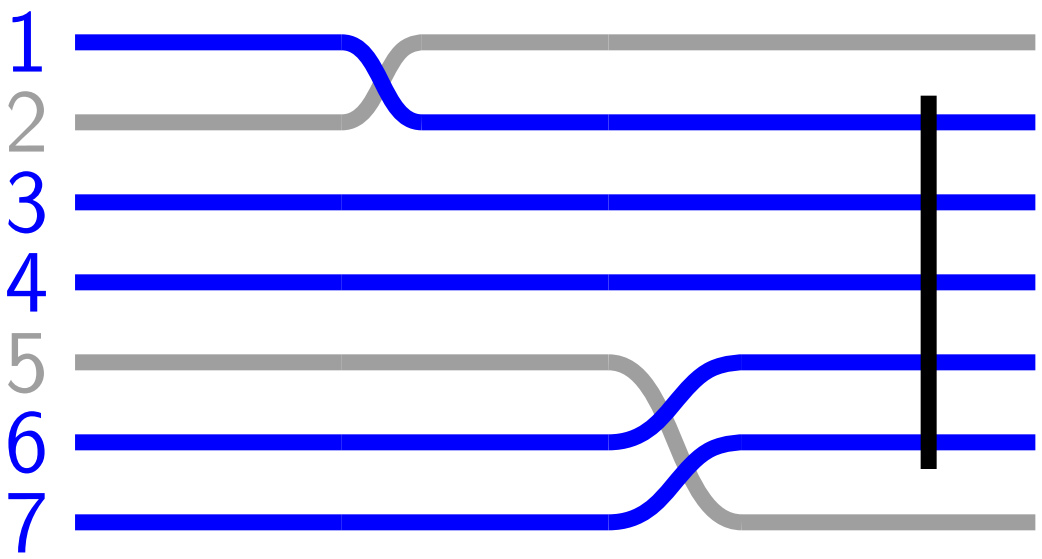


Lösung des Beispiels:

Grundidee

$M_1 = \{1, 3, 4, 6, 7\}$

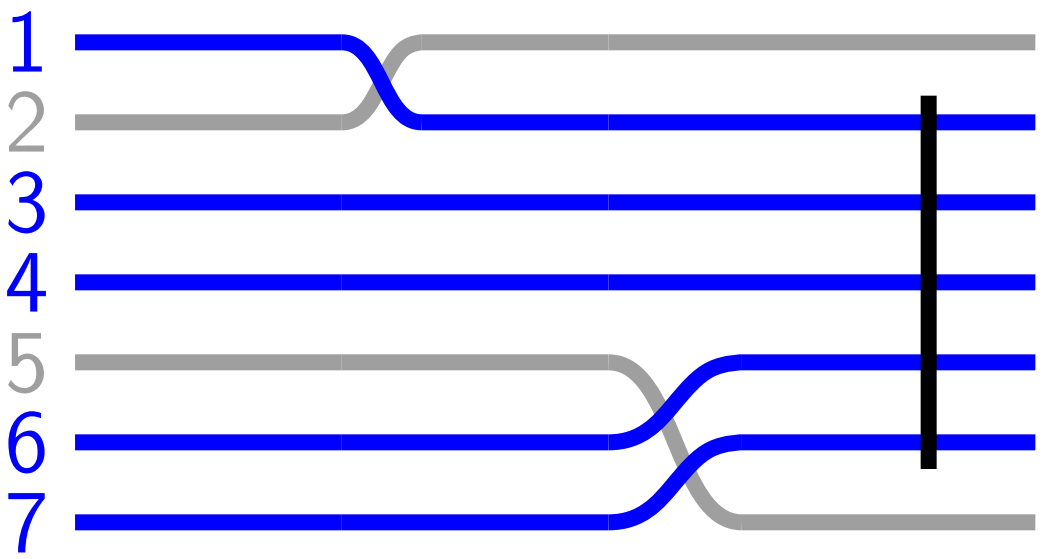
Lösung des Beispiels:



Grundidee

$M_1 = \{1, 3, 4, 6, 7\}$

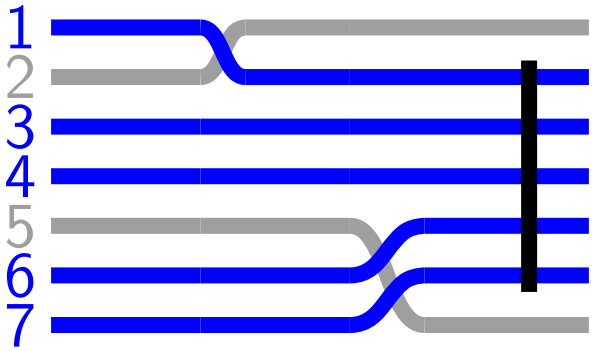
Lösung des Beispiels:



- jede Blockkreuzung verbindet maximal zwei Gruppen
- für g Gruppen werden demnach mindestens $g - 1$ Blockkreuzungen benötigt

Grundidee

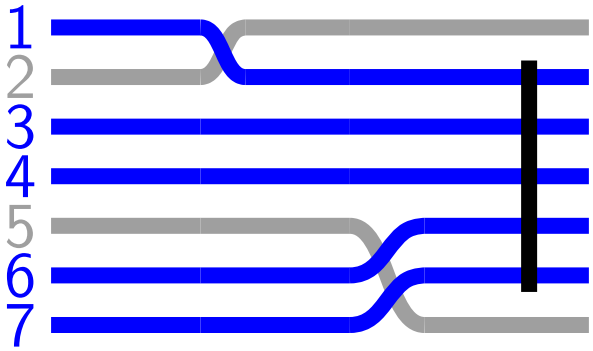
$M_1 = \{1, 3, 4, 6, 7\}$



$\Rightarrow$ die Anzahl der Gruppen g soll für die nächsten Treffen möglichst klein sein

Grundidee

$M_1 = \{1, 3, 4, 6, 7\}$



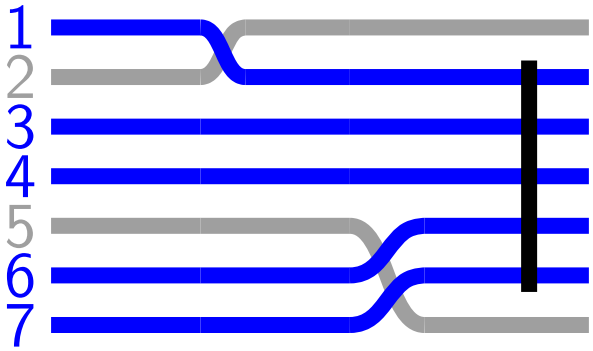
$\Rightarrow$ die Anzahl der Gruppen g soll für die nächsten Treffen möglichst klein sein

Blockkreuzungen sollen:

- zwei Gruppen zusammenführen
- Charaktere zu Nachbarn machen, die sich zukünftig treffen

Grundidee

$M_1 = \{1, 3, 4, 6, 7\}$



$\Rightarrow$ die Anzahl der Gruppen g soll für die nächsten Treffen möglichst klein sein

Blockkreuzungen sollen:

- zwei Gruppen zusammenführen
- Charaktere zu Nachbarn machen, die sich zukünftig treffen

Dafür soll ein Ähnlichkeitsmaß verwendet werden.

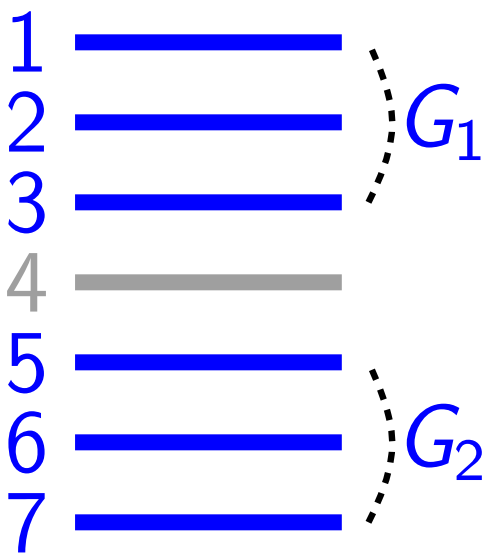
Charakterwahl

Es soll ein Charakterpaar aus verschiedenen Gruppen ausgewählt werden, welches zusammengeführt wird

Charakterwahl

Es soll ein Charakterpaar aus verschiedenen Gruppen ausgewählt werden, welches zusammengeführt wird

$M_1 = \{1, 2, 3, 5, 6, 7\}$

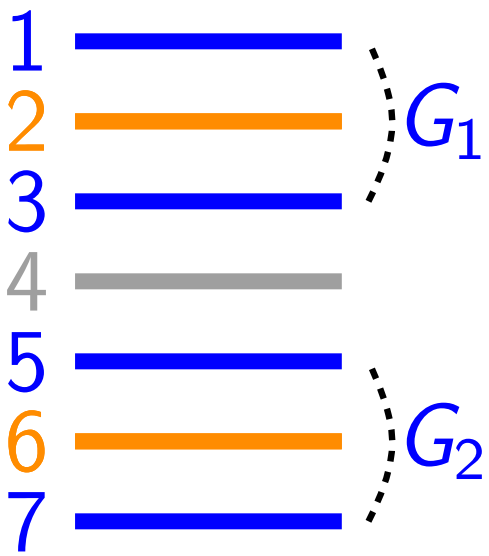


Charakterwahl

Es soll ein Charakterpaar aus verschiedenen Gruppen ausgewählt werden, welches zusammengeführt wird

$M_1 = \{1, 2, 3, 5, 6, 7\}$

Paar aus M_1 mit höchstem Ähnlichkeitsmaß: $(2, 6)$

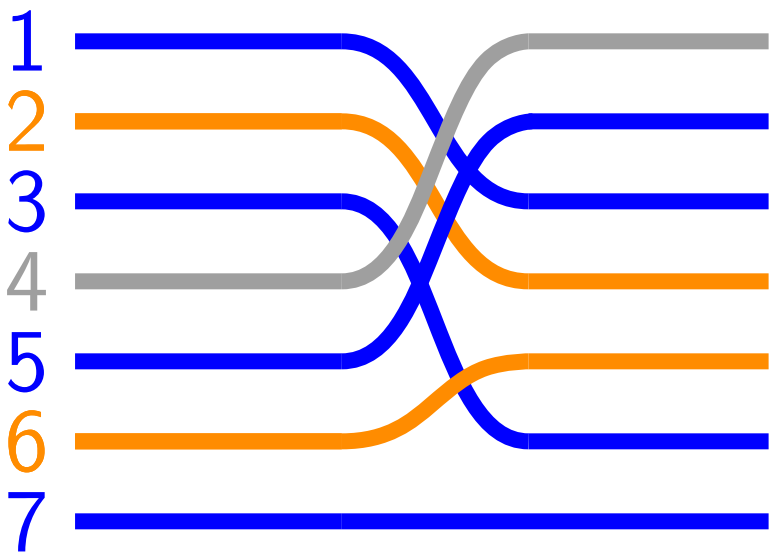


Charakterwahl

Es soll ein Charakterpaar aus verschiedenen Gruppen ausgewählt werden, welches zusammengeführt wird

$M_1 = \{1, 2, 3, 5, 6, 7\}$

Paar aus M_1 mit höchstem Ähnlichkeitsmaß: $(2, 6)$

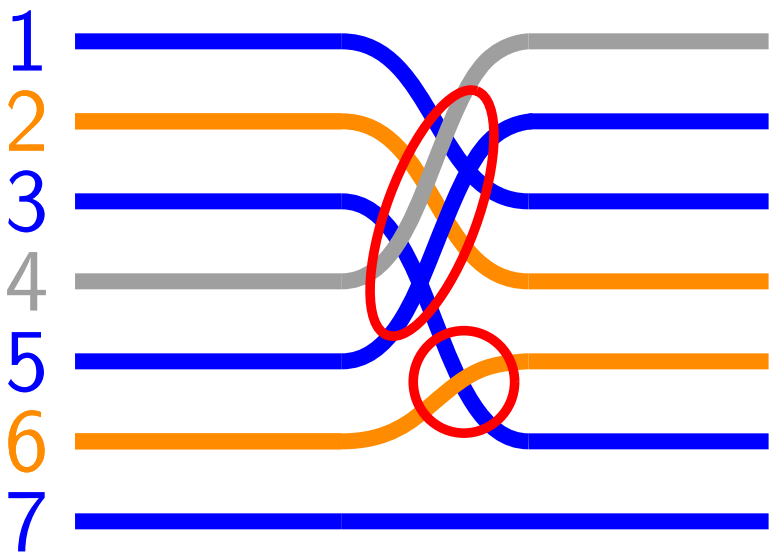


Charakterwahl

Es soll ein Charakterpaar aus verschiedenen Gruppen ausgewählt werden, welches zusammengeführt wird

$M_1 = \{1, 2, 3, 5, 6, 7\}$

Paar aus M_1 mit höchstem Ähnlichkeitsmaß: $(2, 6)$



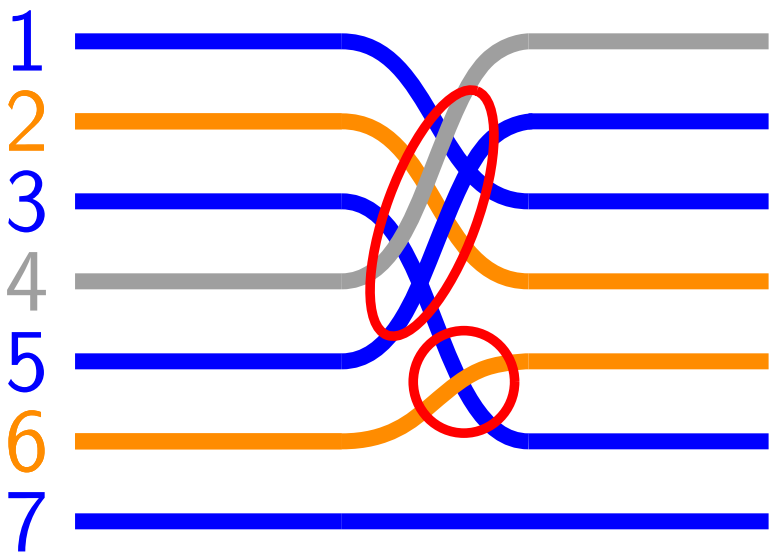
$\Rightarrow$ Es werden zwei Blockkreuzungen benötigt!

Charakterwahl

Es soll ein Charakterpaar aus verschiedenen Gruppen ausgewählt werden, welches zusammengeführt wird

$M_1 = \{1, 2, 3, 5, 6, 7\}$

Paar aus M_1 mit höchstem Ähnlichkeitsmaß: $(2, 6)$



$\Rightarrow$ Es werden zwei Blockkreuzungen benötigt!

$\Rightarrow$ Weitere Einschränkungen sind notwendig

Ähnlichkeitsmaß

Wähle Charakterpaar (p, q) folgendermaßen:

- p und q sind Teil verschiedener Gruppen
- p liegt am Rand seiner Gruppe

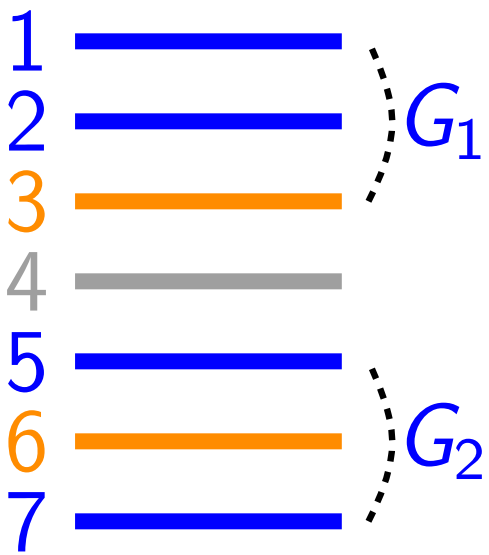
Ähnlichkeitsmaß

Wähle Charakterpaar (p, q) folgendermaßen:

- p und q sind Teil verschiedener Gruppen
- p liegt am Rand seiner Gruppe

$M_1 = \{1, 2, 3, 5, 6, 7\}$

Paar aus M_1 mit höchstem Ähnlichkeitsmaß: $(3, 6)$



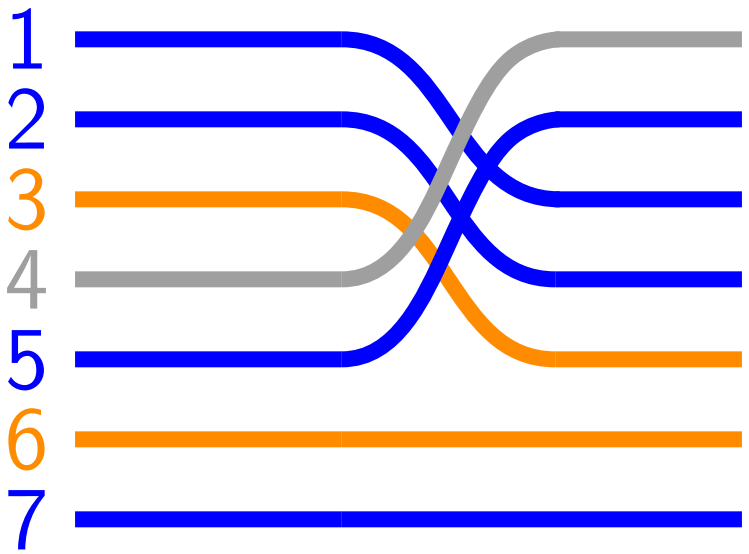
Ähnlichkeitsmaß

Wähle Charakterpaar (p, q) folgendermaßen:

- p und q sind Teil verschiedener Gruppen
- p liegt am Rand seiner Gruppe

$M_1 = \{1, 2, 3, 5, 6, 7\}$

Paar aus M_1 mit höchstem Ähnlichkeitsmaß: $(3, 6)$

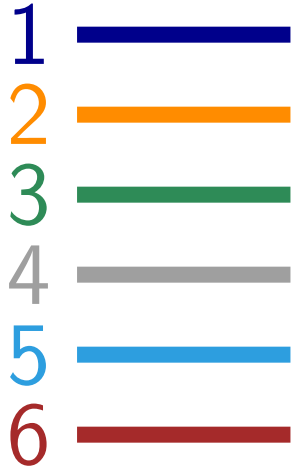


$\Rightarrow$ eindeutige Blockkreuzung führt Charakterpaar und ihre Gruppen zusammen

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

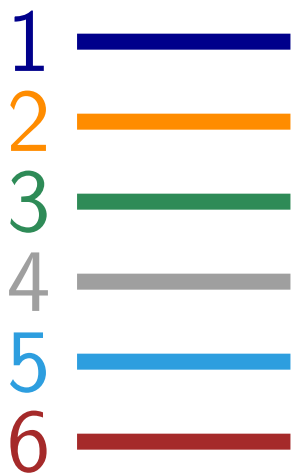
$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$

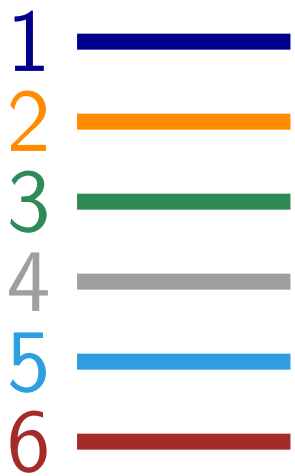


$M_1 = \{2, 3\}$

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



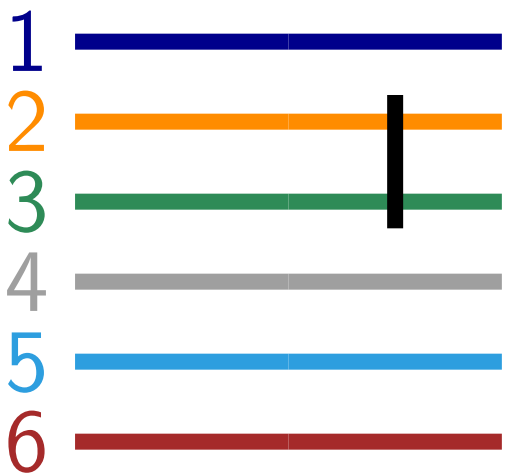
$M_1 = \{2, 3\}$

$\Rightarrow$ kann in der aktuellen Permutation stattfinden

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

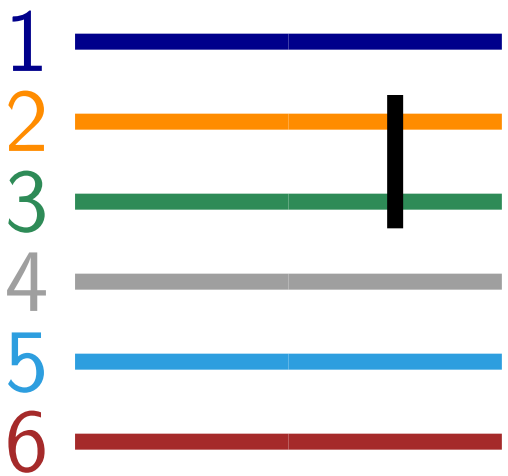
$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$

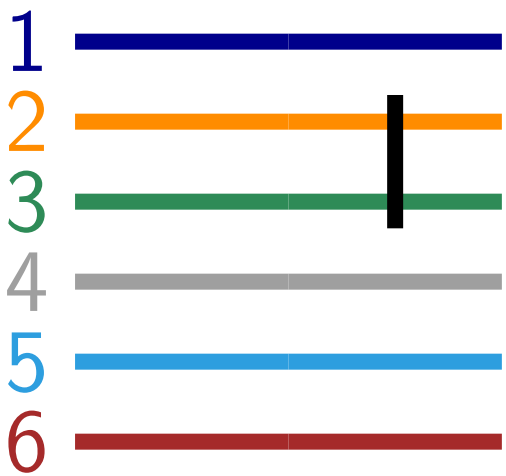


$M_2 = \{3, 4, 5\}$

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



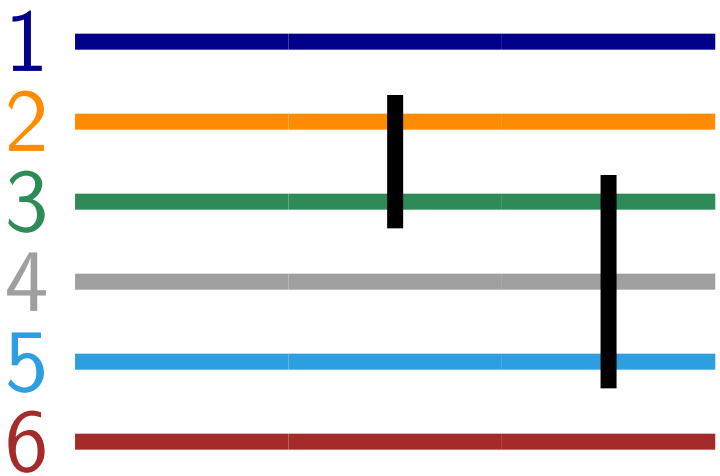
$M_2 = \{3, 4, 5\}$

$\Rightarrow$ kann in der aktuellen Permutation stattfinden

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

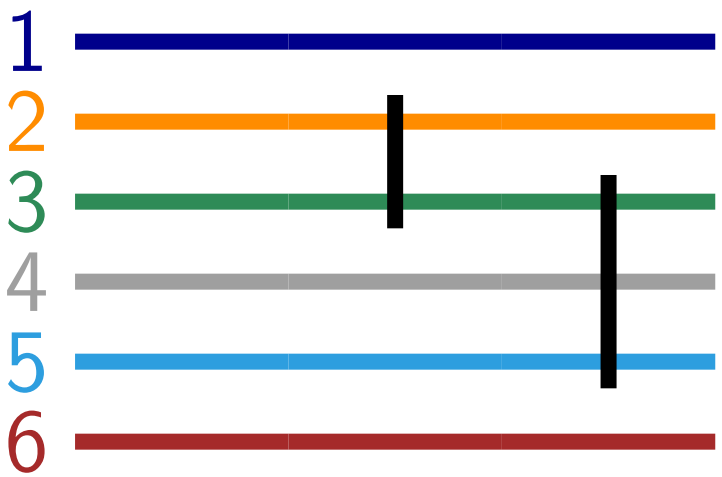
$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$

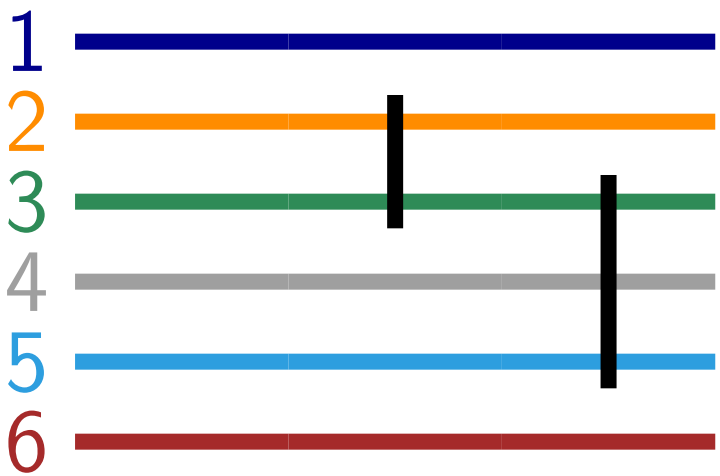


$M_3 = \{1, 2, 4, 5, 6\}$

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



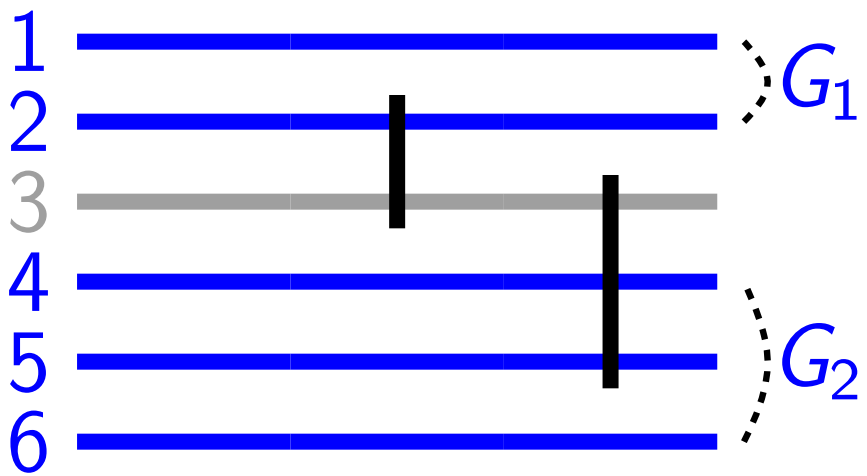
$M_3 = \{1, 2, 4, 5, 6\}$

$\Rightarrow$ kann in aktueller Permutation nicht stattfinden

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



$\Rightarrow$ bilde Gruppen

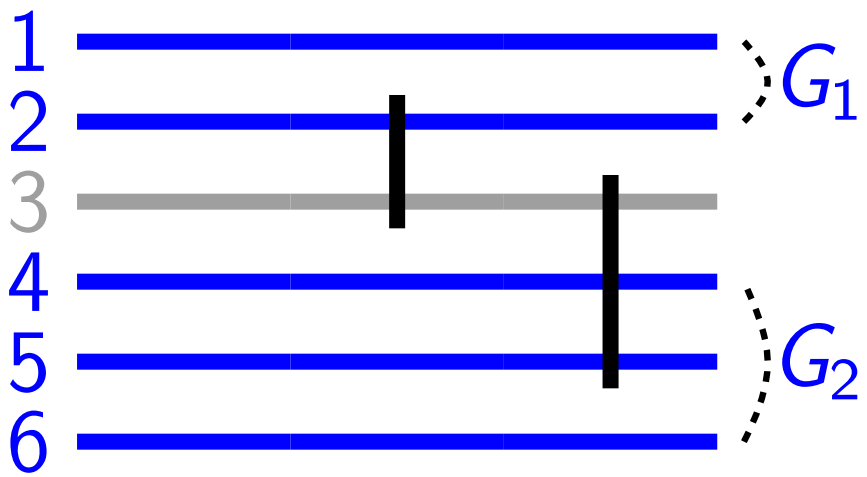
$M_3 = \{1, 2, 4, 5, 6\}$

$\Rightarrow$ kann in aktueller Permutation nicht stattfinden

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



$\Rightarrow$ bilde Gruppen

$\Rightarrow$ berechne Ähnlichkeitsmaße

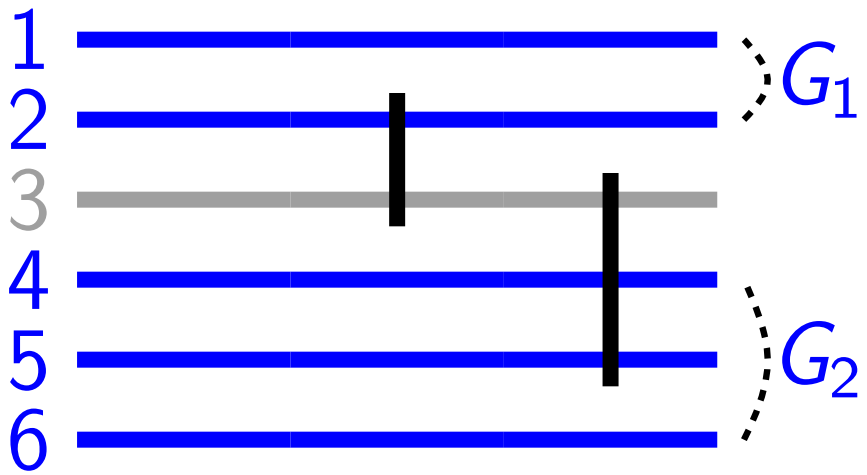
$M_3 = \{1, 2, 4, 5, 6\}$

$\Rightarrow$ kann in aktueller Permutation nicht stattfinden

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



$\Rightarrow$ bilde Gruppen

$\Rightarrow$ berechne Ähnlichkeitsmaße

$M_3 = \{1, 2, 4, 5, 6\}$

$\Rightarrow$ kann in aktueller Permutation nicht stattfinden

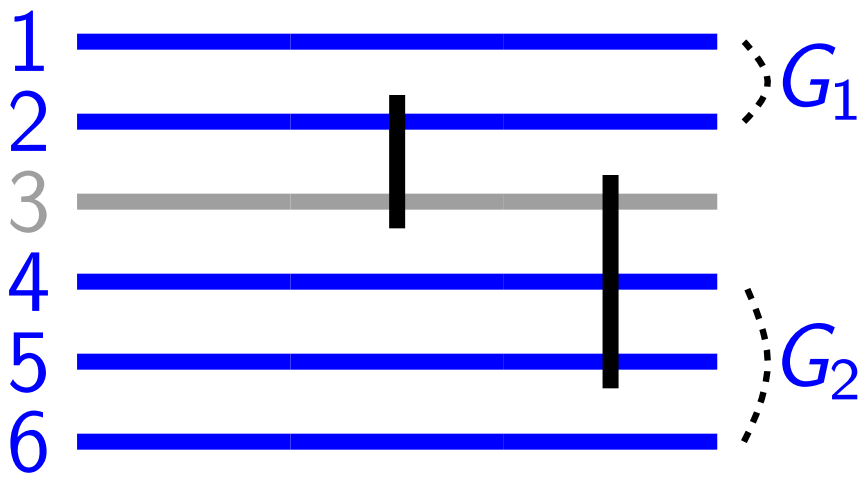
	1	2	4	5	6
1	-	0	0	0	0
2	0	-	0	0	0
4	0	0	-	0	0
5	0	0	0	-	0
6	0	0	0	0	-

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$

Lookahead t



$\Rightarrow$ bilde Gruppen

$\Rightarrow$ berechne Ähnlichkeitsmaße

$M_3 = \{1, 2, 4, 5, 6\}$

$\Rightarrow$ kann in aktueller Permutation nicht stattfinden

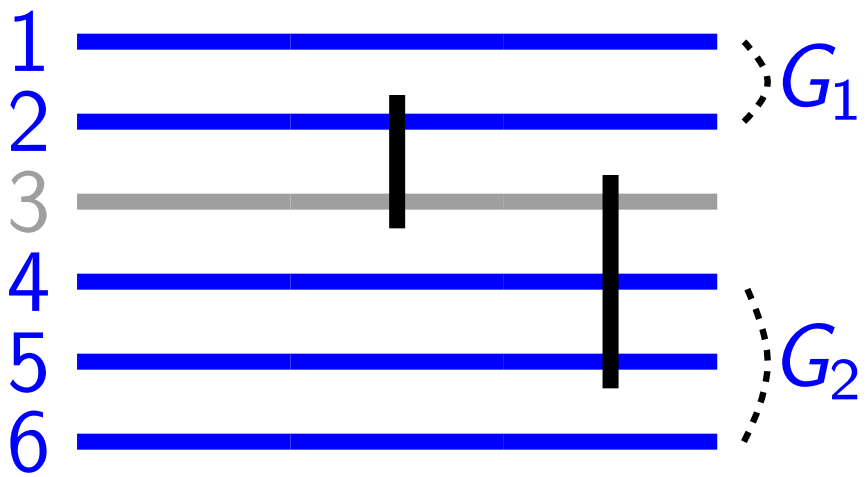
	1	2	4	5	6
1	-	0	0	0	0
2	0	-	0	0	0
4	0	0	-	0	0
5	0	0	0	-	0
6	0	0	0	0	-

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$

1. nachfolgende Treffen



$\Rightarrow$ bilde Gruppen

$\Rightarrow$ berechne Ähnlichkeitsmaße

$+\frac{1}{1}$

	1	2	4	5	6
1	-	0	0	0	0
2	0	-	0	0	0
4	0	0	-	0	0
5	0	0	0	-	0
6	0	0	0	0	-

$M_3 = \{1, 2, 4, 5, 6\}$

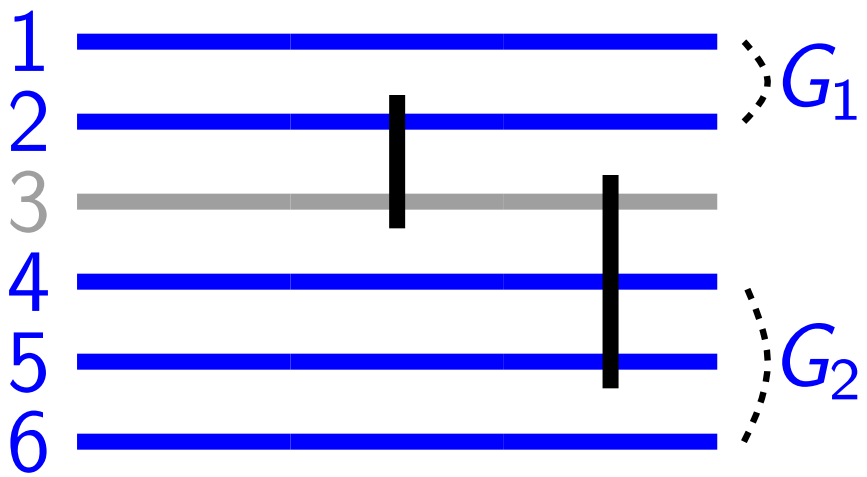
$\Rightarrow$ kann in aktueller Permutation nicht stattfinden

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$

1. nachfolgende Treffen



$\Rightarrow$ bilde Gruppen

$\Rightarrow$ berechne Ähnlichkeitsmaße

$+\frac{1}{1}$

	1	2	4	5	6
1	-	0	0	1	0
2	0	-	0	0	0
4	0	0	-	0	0
5	1	0	0	-	0
6	0	0	0	0	-

$M_3 = \{1, 2, 4, 5, 6\}$

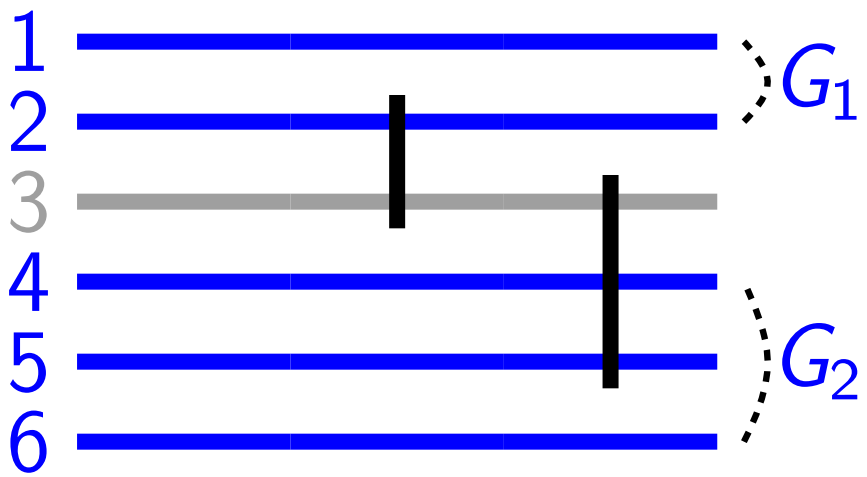
$\Rightarrow$ kann in aktueller Permutation nicht stattfinden

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$

2. nachfolgende Treffen



$\Rightarrow$ bilde Gruppen

$\Rightarrow$ berechne Ähnlichkeitsmaße

$+\frac{1}{2}$

$M_3 = \{1, 2, 4, 5, 6\}$

$\Rightarrow$ kann in aktueller Permutation nicht stattfinden

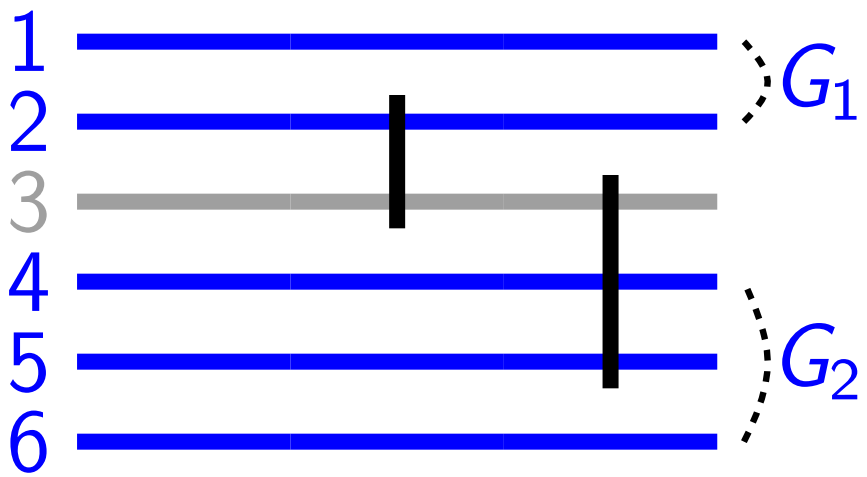
	1	2	4	5	6
1	-	0	0	1	0
2	0	-	0	0	0
4	0	0	-	0	0
5	1	0	0	-	0
6	0	0	0	0	-

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$

2. nachfolgende Treffen



$\Rightarrow$ bilde Gruppen

$\Rightarrow$ berechne Ähnlichkeitsmaße

$+\frac{1}{2}$

$M_3 = \{1, 2, 4, 5, 6\}$

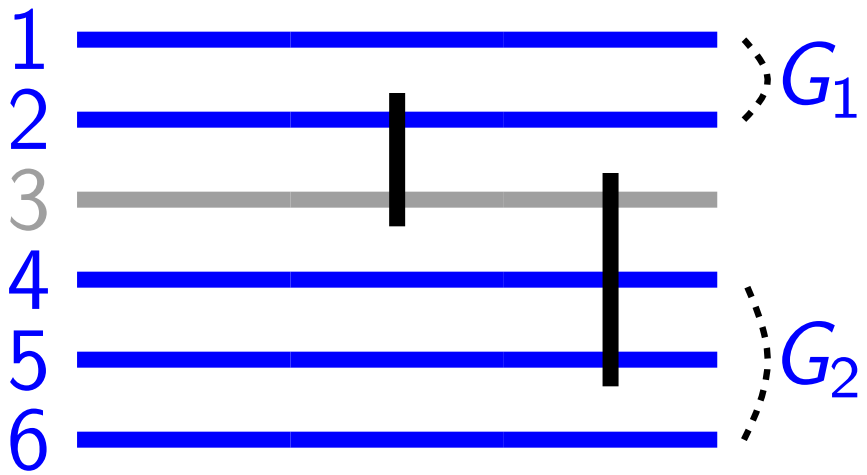
$\Rightarrow$ kann in aktueller Permutation nicht stattfinden

	1	2	4	5	6
1	-	0,5	0	1,5	0
2	0,5	-	0	0,5	0
4	0	0	-	0	0
5	1,5	0,5	0	-	0
6	0	0	0	0	-

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



$\Rightarrow$ bilde Gruppen

$\Rightarrow$ berechne Ähnlichkeitsmaße

$M_3 = \{1, 2, 4, 5, 6\}$

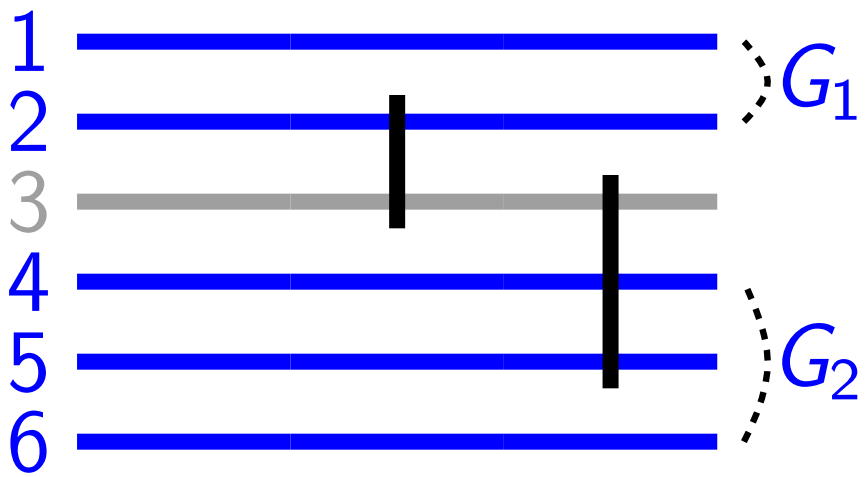
$\Rightarrow$ kann in aktueller Permutation nicht stattfinden

	1	2	4	5	6
1	-	0,5	0	1,5	0
2	0,5	-	0	0,5	0
4	0	0	-	0	0
5	1,5	0,5	0	-	0
6	0	0	0	0	-

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



$\Rightarrow$ bilde Gruppen

$\Rightarrow$ berechne Ähnlichkeitsmaße

$M_3 = \{1, 2, 4, 5, 6\}$

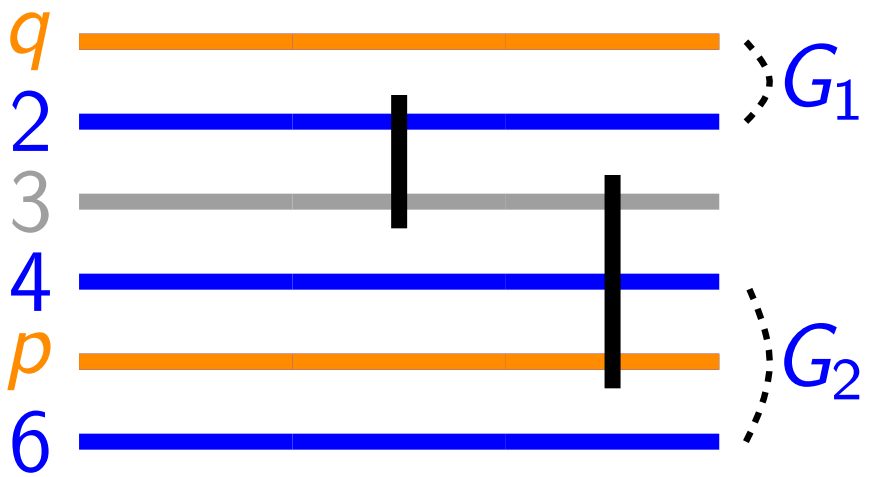
$\Rightarrow$ kann in aktueller Permutation nicht stattfinden

	1	2	4	5	6
1	-	0,5	0	1,5	0
2	0,5	-	0	0,5	0
4	0	0	-	0	0
5	1,5	0,5	0	-	0
6	0	0	0	0	-

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



$\Rightarrow$ bilde Gruppen

$\Rightarrow$ berechne Ähnlichkeitsmaße

	1	2	4	5	6
1	-	0,5	0	1,5	0
2	0,5	-	0	0,5	0
4	0	0	-	0	0
5	1,5	0,5	0	-	0
6	0	0	0	0	-

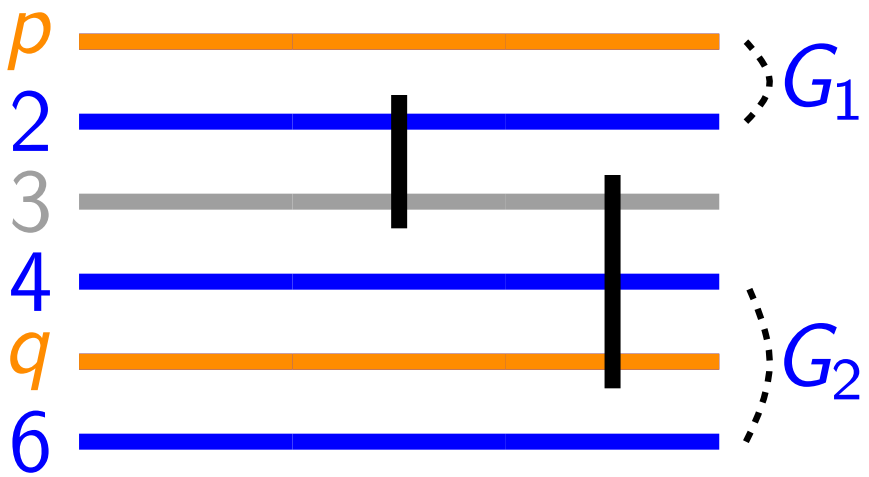
Paar $(5, 1)$:

- unterschiedliche Gruppen ✓
- p am Rand ✗

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



$\Rightarrow$ bilde Gruppen

$\Rightarrow$ berechne Ähnlichkeitsmaße

	1	2	4	5	6
1	-	0,5	0	1,5	0
2	0,5	-	0	0,5	0
4	0	0	-	0	0
5	1,5	0,5	0	-	0
6	0	0	0	0	-

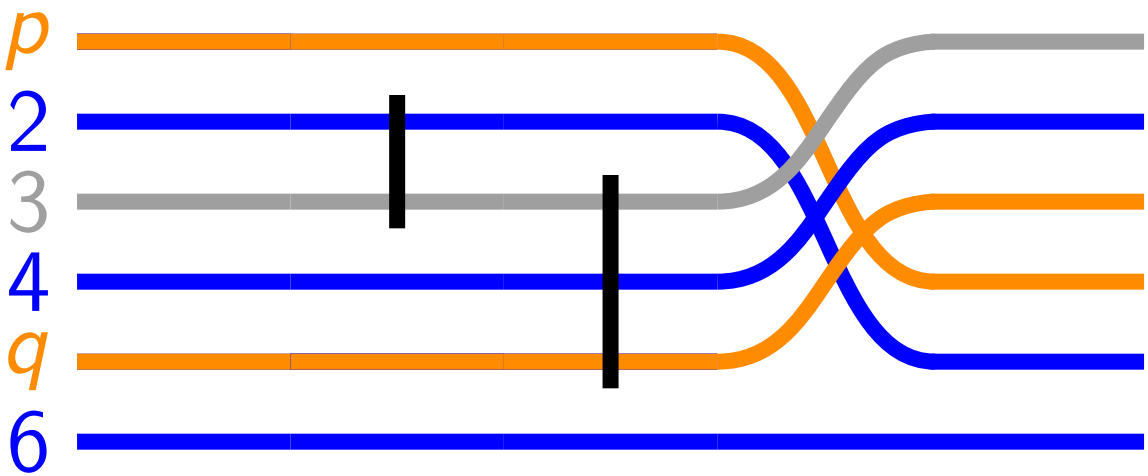
Paar $(1, 5)$:

- unterschiedliche Gruppen ✓
- p am Rand ✓

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



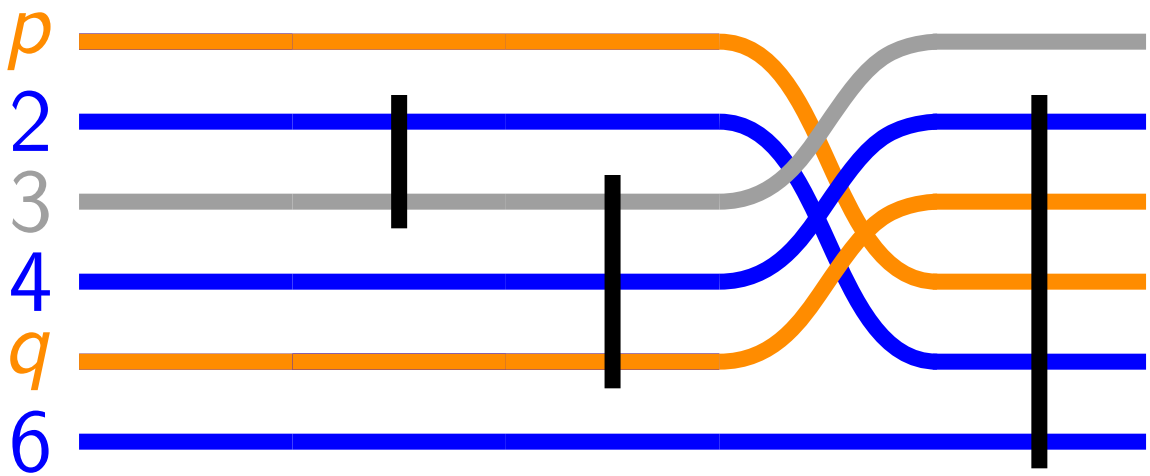
$M_3 = \{1, 2, 4, 5, 6\}$

$\Rightarrow$ kann in aktueller Permutation nicht stattfinden

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



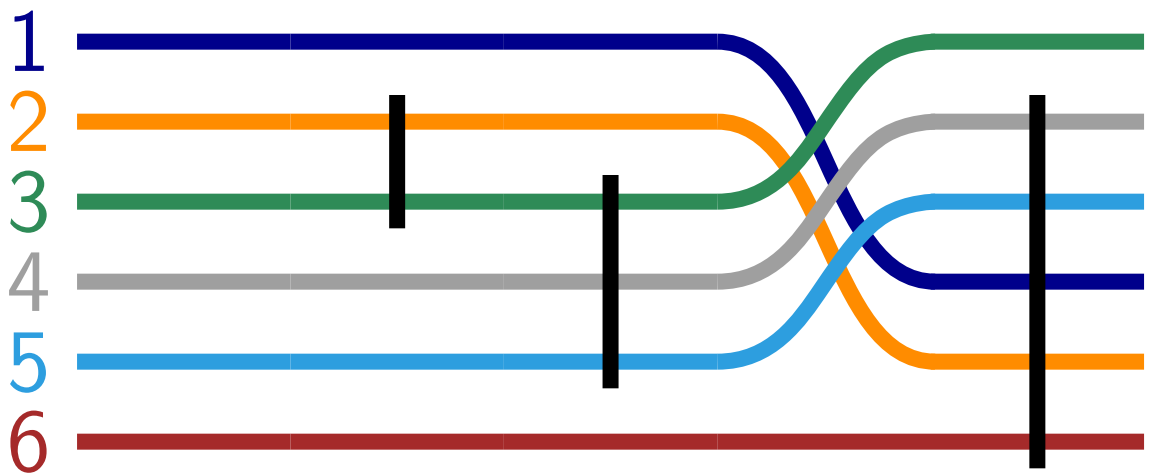
$M_3 = \{1, 2, 4, 5, 6\}$

$\Rightarrow$ kann in der aktuellen Permutation stattfinden

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

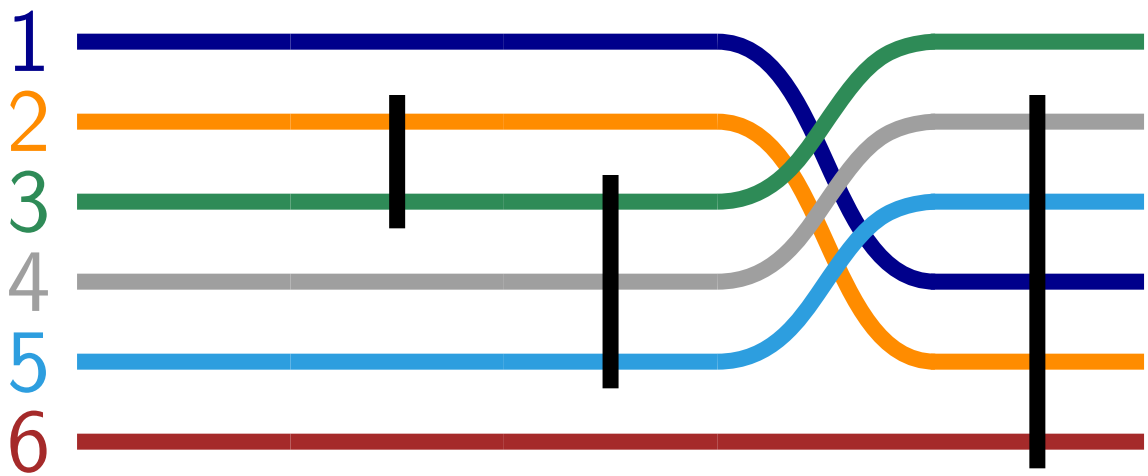
$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



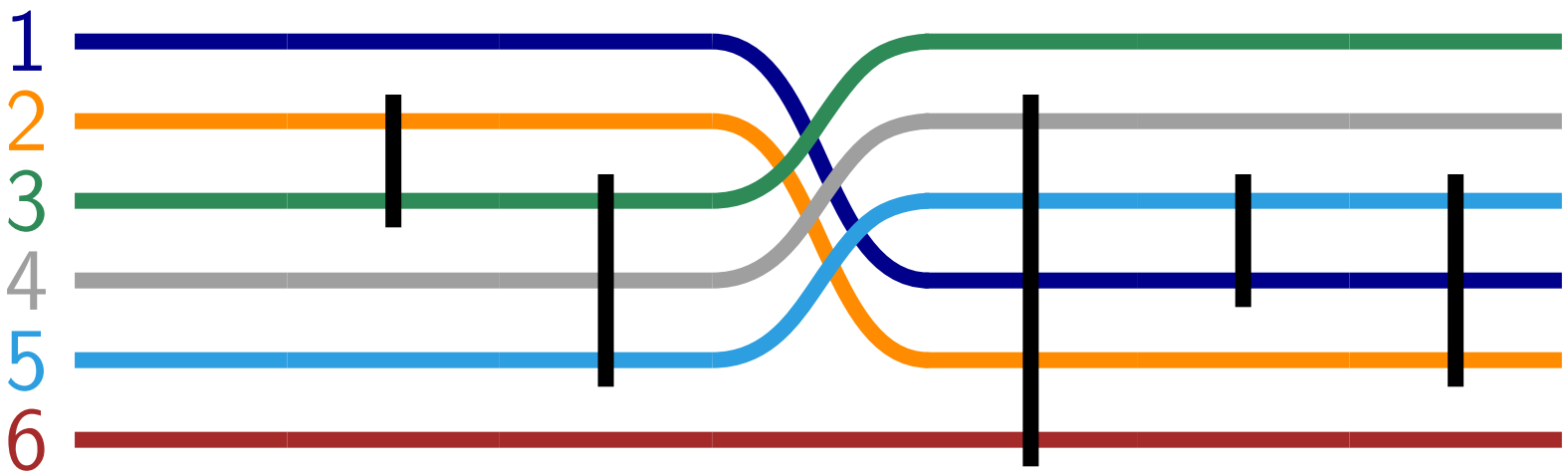
M_4 und M_5

$\Rightarrow$ können in der aktuellen Permutation stattfinden

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



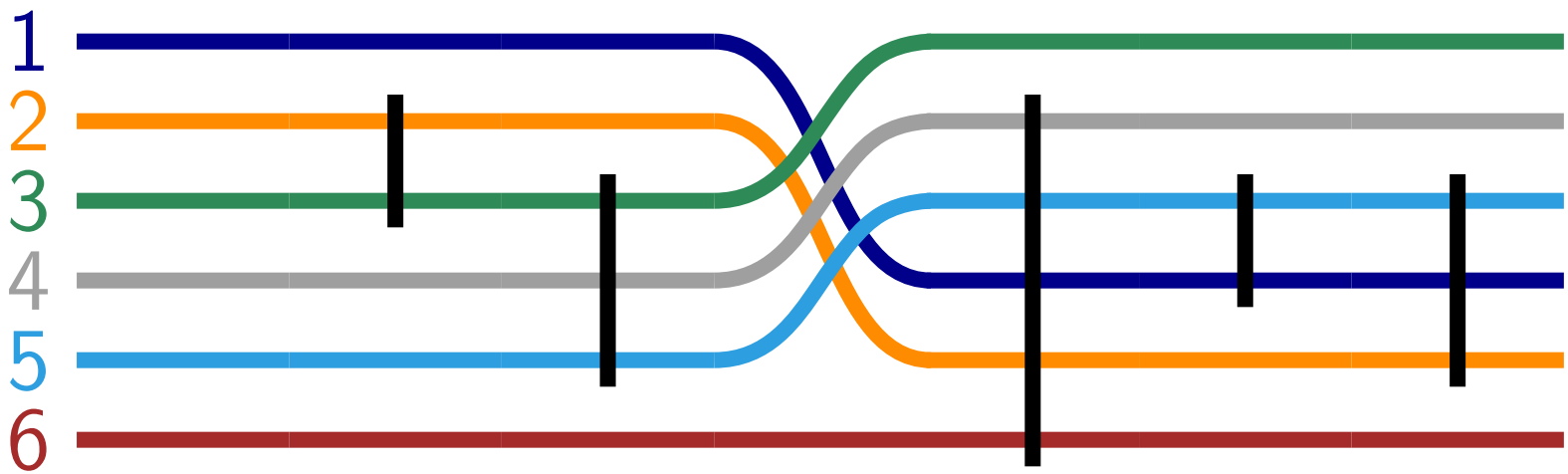
M_4 und M_5

$\Rightarrow$ können in der aktuellen Permutation stattfinden

Beispiel

$C = \{1, 2, 3, 4, 5, 6\}$

$M = (\{2, 3\}, \{3, 4, 5\}, \{1, 2, 4, 5, 6\}, \{1, 5\}, \{1, 2, 5\})$



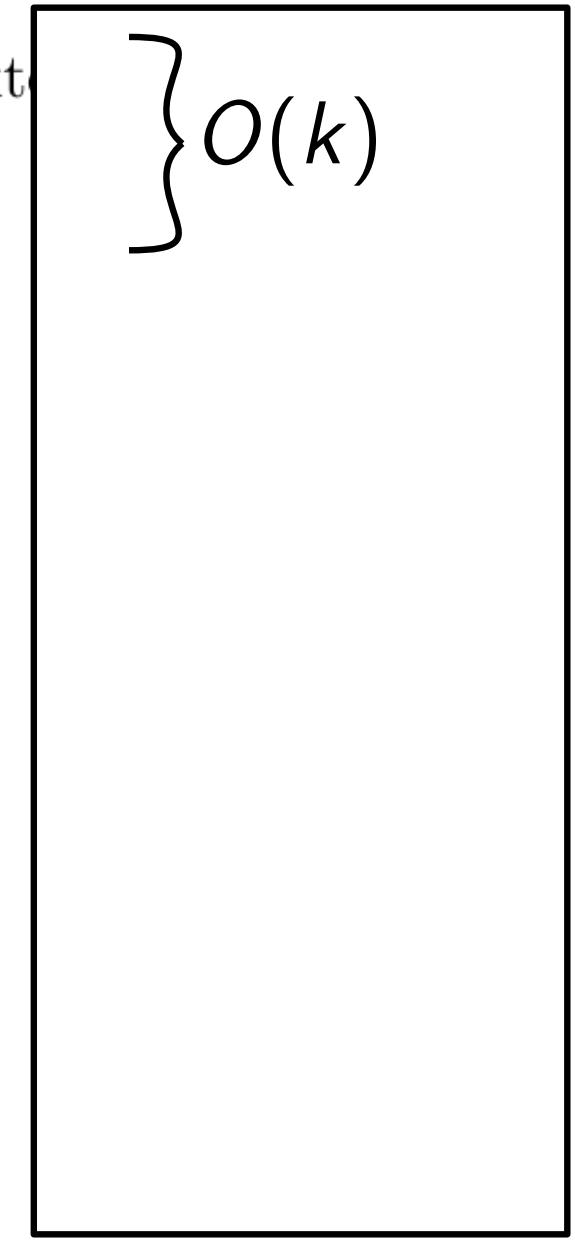
$\Rightarrow$ alle Treffen sind abgearbeitet

Laufzeit

```
1  erstelle zufällige Startpermutation π der Charaktere
2  erstelle leere Liste von Blockkreuzungen B
3  i=1
4  while i ≤ M.length do
5      if isValidMeeting(π, M_i) then
6          i++
7      else
8          A = buildSimilarityMatrix(π, M, i, t)
9          while !isValidMeeting(π, M_i) do
10             G = buildGroups(π, M_i)
11             p = findPair(π, G, A)
12             b = findBlockCrossing(p, π, G)
13             B.append(b)
14             π = updatePermutation(π, b)
15 return (π, B)
```

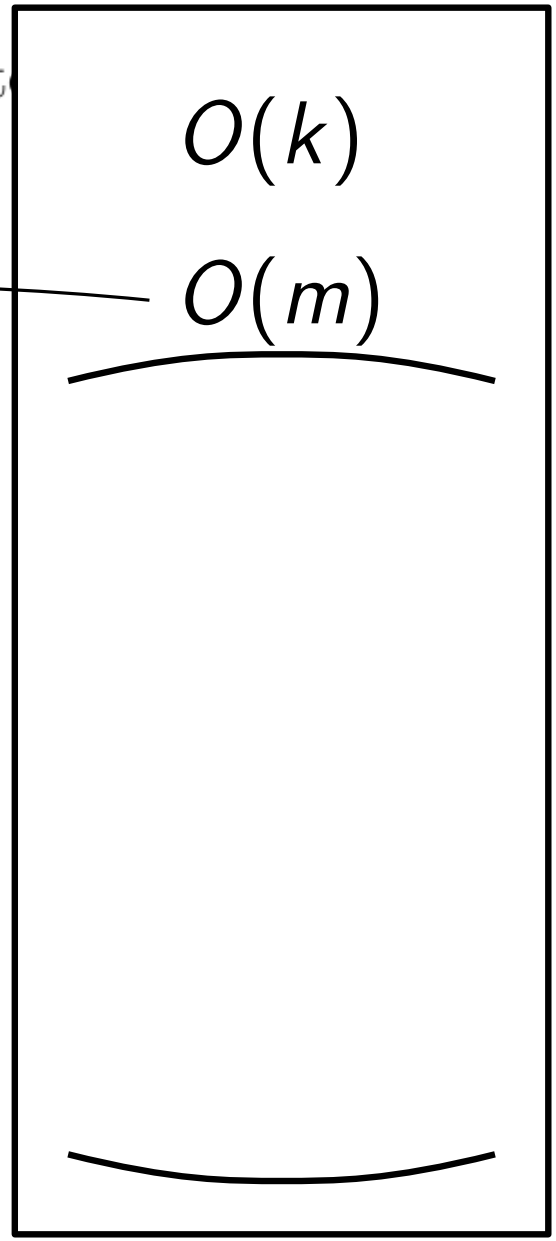
Laufzeit

```
1  erstelle zufällige Startpermutation π der Charakt
2  erstelle leere Liste von Blockkreuzungen B
3  i=1
4  while i ≤ M.length do
5      if isValidMeeting(π, M_i) then
6          i++
7      else
8          A = buildSimilarityMatrix(π, M, i, t)
9          while !isValidMeeting(π, M_i) do
10             G = buildGroups(π, M_i)
11             p = findPair(π, G, A)
12             b = findBlockCrossing(p, π, G)
13             B.append(b)
14             π = updatePermutation(π, b)
15 return (π, B)
```



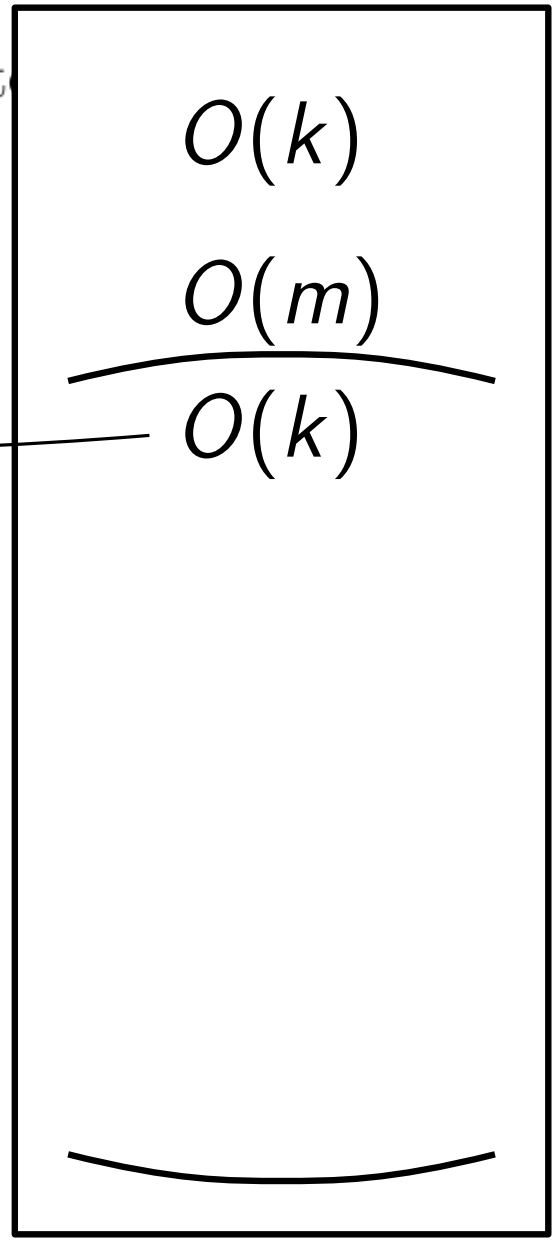
Laufzeit

```
1  erstelle zufällige Startpermutation π der Charakt
2  erstelle leere Liste von Blockkreuzungen B
3  i=1
4  while i ≤ M.length do
5      if isValidMeeting(π, M_i) then
6          i++
7      else
8          A = buildSimilarityMatrix(π, M, i, t)
9          while !isValidMeeting(π, M_i) do
10             G = buildGroups(π, M_i)
11             p = findPair(π, G, A)
12             b = findBlockCrossing(p, π, G)
13             B.append(b)
14             π = updatePermutation(π, b)
15 return (π, B)
```



Laufzeit

```
1  erstelle zufällige Startpermutation π der Charakt
2  erstelle leere Liste von Blockkreuzungen B
3  i=1
4  while i ≤ M.length do
5      if isValidMeeting(π, M_i) then
6          i++
7      else
8          A = buildSimilarityMatrix(π, M, i, t)
9          while !isValidMeeting(π, M_i) do
10             G = buildGroups(π, M_i)
11             p = findPair(π, G, A)
12             b = findBlockCrossing(p, π, G)
13             B.append(b)
14             π = updatePermutation(π, b)
15 return (π, B)
```



Laufzeit

```
1  erstelle zufällige Startpermutation π der Charakt
2  erstelle leere Liste von Blockkreuzungen B
3  i=1
4  while i ≤ M.length do
5      if isValidMeeting(π, M_i) then
6          i++
7      else
8          A = buildSimilarityMatrix(π, M, i, t)
9          while !isValidMeeting(π, M_i) do
10             G = buildGroups(π, M_i)
11             p = findPair(π, G, A)
12             b = findBlockCrossing(p, π, G)
13             B.append(b)
14             π = updatePermutation(π, b)
15 return (π, B)
```

$O(k)$

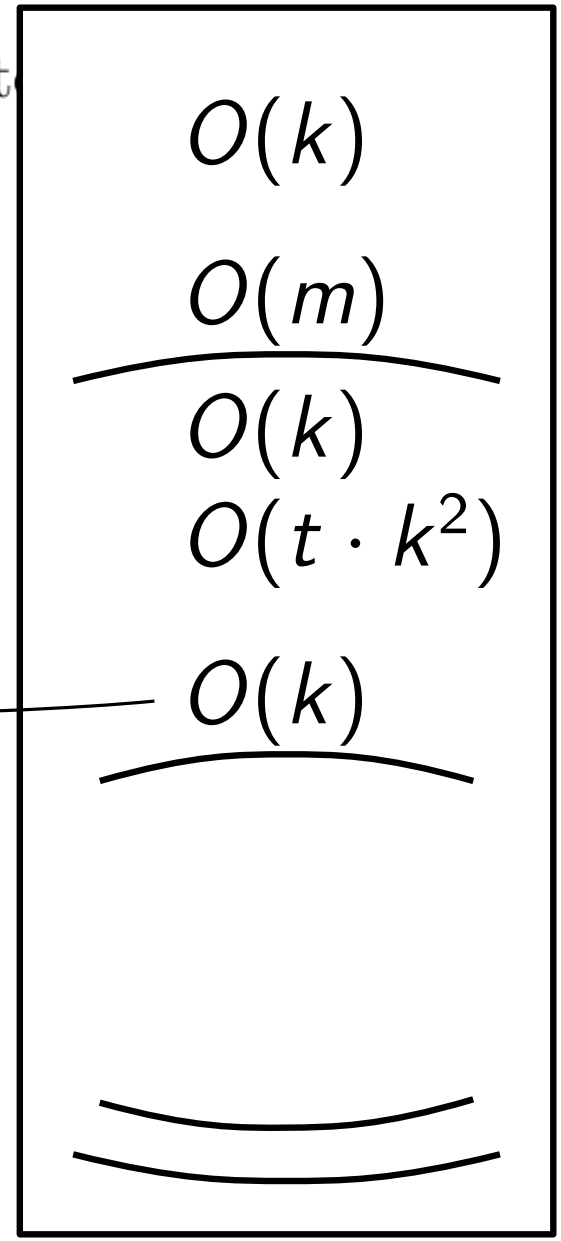
$O(m)$

$O(k)$

$O(t \cdot k^2)$

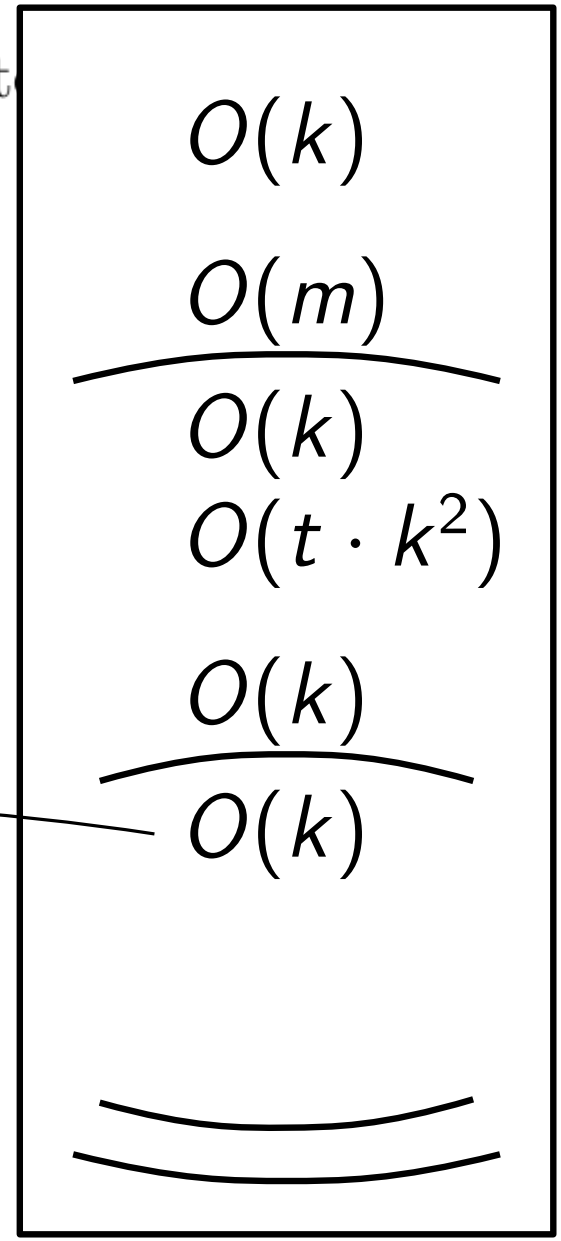
Laufzeit

```
1  erstelle zufällige Startpermutation π der Charakt
2  erstelle leere Liste von Blockkreuzungen B
3  i=1
4  while i ≤ M.length do
5      if isValidMeeting(π, M_i) then
6          i++
7      else
8          A = buildSimilarityMatrix(π, M, i, t)
9          while !isValidMeeting(π, M_i) do
10             G = buildGroups(π, M_i)
11             p = findPair(π, G, A)
12             b = findBlockCrossing(p, π, G)
13             B.append(b)
14             π = updatePermutation(π, b)
15 return (π, B)
```



Laufzeit

```
1  erstelle zufällige Startpermutation π der Charakt
2  erstelle leere Liste von Blockkreuzungen B
3  i=1
4  while i ≤ M.length do
5      if isValidMeeting(π, M_i) then
6          i++
7      else
8          A = buildSimilarityMatrix(π, M, i, t)
9          while !isValidMeeting(π, M_i) do
10             G = buildGroups(π, M_i)
11             p = findPair(π, G, A)
12             b = findBlockCrossing(p, π, G)
13             B.append(b)
14             π = updatePermutation(π, b)
15 return (π, B)
```



Laufzeit

```
1  erstelle zufällige Startpermutation π der Charakt
2  erstelle leere Liste von Blockkreuzungen B
3  i=1
4  while i ≤ M.length do
5      if isValidMeeting(π, M_i) then
6          i++
7      else
8          A = buildSimilarityMatrix(π, M, i, t)
9          while !isValidMeeting(π, M_i) do
10             G = buildGroups(π, M_i)
11             p = findPair(π, G, A)
12             b = findBlockCrossing(p, π, G)
13             B.append(b)
14             π = updatePermutation(π, b)
15 return (π, B)
```

$O(k)$

$O(m)$

$O(k)$

$O(t \cdot k^2)$

$O(k)$

$O(k)$

$O(k^2)$

Laufzeit

```
1  erstelle zufällige Startpermutation π der Charakt
2  erstelle leere Liste von Blockkreuzungen B
3  i=1
4  while i ≤ M.length do
5      if isValidMeeting(π, M_i) then
6          i++
7      else
8          A = buildSimilarityMatrix(π, M, i, t)
9          while !isValidMeeting(π, M_i) do
10             G = buildGroups(π, M_i)
11             p = findPair(π, G, A)
12             b = findBlockCrossing(p, π, G)
13             B.append(b)
14             π = updatePermutation(π, b)
15 return (π, B)
```

$O(k)$

$O(m)$

$O(k)$

$O(t \cdot k^2)$

$O(k)$

$O(k)$

$O(k^2)$

$O(k)$

Laufzeit

```
1  erstelle zufällige Startpermutation π der Charakt
2  erstelle leere Liste von Blockkreuzungen B
3  i=1
4  while i ≤ M.length do
5      if isValidMeeting(π, M_i) then
6          i++
7      else
8          A = buildSimilarityMatrix(π, M, i, t)
9          while !isValidMeeting(π, M_i) do
10             G = buildGroups(π, M_i)
11             p = findPair(π, G, A)
12             b = findBlockCrossing(p, π, G)
13             B.append(b)
14             π = updatePermutation(π, b)
15 return (π, B)
```

$O(k)$

$O(m)$

$O(k)$

$O(t \cdot k^2)$

$O(k^3)$

Laufzeit

```
1  erstelle zufällige Startpermutation π der Charakt
2  erstelle leere Liste von Blockkreuzungen B
3  i=1
4  while i ≤ M.length do
5      if isValidMeeting(π, M_i) then
6          i++
7      else
8          A = buildSimilarityMatrix(π, M, i, t)
9          while !isValidMeeting(π, M_i) do
10             G = buildGroups(π, M_i)
11             p = findPair(π, G, A)
12             b = findBlockCrossing(p, π, G)
13             B.append(b)
14             π = updatePermutation(π, b)
15 return (π, B)
```

$O(k)$

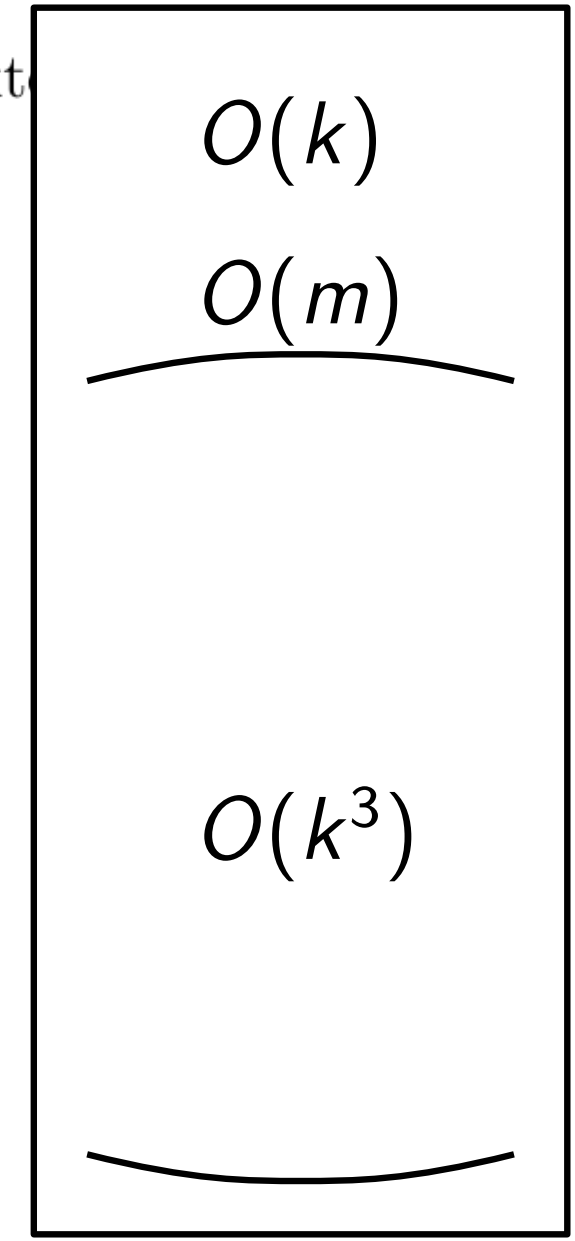
$O(m)$

$O(t \cdot k^2)$

$O(k^3)$

Laufzeit

```
1  erstelle zufällige Startpermutation π der Charakt
2  erstelle leere Liste von Blockkreuzungen B
3  i=1
4  while i ≤ M.length do
5      if isValidMeeting(π, M_i) then
6          i++
7      else
8          A = buildSimilarityMatrix(π, M, i, t)
9          while !isValidMeeting(π, M_i) do
10             G = buildGroups(π, M_i)
11             p = findPair(π, G, A)
12             b = findBlockCrossing(p, π, G)
13             B.append(b)
14             π = updatePermutation(π, b)
15 return (π, B)
```



Laufzeit

```
1  erstelle zufällige Startpermutation π der Charakt
2  erstelle leere Liste von Blockkreuzungen B
3  i=1
4  while i ≤ M.length do
5      if isValidMeeting(π, M_i) then
6          i++
7      else
8          A = buildSimilarityMatrix(π, M, i, t)
9          while !isValidMeeting(π, M_i) do
10             G = buildGroups(π, M_i)
11             p = findPair(π, G, A)
12             b = findBlockCrossing(p, π, G)
13             B.append(b)
14             π = updatePermutation(π, b)
15 return (π, B)
```

$O(mk^3)$

Startpermutationen

- GreedyStartRandom
  bisher verwendete Version mit einer zufälligen Startpermutation

Startpermutationen

- GreedyStartRandom
  bisher verwendete Version mit einer zufälligen Startpermutation

- GreedyStartHeuristic
  verwendet naive Heuristik um Startpermutation festzulegen

Startpermutationen

- GreedyRepetitionRandom
  wiederholt den Algorithmus zehn mal mit jeweils zufälliger Startpermutation und gibt bestes Ergebnis zurück

Startpermutationen

- GreedyRepetitionRandom
  wiederholt den Algorithmus zehn mal mit jeweils zufälliger Startpermutation und gibt bestes Ergebnis zurück

- GreedyRepetitionAll
  führt den Algorithmus mit jeder möglichen Startpermutation durch und gibt bestes Ergebnis zurück

Gliederung

1. Grundlagen
2. Greedy-Algorithmus
3. **Auswertung und Fazit**

Auswertung

wird anhand von Beispielen aus `dblp` durchgeführt:

Auswertung

wird anhand von Beispielen aus `dblp` durchgeführt:

- die vier Varianten und ein FTP-Algorithmus werden getestet

Auswertung

wird anhand von Beispielen aus `dblp` durchgeführt:

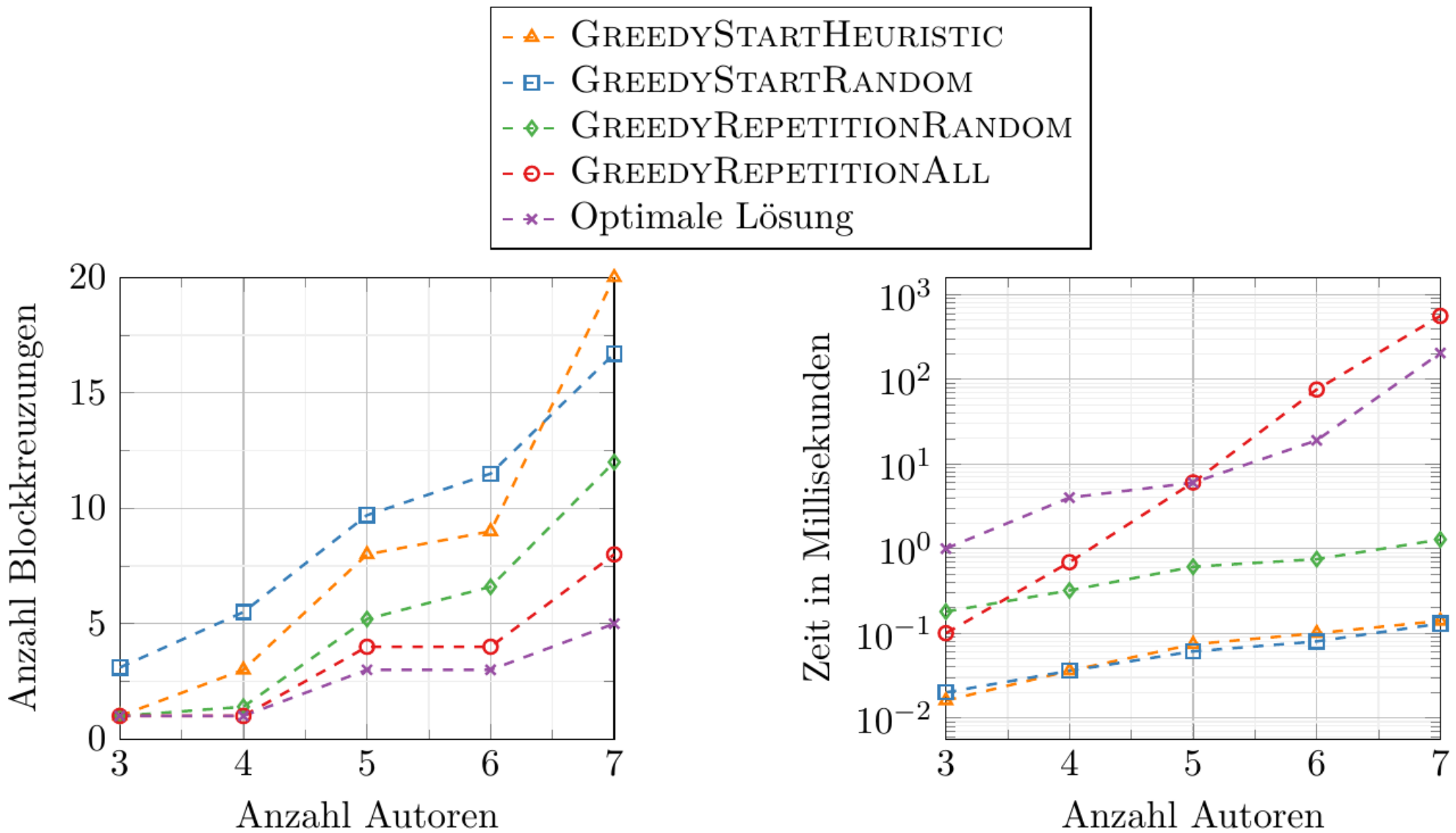
- die vier Varianten und ein FTP-Algorithmus werden getestet
- es wird die Anzahl an Blockkreuzungen und die Berechnungsdauer

Auswertung

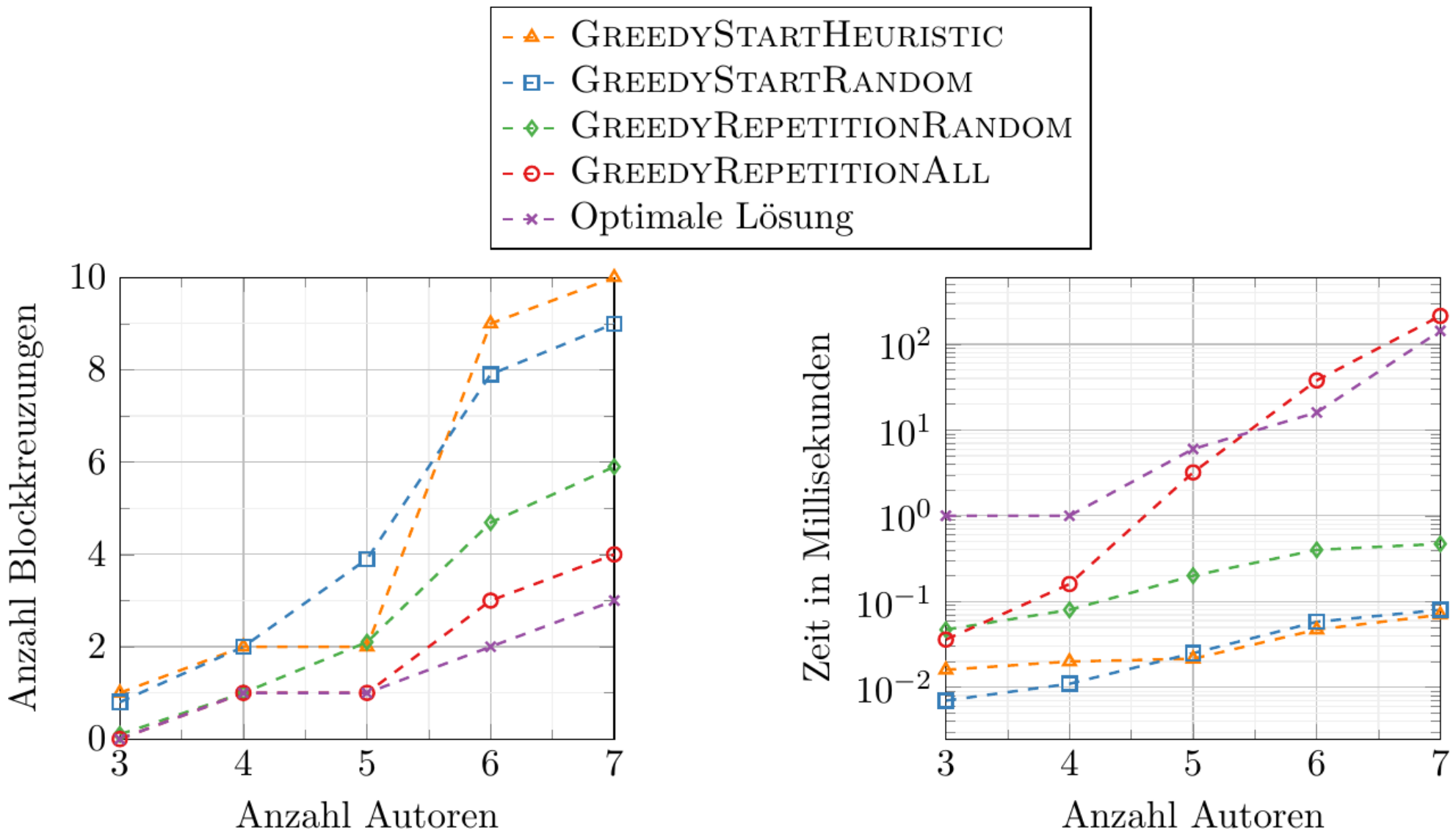
wird anhand von Beispielen aus `dblp` durchgeführt:

- die vier Varianten und ein FTP-Algorithmus werden getestet
- es wird die Anzahl an Blockkreuzungen und die Berechnungsdauer
- die zufälligen Verfahren wurden über 1000 Wiederholungen gemittelt

Auswertung


GreedyStartHeuristic
GreedyStartRandom
GreedyRepetitionRandom
GreedyRepetitionAll
Optimale Lösung
Anzahl Blockkreuzungen
0
5
10
15
20
3
4
5
6
7
Anzahl Autoren
Zeit in Millisekunden
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
3
4
5
6
7
Anzahl Autoren

Auswertung


GreedyStartHeuristic
GreedyStartRandom
GreedyRepetitionRandom
GreedyRepetitionAll
Optimale Lösung
Anzahl Blockkreuzungen
0
2
4
6
8
10
3
4
5
6
7
Anzahl Autoren
Zeit in Millisekunden
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
3
4
5
6
7
Anzahl Autoren

Fazit

- Varianten, die den Greedy-Algorithmus nur einmal durchführen, sind zwar sehr schnell, erzeugen aber noch sehr viele Blockkreuzungen

Fazit

- Varianten, die den Greedy-Algorithmus nur einmal durchführen, sind zwar sehr schnell, erzeugen aber noch sehr viele Blockkreuzungen
- bei zehn Wiederholungen mit je zufälliger Startpermutation sind meist deutliche Verbesserungen bemerkbar

Fazit

- Varianten, die den Greedy-Algorithmus nur einmal durchführen, sind zwar sehr schnell, erzeugen aber noch sehr viele Blockkreuzungen
- bei zehn Wiederholungen mit je zufälliger Startpermutation sind meist deutliche Verbesserungen bemerkbar
- werden alle möglichen Startpermutationen ausprobiert sind die Ergebnisse zwar verhältnismäßig gut, die Berechnungszeit ist aber sehr hoch

Ausblick

- Startpermutation besser wählen

Ausblick

- Startpermutation besser wählen
  - bessere Heuristik

Ausblick

- Startpermutation besser wählen
  - bessere Heuristik
  - Kombination von Heuristiken

Ausblick

- Startpermutation besser wählen
  - bessere Heuristik
  - Kombination von Heuristiken
  - Pruning

Ausblick

- Startpermutation besser wählen
  - bessere Heuristik
  - Kombination von Heuristiken
  - Pruning
- mehr Blockkreuzungen betrachten

Ausblick

- Startpermutation besser wählen
  - bessere Heuristik
  - Kombination von Heuristiken
  - Pruning
- mehr Blockkreuzungen betrachten
  - alle möglichen Blockkreuzungen

Ausblick

- Startpermutation besser wählen
  - bessere Heuristik
  - Kombination von Heuristiken
  - Pruning
- mehr Blockkreuzungen betrachten
  - alle möglichen Blockkreuzungen
  - betrachte Charaktere außerhalb Gruppen

Anhang

Treffen M_i kann nicht stattfinden

$\Rightarrow$ das Ähnlichkeitsmaß aller Charakterpaare aus M_u werden um $\frac{1}{u-i}$ erhöht (für $u \in [i+1, i+t]$)

Anhang

Treffen M_i kann nicht stattfinden

$\Rightarrow$ das Ähnlichkeitsmaß aller Charakterpaare aus M_u werden um $\frac{1}{u-i}$ erhöht (für $u \in [i+1, i+t]$)

Beispiel:

Treffen im Lookahead:

1.) $[r, s]$

2.) $[u, v]$

3.) $[x, y, w]$

4.) $[x, y, z]$

Anhang

Treffen M_i kann nicht stattfinden

$\Rightarrow$ das Ähnlichkeitsmaß aller Charakterpaare aus M_u werden um $\frac{1}{u-i}$ erhöht (für $u \in [i+1, i+t]$)

Beispiel:

Treffen im Lookahead:

1.) $[r, s]$

2.) $[u, v]$

3.) $[x, y, w]$

4.) $[x, y, z]$

$\Rightarrow a_{u,v} = \frac{1}{2} = \frac{6}{12}$

$\Rightarrow a_{x,y} = \frac{1}{3} + \frac{1}{4} = \frac{7}{12}$

Anhang

Treffen M_i kann nicht stattfinden

$\Rightarrow$ das Ähnlichkeitsmaß aller Charakterpaare aus M_u werden um $\frac{1}{u-i}$ erhöht (für $u \in [i+1, i+t]$)

Beispiel:

Treffen im Lookahead:

1.) $[r, s]$

2.) $[u, v]$

3.) $[x, y, w]$

4.) $[x, y, z]$

$\Rightarrow a_{u,v} = \frac{1}{2} = \frac{6}{12}$

$\Rightarrow a_{x,y} = \frac{1}{3} + \frac{1}{4} = \frac{7}{12}$

$\Rightarrow$ Paar (x, y) wird eher gewählt als (u, v)

Anhang

Treffen M_i kann nicht stattfinden

$\Rightarrow$ das Ähnlichkeitsmaß aller Charakterpaare aus M_u werden um $\frac{1}{u-i}$ erhöht (für $u \in [i+1, i+t]$)

Beispiel:

Treffen im Lookahead:

1.) $[u, v]$

2.) $[x, y, w]$

3.) $[x, y, z]$

4.) $[r, s]$

Anhang

Treffen M_i kann nicht stattfinden

$\Rightarrow$ das Ähnlichkeitsmaß aller Charakterpaare aus M_u werden um $\frac{1}{u-i}$ erhöht (für $u \in [i+1, i+t]$)

Beispiel:

Treffen im Lookahead:

1.) $[u, v]$ $\Rightarrow a_{u,v} = \frac{1}{1} = \frac{12}{12}$

2.) $[x, y, w]$ $\Rightarrow a_{x,y} = \frac{1}{2} + \frac{1}{3} = \frac{9}{12}$

3.) $[x, y, z]$

4.) $[r, s]$

Anhang

Treffen M_i kann nicht stattfinden

$\Rightarrow$ das Ähnlichkeitsmaß aller Charakterpaare aus M_u werden um $\frac{1}{u-i}$ erhöht (für $u \in [i+1, i+t]$)

Beispiel:

Treffen im Lookahead:

1.) $[u, v]$

2.) $[x, y, w]$

3.) $[x, y, z]$

4.) $[r, s]$

$\Rightarrow a_{u,v} = \frac{1}{1} = \frac{12}{12}$

$\Rightarrow a_{x,y} = \frac{1}{2} + \frac{1}{3} = \frac{9}{12}$

$\Rightarrow$ Paar (u, v) wird jetzt eher gewählt als (x, y)

Anhang

Treffen M_i kann nicht stattfinden

$\Rightarrow$ das Ähnlichkeitsmaß aller Charakterpaare aus M_u werden um $\frac{1}{u-i}$ erhöht (für $u \in [i+1, i+t]$)

$\Rightarrow$ Priorisierung kann sich ändern, obwohl:

- das Treffen, das nicht mehr beachtet wird, nicht relevant ist
- das neue Treffen nicht relevant ist

Anhang

Treffen M_i kann nicht stattfinden

$\Rightarrow$ das Ähnlichkeitsmaß aller Charakterpaare aus M_u werden um $\frac{1}{u-i}$ erhöht (für $u \in [i+1, i+t]$)

$\Rightarrow$ Priorisierung kann sich ändern, obwohl:

- das Treffen, das nicht mehr beachtet wird, nicht relevant ist
- das neue Treffen nicht relevant ist

Erhöhen das Ähnlichkeitsmaß um $\frac{1}{2^{(u-i)}}$ löst diese Problematik

Anhang

$\Rightarrow$ mit dieser Gewichtsfunktion könnten wir in jedem Schritt eine Art Prioritätsschlange aktualisieren

Anhang

$\Rightarrow$ mit dieser Gewichtsfunktion könnten wir in jedem Schritt eine Art Prioritätsschlange aktualisieren

Problematik 1:
Das aktuelle beste Paar zu kennen reicht nicht aus, da beliebige Paare nicht immer die benötigten Bedingungen erfüllen

Anhang

$\Rightarrow$ mit dieser Gewichtsfunktion könnten wir in jedem Schritt eine Art Prioritätsschlange aktualisieren

Problematik 1:
Das aktuelle beste Paar zu kennen reicht nicht aus, da beliebige Paare nicht immer die benötigten Bedingungen erfüllen

Problematik 2:
Es wird nicht in jedem Schritt eine Blockkreuzung benötigt