Bachelorarbeit

Optimale Zeichnungen von Storylines mit Blockkreuzungen

Peter Markfelder

Abgabedatum:	23. Februar 2017
Betreuer:	Prof. Dr. Alexander Wolff
	Fabian Lipp
	Dr. Thomas van Dijk

Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen, Komplexität und wissensbasierte Systeme

Zusammenfassung

Eine Storyline-Visualisierung ist eine Darstellung von Charakteren beispielsweise aus einem Film oder Buch, welche durch x-monotone Linien repräsentiert werden. Dabei ist auf der x-Achse die Zeit aufgetragen. Um eine soziale Interaktion der Charaktere darzustellen, werden die betroffenen Linien für den Zeitraum der Interaktion zusammengeschoben, d. h. es befindet sich keine Linie, welche nicht Teil der Interaktion ist, zwischen den Linien dieser. Ziel ist es die Anzahl der Blockkreuzungen der Linien zu minimieren. Dieses Problem ist NP-schwer, weswegen zur Lösung ein ganzzahlig lineares Programm und eine Formulierung als SAT-Instanz verwendet werden soll. Gronemann et al. [5] lösten mithilfe eines ILPs das Problem bereits für Einzelkreuzungen optimal. Wir verwenden zuerst eine für Blockkreuzungen angepasste Version ihres Modells und ihres ILPs. Danach wird noch ein eigenes Modell entwickelt, für welches ein zweites ILP und eine Formulierung als SAT-Instanz aufgestellt wird.

Inhaltsverzeichnis

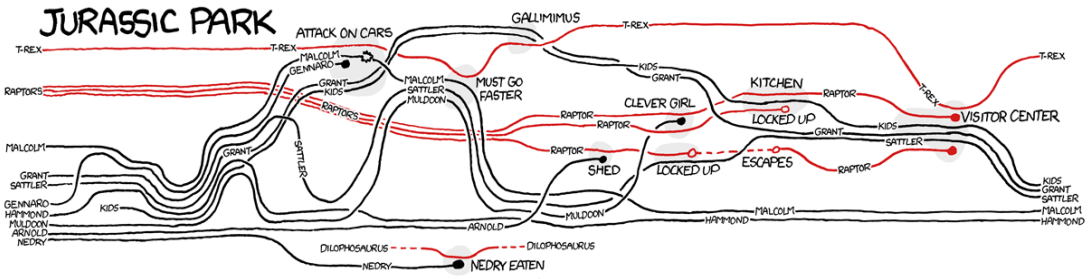


Abb. 1: Storyline für den Film „Jurassic Park“ [14]

1 Einleitung

Im Jahr 1869 veröffentlichte Charles Joseph Minard (1781−1870) eine grafische Darstellung von Napoleons Russlandfeldzug [12], welche Informationen wie Truppenstärke, Position und Temperatur in einer einzigen Grafik zusammenfasst. Solche Visualisierungen sind ein Mittel um Informationen und Daten interessant der Öffentlichkeit zugänglich zu machen. Sie können auch verwendet werden, um interessierte Laien an komplexe, wissenschaftliche Themen heranzuführen, oder auch eine künstlerische Seite dieser zu zeigen. Zwei Beispiele dafür sind die Darstellung von verschiedenen Fraktalen [1] oder der Kreiszahl π [10].

Diese Arbeit wird sich mit der Visualisierung sog. Storylines beschäftigen, welche sich, seit Munroe [14] einige solcher handgezeichneten Darstellungen erstellte, an immer mehr auch wissenschaftlicher Aufmerksamkeit erfreut. Als Storyline-Visualisierung bezeichnet man die chronologische Darstellung einer Geschichte. Alle Charaktere dieser werden durch x-monotone Linien auf einer Zeitachse repräsentiert. Interagieren mehrere Charaktere zu einem Zeitpunkt miteinander werden die betroffenen Linien näher zusammengeschoben und es darf sich keine andere, „fremde“ Linie zwischen Teilnehmern einer Interaktion befinden. Zusammengeschobene Linien würden dann z.B. ein Telefonat, einen Kampf oder ein einfaches Treffen der Charaktere repräsentieren. Abbildung 1 zeigt beispielsweise eine Storyline-Visualisierung.

Storylines wurden bereits von Kim et al. [7] verwendet, um Familienstammbäume, mit Fokus auf Eltern-Kind-Beziehungen, zu erstellen. Tanahashi und Ma [17] generierten Storyline-Visualisierungen automatisch und verbesserten ihre Darstellung anschließend durch Anpassen einzelner Linien. Muelder et al. [13] verwendeten Storylines, um lokale Änderungen in großen, dynamischen Graphen über einen Zeitraum darzustellen.

Das Ziel einer Storyline-Visualisierung ist es, wie bei den oben genannten Beispielen, eine möglichst übersichtliche Zeichnung der Storylines zu erstellen. Dabei ist es naheliegend, dass Kreuzungen von Linien möglichst vermieden werden sollten. Tatsächlich ist

die Minimierung der Kreuzungen von Storylines ein bereits mehrfach diskutiertes Thema. So zeigten Kostitsyna et al. [9] sowohl die NP-Vollständigkeit dieses Problems, als auch, dass das Problem Festparameter-berechenbar in der Anzahl der Charaktere κ ist.

Die Minimierung von Einzelkreuzungen ist allerdings nicht unbedingt die beste Lösung. So sind mehrere Kreuzungen gebündelt an einer Stelle meist übersichtlicher, als weniger Kreuzungen, die über die ganze Zeichnung verstreut sind. Wir betrachten in dieser Arbeit daher die Minimierung von Blockkreuzungen, um eventuell bessere Ergebnisse zu erzielen. Eine Blockkreuzung ist eine Kreuzung von zwei nichtleeren, benachbarten und zusammenhängenden Mengen von Linien. Abbildung 2 zeigt als Beispiel eine Blockkreuzung. Die Verwendung dieser speziellen Kreuzungen ist nicht neu. Bafna et al. [2] nutzen Blockkreuzungen um Mengen zu sortieren. Fink et al. [4] minimierten Blockkreuzungen für U-Bahn Karten. Hierbei ist die Ähnlichkeit zur Darstellung von Storylines bereits sehr gut zu erkennen, denn eine U-Bahn Karte enthält ebenfalls eine Menge von (meist) monotonen Linien. So ist es nicht verwunderlich, dass die Minimierung von Blockkreuzungen auch schon für Storylines betrachtet wurde. Van Dijk et al. [18] zeigten, dass ähnlich der Einzelkreuzungen auch unter Verwendung von Blockkreuzungen das Problem der Kreuzungsminimierung NP-vollständig und Festparameter-berechenbar in κ ist. Es ist also auch hier schwierig größere Instanzen effizient zu lösen. Gronemann et al. [5] entwickelten deshalb ein ganzzahlig lineares Programm (ILP), um das Problem der Minimierung von Einzelkreuzungen exakt zu lösen.

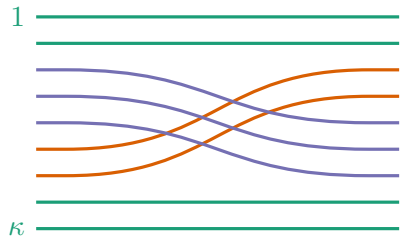


Abb. 2: Eine Blockkreuzung

Ein lineares Programm besteht aus einer lineare Zielfunktion und einer Menge von linearen Beschränkungen für eine Menge von Variablen. Das Ziel ist dabei, Werte für die Variablen so zu finden, dass die Zielfunktion minimiert oder maximiert wird. Beim ganzzahlig linearen Problem können die Variablen nur ganze Zahlen als Werte annehmen. Häufig sind sie sogar binär, d. h. sie können nur die Werte 0 oder 1 annehmen [8]. Die Komplexität eines Problems ändert sich dabei nicht, allerdings existiert ausgefeilte Software zur Lösung dieser Programme.

Gronemann et al. implementierten ihr ILP und präsentierten so anschauliche Lösungen von realen Instanzen aus Filmen und Büchern. Sie generierten z. B. eine optimale Lösung für eine Instanz mit 13 Charakteren und 97 Treffen in etwas weniger als einer Sekunde. In Anbetracht dessen möchten wir ihr Modell und ihr ILP erweitern, um das Problem für Blockkreuzungen exakt lösen zu können. Das daraus resultierende neue ILP soll ebenfalls implementiert und auf denselben Instanzen getestet werden, um einen anschaulichen Vergleich herstellen zu können. Wir werden sehen, dass die Komplexität von Blockkreuzungen eine Implementierung schwierig gestaltet, da weit mehr Variablen und vor allem Beschränkungen benötigt werden, was sich negativ auf die Laufzeit und den benötigten Speicher auswirkt.

Aus diesem Grund werden wir anschließend ein eigenes Modell entwickeln und darauf basierend einen zweiten ILP-Ansatz und eine Formulierung als SAT-Instanz vorstellen. Bei dem SAT-Entscheidungsproblem gilt es eine Menge von Variablen so zu entscheiden,

dass eine aussagenlogische Formel erfüllbar ist. Das Problem der Blockkreuzungsminimierung bleibt dabei NP-schwer, allerdings werden wir sehen, dass unsere verwendete Software Lösungen für reale Instanzen in hinnehmbarer Zeit findet.

Zuletzt werden wir experimentelle Ergebnisse vorstellen, welche wir mit den Ergebnissen von Gronemann et al. vergleichen werden.

2 Definition des Problems

Gegeben ist eine Storyline $S = (\mathcal{K}, \mathcal{M}, \{E_i \mid i \in \mathcal{K}\})$ und eine endliche Menge von Zeitpunkten $\mathcal{F}$ wobei $\mathcal{K} = \{1, \ldots, \kappa\}$ eine Menge von Charakteren und $\mathcal{M} = \{m_1, \ldots, m_\nu\}$ eine Menge von Treffen ist. Ein Treffen m_j ist ein Tripel $(s_{m_j}, e_{m_j}, C_{m_j})$, wobei $s_{m_j} \in \mathcal{F}$ den Startzeitpunkt und $e_{m_j} \in \mathcal{F}$ den Endzeitpunkt des Treffens angibt mit $s_{m_j} < e_{m_j}$. Das dritte Element $C_{m_j} \subseteq \mathcal{K}$ ist die Menge der Charaktere, welche an diesem Treffen beteiligt sind. Ein Charakter darf dabei nicht gleichzeitig an mehreren Treffen teilnehmen, d. h. für jedes paar von Treffen (m_j, m_l) mit $j \neq l$, $s_{m_j} < s_{m_l}$ und $C_{m_j} \cap C_{m_l} \neq \emptyset$ muss $e_{m_j} \leq s_{m_l}$ gelten. Charaktere werden als x-monotone Linien auf der Euklidischen Ebene dargestellt, wobei die x-Achse als Zeitachse dient.

Die Menge $E_i = \{[b_i^1, d_i^1 - 1], \ldots, [b_i^{\eta_i}, d_i^{\eta_i} - 1]\}$ mit $b_i^t, d_i^t \in \mathcal{F}$ enthält η_i Zeitintervalle, in denen ein Charakter i existiert, d. h. nur in diesen Zeitintervallen wird die Linie i gezeichnet. Diese Intervalle sind disjunkt. Wir bezeichnen diese Zeitintervalle als *Existenzintervalle* einer Linie i. Ein Charakter i darf nur an einem Treffen m_j teilnehmen, wenn er im Zeitintervall des Treffens existiert, d. h. wenn $s_{m_j} \geq b_i^t$ und $e_{m_j} \leq d_i^t$ mit $[b_i^t, d_i^t] \in E_i$ gilt.

Sei m_j ein Treffen und i, w, q Linien mit $i \notin C_{m_j}$ und $w, q \in C_{m_j}$. Dann darf i für die Dauer des Treffens $[s_{m_j}, e_{m_j} - 1]$ nicht zwischen w und q liegen, sofern i existiert.

Wir bezeichnen die Menge der Zeitpunkte $b_i^t, d_i^t, s_{m_j}, e_{m_j} \in \mathcal{F}$ für alle Charaktere $i \in \mathcal{K}$, alle $t \in \{1, \ldots, \eta_i\}$ und alle $j \in \{1, \ldots, \nu\}$ als *Events* $\mathcal{E}$.

Sei $\mathcal{R}$ eine weitere endliche Menge von Zeitpunkten mit $|\mathcal{R}| = \lambda$. Gesucht ist eine Folge von Permutationen $\mathcal{P}$ von Linien und eine Abbildung $A : \mathcal{E} \to \mathcal{P}$. Eine Permutation wird mit π^r für Zeitpunkte $r \in \mathcal{R}$ bezeichnet und stellt die Reihenfolge der Charaktere zu einem Zeitpunkt r dar. Die Abbildung A ordnet jedem Zeitpunkt $f \in \mathcal{F}$ eine Permutation π^r zu. Wir schreiben auch: Der Zeitpunkt f findet auf π^r statt. Dabei muss die chronologische Reihenfolge aller f bewahrt werden.

Definition 2.1 (Zeitbedingung)**.** Für zwei Zeitpunkte $f^r, f^t \in \mathcal{F}$, wobei f^r auf π^r und f^t auf π^t stattfindet, gilt $r \leq t$, falls $f^r < f^t$.

Auf welchen Permutationen ein Treffen aktiv ist lässt sich folgendermaßen festlegen.

Definition 2.2 (Treffbedingung)**.** Sei m ein Treffen und $f^r, f^t \in \mathcal{F}$ zwei Zeitpunkte, wobei f^r auf π^r und f^t auf π^t stattfindet und $f^r = s_m$, $f^t = e_m$ gilt. Dann ist das Treffen m auf einer Permutation π^u genau dann aktiv, d. h. alle Linien aus C_m müssen auf π^u zusammen liegen, falls $r \leq u < t$.

Auf welchen Permutationen ein Charakter i existiert kann ähnlich dazu bestimmt werden.

Definition 2.3 (Existenzbedingung)**.** Sei i ein Charakter und $f^r, f^t \in \mathcal{F}$ zwei Zeitpunkte, wobei f^r auf π^r und f^t auf π^t stattfindet und $f^r = b_i^p,\ f^t = d_i^p$ für ein $p \in \{1, \ldots, \eta_i\}$. Dann existiert i auf einer Permutation π^u, falls $r \leq u < t$. Sei nun $f^r \in \mathcal{F}$ ein Zeitpunkt, der auf π^r stattfindet und $f^r \notin [b_i^q, d_i^q - 1]$ für alle $q \in \{1, \ldots, \eta_i\}$. Dann existiert i nicht auf π^r.

Jede Permutation π^r muss so sortiert sein, dass alle Treffen, welche auf π^r aktiv sind, ein Intervall auf π^r bilden. Für das Sortieren dürfen nur Blockkreuzungen verwendet werden, wobei eine Blockkreuzung eine Kreuzung von zwei nichtleeren, benachbarten und zusammenhängenden Mengen von Linien ist.

3 Formulierung als ganzzahlig lineares Programm: Erster Ansatz

Um das ILP von Gronemann et al. für Blockkreuzungen anzupassen, konstruieren wir eine Instanz des Problems wie folgt:

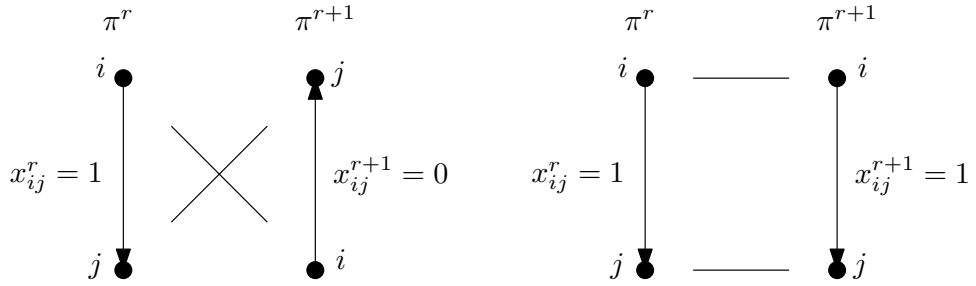
3.1 Konstruktion einer Instanz

Wir erstellen für jedes Event eine Permutation, auf der das Event stattfindet und sortieren sie so, dass die Zeitbedingung erfüllt ist. Diese Permutationen werden als *Eventpermutationen* bezeichnet. Um Blockkreuzungen als lineare Beschränkungen definieren zu können, fordern wir, dass zwischen zwei Permutationen höchstens eine Blockkreuzung stattfinden darf. Bafna et al. [2] haben gezeigt, dass sich jede beliebige Permutation der Länge κ mit $3\kappa/4$ Blockkreuzungen sortieren lässt. Wir machen uns dies zunutze, indem wir zwischen jedem Paar (π^r, π^{r+1}) von Permutationen der Folge der Eventpermutationen $3\kappa/4$ weitere Permutationen einfügen. So garantieren wir, dass zwischen zwei Events genug Blockkreuzungen stattfinden können, um jede beliebige Permutation zu erhalten, ohne zwischen zwei Permutationen mehr als eine Blockkreuzung machen zu müssen. Alle Zeitpunkte $f \in \mathcal{F}$ welche keine Events sind, werden anschließend beliebig auf Permutationen abgebildet, sodass die Zeitbedingung nicht verletzt ist. Durch diese Methode ist die Abbildung aller Zeitpunkte $f \in \mathcal{F}$ auf eine Permutation bereits gegeben. Es bleibt nur noch die Permutationen $\mathcal{P}$ so zu sortieren, dass die Treffbedingung erfüllt ist.

3.2 Beschränkungen und Variablen

Wir wissen, dass zwischen zwei Permutationen höchstens eine Blockkreuzung liegen muss um die Storyline darzustellen. Wir definieren daher $y^r \in \{0, 1\}$ als die Anzahl der Blockkreuzungen zwischen π^r und π^{r+1} für den Zeitpunkt $r \in \mathcal{R}$. Damit das ILP nur diese eine Blockkreuzung erzeugt, wird es wie folgt linear beschränkt. Wir definieren $x_{ij}^r \in \{0, 1\}$ mit $x_{ij}^r = 1$ genau dann, wenn die Linie i über der Linie j auf π^r liegt (wir schreiben auch $\pi^r(i) < \pi^r(j)$), andernfalls 0. Daraus folgt, dass sich die Linien i und j genau dann zwischen zwei Permutationen π^r und π^{r+1} überkreuzen, wenn $x_{ij}^r + x_{ij}^{r+1} = 1$. Ein Beispiel dafür ist in Abbildung 3 zu sehen.

Die Reihenfolge der Linien in einer Permutation muss eindeutig sein. Das heißt zum Einen, dass $x_{ij}^r \neq x_{ji}^r$ und zum Anderen, dass eine Permutation keinen Kreis enthalten darf, so wie es in Abbildung 4 gezeigt wird.



(a) $x_{ij}^r \neq x_{ij}^{r+1}$ **(b)** $x_{ij}^r = x_{ij}^{r+1}$

Abb. 3: i und j überkreuzen sich, genau dann wenn $x_{ij}^r \neq x_{ij}^{r+1}$

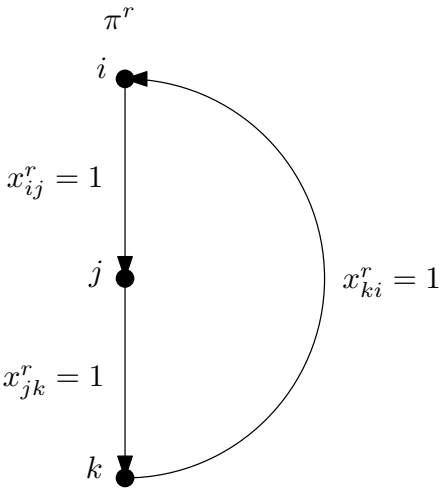


Abb. 4: Auf π^r besteht ein Kreis, da $x_{ij}^r = x_{jk}^r = x_{ki}^r = 1$

Diese beiden Bedingungen lassen sich darstellen durch

$$x_{ij}^r + x_{ji}^r = 1 \qquad \text{für alle } r \in \{1, \ldots, \lambda\},\ i, j \in \mathcal{K}, \text{ falls } i \neq j \text{ und } i, j \text{ existieren auf } \pi^r, \tag{3.1}$$

$$x_{ij}^r + x_{jk}^r + x_{ki}^r \leq 2 \qquad \text{für alle } r \in \{1, \ldots, \lambda\},\ i, j, k \in \mathcal{K}, \text{ falls } i \neq j, i \neq k, j \neq k \text{ und } i, j, k \text{ existieren auf } \pi^r. \tag{3.2}$$

Um dies für spätere Beschränkungen nutzen zu können, führen wir eine Kreuzungsvariable $\chi_{ij}^r \in \{0,1\}$ ein, mit $\chi_{ij}^r = 1$ genau dann, wenn sich die Linien i und j zwischen π^r und π^{r+1} überkreuzen. Wir betrachten hierfür die folgenden Beschränkungen.

$$\chi_{ij}^r \geq x_{ij}^r - x_{ij}^{r+1} \quad \text{für alle } r \in \{1,\dots,\lambda-1\},\ i,j \in \mathcal{K}, \text{ falls } i \neq j \text{ und } i,j \text{ existieren auf } \pi^r, \tag{3.3}$$

$$\chi_{ij}^r \geq x_{ij}^{r+1} - x_{ij}^r \quad \text{für alle } r \in \{1,\dots,\lambda-1\},\ i,j \in \mathcal{K}, \text{ falls } i \neq j \text{ und } i,j \text{ existieren auf } \pi^r, \tag{3.4}$$

$$\chi_{ij}^r \leq x_{ij}^r + x_{ij}^{r+1} \quad \text{für alle } r \in \{1,\dots,\lambda-1\},\ i,j \in \mathcal{K}, \text{ falls } i \neq j \text{ und } i,j \text{ existieren auf } \pi^r, \tag{3.5}$$

$$\chi_{ij}^r \leq x_{ji}^r + x_{ji}^{r+1} \quad \text{für alle } r \in \{1,\dots,\lambda-1\},\ i,j \in \mathcal{K}, \text{ falls } i \neq j \text{ und } i,j \text{ existieren auf } \pi^r. \tag{3.6}$$

Lemma 3.1. *Beschränkungen 3.3 und 3.4 erzwingen $\chi_{ij}^r = 1$, falls $x_{ij}^r + x_{ij}^{r+1} = 1$*

Beweis. Sei $x_{ij}^r + x_{ij}^{r+1} = 1$. Das bedeutet, dass entweder $x_{ij}^r = 1$ oder $x_{ij}^{r+1} = 1$. Aus diesen beiden Fällen folgt entweder $x_{ij}^r - x_{ij}^{r+1} = 1$ oder $x_{ij}^{r+1} - x_{ij}^r = 1$. Durch die Beschränkungen 3.3 und 3.4 gilt folglich $\chi_{ij}^r \geq 1$. Da $\chi_{ij}^r \in \{0,1\}$, gilt also $\chi_{ij}^r = 1$. □

Lemma 3.2. *Beschränkungen 3.5 und 3.6 erzwingen $\chi_{ij}^r = 0$, falls $x_{ij}^r + x_{ij}^{r+1} \neq 1$*

Beweis. Sei $x_{ij}^r + x_{ij}^{r+1} = 2$. Daraus folgt $x_{ij}^r = 1$ und $x_{ij}^{r+1} = 1$. Wir wissen aus Beschränkung 3.1, dass $x_{ij}^r + x_{ji}^r = 1$. Also gilt $x_{ji}^r + x_{ji}^{r+1} = 0$. Beschränkung 3.6 erzwingt dadurch $\chi_{ij}^r \leq 0$. Da $\chi_{ij}^r \in \{0,1\}$, gilt folglich $\chi_{ij}^r = 0$. Für $x_{ij}^r + x_{ij}^{r+1} = 0$ erzwingt 3.5 $\chi_{ij}^r \leq 0$ und somit $\chi_{ij}^r = 0$. □

Aus den beiden Lemmata 3.1 und 3.2 folgt das erwünschte Verhalten von χ_{ij}^r.

Korollar 3.3. *Beschränkungen 3.3, 3.4, 3.5 und 3.6 erzwingen $\chi_{ij}^r = 1$, genau dann, wenn $x_{ij}^r + x_{ij}^{r+1} = 1$, d.h. wenn sich i und j zwischen π^r und π^{r+1} kreuzen. Ferner gilt $\chi_{ij}^r = \chi_{ji}^r$.*

Eine Blockkreuzung ist eine Kreuzung von zwei benachbarten und zusammenhängenden Mengen $\mathcal{B}$ und $\mathcal{C}$ von Linien. Da zwischen zwei Permutationen höchstens eine Blockkreuzung liegt, definieren wir für jede Linie auf jeder Permutation drei weitere Variablen a, b und c. Diese geben an in welcher Menge sich eine Linie befindet.

$a_i^r = 1$ falls i auf π^r in keiner der Mengen $\mathcal{B}$ und $\mathcal{C}$ ist, andernfalls 0,
$b_i^r = 1$ falls i auf π^r in $\mathcal{B}$ ist, andernfalls 0,
$c_i^r = 1$ falls i auf π^r in $\mathcal{C}$ ist, andernfalls 0.

Jede Linie liegt entweder in $\mathcal{B}$, in $\mathcal{C}$ oder in keiner Menge, was sich folgendermaßen ausdrücken lässt.

$$a_i^r + b_i^r + c_i^r = 1 \qquad \text{für alle } r \in \{1, \ldots, \lambda\},\ i \in \mathcal{K}, \text{ falls } i \text{ auf } \pi^r \text{ existiert.} \tag{3.7}$$

Um zwischen $\mathcal{B}$ und $\mathcal{C}$ nicht durcheinander zu kommen, wird gefordert, dass alle Linien aus $\mathcal{B}$ über allen Linien aus $\mathcal{C}$ liegen. Daher gilt $c_j^r \wedge b_i^r \Rightarrow x_{ij}^r$ und somit

$$c_j^r + b_i^r - 1 \leq x_{ij}^r \qquad \text{für alle } r \in \{1, \ldots, \lambda\},\ i, j \in \mathcal{K}, \text{ falls } i \neq j \text{ und } i, j \text{ existieren auf } \pi^r. \tag{3.8}$$

Durch Beschränkung 3.8 muss also für jedes Paar von Linien i, j und jede Permutation π^r, $\pi^r(i) < \pi^r(j)$ gelten, falls $b_i^r = 1$ und $c_j^r = 1$.

Damit $\mathcal{B}$ garantiert zusammenhängend ist, muss $b_j^r = 1$ für alle j auf π^r gelten, falls j zwischen zwei Linien i und k liegt mit $b_i^r = 1$ und $b_k^r = 1$. Dies lässt sich durch die Implikation $x_{ij}^r \wedge x_{jk}^r \wedge b_i^r \wedge b_k^r \Rightarrow b_j^r$ darstellen. Dieses Verhalten wird durch die folgende Beschränkung erreicht.

$$x_{ij}^r + x_{jk}^r + b_i^r + b_k^r - 3 \leq b_j^r \qquad \text{für alle } r \in \{1, \ldots, \lambda\},\ i, j, k \in \mathcal{K}, \text{ falls } i \neq j, i \neq k, j \neq k \text{ und } i, j, k \text{ existieren auf } \pi^r. \tag{3.9}$$

Da dasselbe analog für alle c_j^r gilt, folgt:

$$x_{ij}^r + x_{jk}^r + c_i^r + c_k^r - 3 \leq c_j^r \qquad \text{für alle } r \in \{1, \ldots, \lambda\},\ i, j, k \in \mathcal{K}, \text{ falls } i \neq j, i \neq k, j \neq k \text{ und } i, j \text{ existieren auf } \pi^r. \tag{3.10}$$

Wir müssen ebenfalls garantieren, dass $\mathcal{B}$ und $\mathcal{C}$ benachbart sind. Dies kann ähnlich zu den Beschränkungen 3.9 und 3.10 erreicht werden, denn auch hier fordern wir eine bestimmte Mengenzugehörigkeit für eine Linie, wenn sie zwischen zwei anderen Linien mit ebenfalls spezieller Mengenzugehörigkeit liegt. So fordern wir $a_j^r = 0$ für alle Linien j auf π^r, falls sie zwischen zwei Linien i und k mit $b_i^r = 1$ und $c_k^r = 1$ liegt. Die Implikation $x_{ij}^r \wedge x_{jk}^r \wedge b_i^r \wedge c_k^r \Rightarrow \neg a_j^r$ erfüllt dies und die lineare Beschränkung ist entsprechend

$$-(x_{ij}^r + x_{jk}^r + b_i^r + c_k^r) + 4 \geq a_j^r \qquad \text{für alle } r \in \{1, \ldots, \lambda\},\ i, j, k \in \mathcal{K}, \text{ falls } i \neq j, i \neq k, j \neq k \text{ und } i, j, k \text{ existieren auf } \pi^r. \tag{3.11}$$

Hierbei wird ersichtlich, wozu Beschränkung 3.8 benötigt wird. Wären eine Linie i aus $\mathcal{C}$ auf π^r über einer Linie k aus $\mathcal{B}$ und eine weitere Linie j zwischen i und k, dann würde gelten:

$$\begin{aligned} -(1+1+0+0)+4 &\geq a_j^r \\ 2 &\geq a_j^r \end{aligned}$$

Die Variable a_j^r könnte also den Wert 1 annehmen, obwohl j zwischen den Intervallen $\mathcal{B}$ und $\mathcal{C}$ liegt.
Zuletzt muss sichergestellt werden, dass sich die Linien von $\mathcal{B}$ und $\mathcal{C}$, und auch nur genau diese, überkreuzen mit $\chi_{ij}^r = 1$ genau dann, wenn $b_i^r + c_j^r = 2$ für alle $i, j, r \in \mathcal{K}$. Dies kann erreicht werden durch die linearen Beschränkungen

$$\chi_{ij}^r \geq b_i^r + c_j^r - 1 \quad \text{für alle } r \in \{1, \ldots, \lambda - 1\}, \; i, j \in \mathcal{K}, \text{ falls } i \neq j \text{ und } i, j \text{ existieren auf } \pi^r, \tag{3.12}$$

$$\chi_{ij}^r \leq 1 - a_i^r \quad \text{für alle } r \in \{1, \ldots, \lambda - 1\}, \; i, j \in \mathcal{K}, \text{ falls } i \neq j \text{ und } i, j \text{ existieren auf } \pi^r, \tag{3.13}$$

$$\chi_{ij}^r \leq -(b_i^r + b_j^r) + 2 \quad \text{für alle } r \in \{1, \ldots, \lambda - 1\}, \; i, j \in \mathcal{K}, \text{ falls } i \neq j \text{ und } i, j \text{ existieren auf } \pi^r, \tag{3.14}$$

$$\chi_{ij}^r \leq -(c_i^r + c_j^r) + 2 \quad \text{für alle } r \in \{1, \ldots, \lambda - 1\}, \; i, j \in \mathcal{K}, \text{ falls } i \neq j \text{ und } i, j \text{ existieren auf } \pi^r. \tag{3.15}$$

Lemma 3.4. *Beschränkungen 3.12, 3.13, 3.14 und 3.15 garantieren, dass $b_i^r + c_j^r = 2$, genau dann, wenn $\chi_{ij}^r = 1$, d. h. die Mengen $\mathcal{B}$ und $\mathcal{C}$ überkreuzen sich.*

Beweis. Es ist offensichtlich, dass Beschränkung 3.12 $\chi_{ij}^r = 1 \Leftarrow b_i^r + c_j^r = 2$ garantiert. Es bleibt also nur noch die andere Richtung zu zeigen. Hierfür eignet sich eine Fallunterscheidung. Falls $a_i^r = 1$ oder $a_j^r = 1$ gilt durch Beschränkung 3.13 $\chi_{ij}^r \leq 1 - 1 = 0$. Falls $b_i^r = 1$ und $b_j^r = 1$ gilt durch Beschränkung 3.14 $\chi_{ij}^r \leq -2 + 2 = 0$. Für $c_i^r = 1$ und $c_j^r = 1$ gilt durch Beschränkung 3.15 dasselbe analog. □

Die Kreuzungsvariable χ_{ij}^r ist nun eine zuverlässige Variable, um eine Kreuzung zu erkennen und durch Lemma 3.4 wissen wir, dass nur zulässige Blockkreuzungen erstellt

werden. Durch diese Erkenntnisse ist es leicht die Blockkreuzungsvariable y^r so zu beschränken, dass sie später zum Zählen der Blockkreuzungen in der Zielfunktion verwendet werden kann.

$$y^r \geq \chi_{ij}^r \qquad \begin{array}{l} \text{für alle } r \in \{1, \ldots, \lambda - 1\},\ i, j \in \mathcal{K}, \\ \text{falls } i \neq j \\ \text{und } i, j \text{ existieren auf } \pi^r. \end{array} \tag{3.16}$$

Wir wissen durch die Treffbedingung auf welchen Permutationen π^r ein Treffen stattfindet, allerdings noch nicht, wie das ILP damit umgehen soll. Wir müssen sicherstellen, dass zwischen zwei beliebigen Linien eines Treffens m keine Linie, welche nicht in C_m ist, liegt. Dies kann erreicht werden durch

$$x_{ij}^r = x_{kj}^r \qquad \begin{array}{l} \text{für alle } r \in \{1, \ldots, \lambda\},\ i, j, k \in \mathcal{K},\ m \in \mathcal{M}, \\ \text{falls } i, k \in C_m,\ j \notin C_m \\ \text{und } m \text{ ist auf } \pi^r \text{ aktiv.} \end{array} \tag{3.17}$$

3.3 Zielfunktion

Da wir die Anzahl der Blockkreuzungen minimieren möchten, ergibt sich als Zielfunktion:

$$\text{minimiere} \sum_{r=1}^{\lambda-1} y^r \tag{3.18}$$

Zusammen mit den linearen Beschränkungen 3.1 bis 3.17 erhält man das geschwünschte ILP.

Satz 3.5. *Das ILP minimiert die Anzahl der Blockkreuzungen für eine Storyline.*

Beweis. Die Korrektheit folgt aus dem korrekten Verhalten der Variablen und Beschränkungen. □

3.4 Analyse

Um einschätzen zu können, wie gut das ILP zum Lösen realer Instanzen geeignet ist, möchten wir dessen Leistung kurz bewerten. Ein einfaches Mittel dafür ist die Anzahl der Variablen, welche das ILP entscheiden muss, in Abhängigkeit der Anzahl der Charaktere κ und der Anzahl der Treffen ν zu betrachten, wobei ν durch die Anzahl der Events $|\mathcal{E}|$ dargestellt wird. Wir beginnen mit der Anzahl der Permutationen λ. Jedes Event erhält seine eigene und weitere $3\kappa/4$ Permutationen, welche zwischen den Eventpermuationen eingefügt werden. Daraus folgt

$$\lambda = \mathcal{O}(|\mathcal{E}| \cdot \left(\frac{3\kappa}{4} + 1\right) + 1) = \mathcal{O}(|\mathcal{E}| \cdot \kappa + |\mathcal{E}|).$$

Die Variablen a_i^r,b_i^r und c_i^r haben dieselbe Anzahl. Da sie die Intervallszugehörigkeit jedes Charakters i auf jeder Permutationen r festlegen, gilt für ihre Anzahl

$$|a| + |b| + |c| = \mathcal{O}(3 \cdot \lambda \cdot \kappa) = \mathcal{O}(3 \cdot (|\mathcal{E}| \cdot \kappa + |\mathcal{E}|) \cdot \kappa) = \mathcal{O}(|\mathcal{E}| \cdot \kappa^2 + |\mathcal{E}|).$$

Die Variable x_{ij}^r weist jedem Paar (i, j) von Charakteren auf jeder Permutation π^r ihre Ordnung zu. Daher gilt

$$|x| = \mathcal{O}(\lambda \cdot \kappa^2) = \mathcal{O}(|\mathcal{E}| \cdot \kappa^3 + |\mathcal{E}| \cdot \kappa^2).$$

Ähnlich verhält es sich für χ_{ij}^r, welche für alle Charakterpaare (i, j) und der Permutation π^r mit $r \in \{1, \ldots, \lambda - 1\}$ ihr Kreuzungsverhalten zwischen π^r und π^{r+1} angibt. Die Anzahl der χ Variablen ist damit analog zu x

$$|\chi| = \mathcal{O}((\lambda - 1) \cdot \kappa^2) = \mathcal{O}(\lambda \cdot \kappa^2) = \mathcal{O}(|\mathcal{E}| \cdot \kappa^3 + |\mathcal{E}| \cdot \kappa^2).$$

Da y^r für alle $r \in \{1, \ldots, \lambda - 1\}$ die Anzahl der Blockkreuzungen zwischen π^r und π^{r+1} angibt, gilt für deren Anzahl

$$|y| = \mathcal{O}(\lambda - 1) = \mathcal{O}(\lambda) = \mathcal{O}(|\mathcal{E}| \cdot \kappa + |\mathcal{E}|).$$

Daraus folgt für die Gesamtzahl der Variablen

$$V = \mathcal{O}(|\mathcal{E}| \cdot \kappa^3 + |\mathcal{E}| \cdot \kappa^2 + |\mathcal{E}| \cdot \kappa + |\mathcal{E}|) = \mathcal{O}(|\mathcal{E}| \cdot (\kappa^3 + \kappa^2 + \kappa + 1)).$$

Daraus wird ersichtlich, dass die Anzahl der Variablen vor allem mit der Anzahl der Charaktere stark steigt, wohingegen sie mit der Anzahl der Events nur linear steigt.

4 Formulierung als ganzzahlig lineares Programm: Zweiter Ansatz

Die experimentellen Tests in Kapitel 6 des ILP werden zeigen, dass dieses für reale Instanzen nicht brauchbar ist. Vor allem eine große Anzahl an Charakteren wirkt sich negativ auf die Laufzeit aus, was in Abschnitt 3.4 ersichtlich wurde. Dies liegt vor allem an der großen Zahl der Permutationen. Im ersten ILP Ansatz benötigen wir diese viele Permutationen, um zwischen zwei Events jede beliebige Permutation der Charaktere erreichen zu können. Dies war nötig um die Optimalität der Lösung zu garantieren. Im Realfall allerdings lassen sich selbst Instanzen mit großen κ und ν häufig mit weitaus weniger Blockkreuzungen, und damit Permutationen, lösen.

4.1 Konstruktion einer Instanz

Wir machen uns dies zunutze, indem wir eine maximale Anzahl an Permutationen λ fest vorgeben und dafür den Zeitpunkt $r \in \mathcal{R}$, zu dem ein Event stattfindet, durch Variablen entscheiden lassen.

Wir übernehmen die Variablen a, b, c, x und χ aus dem ersten Ansatz. Das neue ILP minimiert die Anzahl der Blockkreuzungen nicht mehr, sondern entscheidet lediglich, ob eine Lösung mit λ Permutationen existiert. Deswegen ist die Variable y überflüssig.

Da Zeitpunkte $r \in \mathcal{R}$ für Events nun vom ILP bestimmt werden, müssen Treffen bei der Konstruktion einer Instanz anders behandelt werden. Wir führen daher *Treffgruppen* ein. Das sind Mengen von Treffen, die angeben, welche Treffen gleichzeitig aktiv sind. Die Instanz benötigt dann nur eine Folge von μ Treffgruppen $\mathcal{T} = \{T_1, \ldots, T_\mu\}$, um das Problem zu lösen. Seien beispielsweise m_1 und m_2 Treffen mit Startzeitpunkten $s_{m_1} = 0$ und $s_{m_2} = 1$ und Endzeitpunkten $e_{m_1} = 2$ und $e_{m_2} = 3$. Diese Treffen würden dann durch eine Folge von drei Treffgruppen $\mathcal{T} = (T_1, T_2, T_3)$ dargestellt werden mit $T_1 = \{C_{m_1}\}$, $T_2 = \{C_{m_1}, C_{m_2}\}$ und $T_3 = \{C_{m_2}\}$. Dabei ist ersichtlich, dass eine neue Treffgruppe immer dann entsteht, wenn ein Treffen beginnt oder endet, d. h. es lassen sich für jede Treffgruppe zwei Zeitpunkte $f_s, f_e \in \mathcal{F}$ festlegen, die den Beginn bzw. das Ende der Treffgruppe darstellen. Leere Treffgruppen werden aus der Folge der Treffgruppen $\mathcal{T}$ entfernt.

Neue Variablen werden für jede Treffgruppe T_l eine Permutation π^u entscheiden, auf der T_l aktiv ist und liefern damit einen Teil der gesuchte Abbildung $A : \mathcal{E} \to \mathcal{P}$, welche jeden Zeitpunkt $f \in \mathcal{F}$ auf eine Permutation π^r abbildet. Dies geschieht indem alle Zeitpunkte $f_t \in \mathcal{F}$ mit $f_s \leq f_t < f_e$ auf π^u abgebildet werden.

Um die Events, welche Charaktere hinzufügen oder entfernen, auf eine Permutation abbilden zu können, erzwingen wir, dass jeder Charakter nur genau dann existiert, wenn

er an einem Treffen beteiligt ist. Ein Charakter i, der im Zeitraum t_1 bis t_2 mit $t_1, t_2 \in \mathcal{F}$ existieren soll, obwohl er an keinem Treffen beteiligt ist, kann durch Hinzufügen eines Treffens m mit $s_m = t_1$, $e_m = t_2$ und $C_m = \{i\}$ dargestellt werden.

4.2 Beschränkungen und Variablen

Die neue Variable q_l^r entscheidet für jede Treffgruppe T_l und jeder Permutation π^r, ob T_l auf π^r aktiv ist. Da jede Treffgruppe genau einmal aktiv sein muss, beschränken wir mit

$$\sum_{j=1}^{\lambda} q_l^j = 1 \qquad \text{für alle } l \in \{1, \dots, \mu\}. \tag{4.1}$$

Wir müssen ebenfalls die Ordnung der Treffgruppen wahren. Dafür nutzen wir die partielle Summe

$$\sum_{j=1}^{r} q_{l-1}^j \geq q_l^r \qquad \text{für alle } r \in \{1, \dots, \lambda\},\ l \in \{2, \dots, \mu\} \tag{4.2}$$

und

$$q_1^1 = 1. \tag{4.3}$$

Das erste ILP hat das Hinzufügen bzw. Entfernen von Charakteren im Verlauf einer Storyline einfach gemacht, da die Abbildung der Events auf Permutationen bereits beim Erstellen der Instanz festgelegt war, d.h. wir wussten bereits zu Beginn für jede Permutation π^r, welche Treffen auf π^r aktiv sind und welche Charaktere auf π^r existieren. Da die Variable q_l^r hier allerdings entscheidet, welche Treffgruppen auf welcher Permuation π^r aktiv ist, kann man nicht zu Beginn bestimmen, wann welche Charaktere existieren. Wir benötigen dafür eine weitere Variable d_i^r, welche angibt, ob ein Charakter i auf π^r existiert. Falls $d_i^r = 1$ existiert i auf π^r nicht und falls $d_i^r = 0$ existiert i auf π^r. Da Charaktere genau solange sie in einem Treffen aktiv sind existieren, gilt folglich $q_l^r \Rightarrow \neg d_i^r$ genau dann, wenn i in einem Treffen der Treffgruppe T_l ist. Wir definieren $\mathcal{G}_i$ als die Menge aller l, für welche gilt: Der Charakter i kommt in einem Treffen aus T_l vor. Für die Beschränkungen gilt dann:

$$q_l^r \leq 1 - d_i^r \qquad \text{für alle } r \in \{1, \dots, \lambda\},\ i \in \mathcal{K},\ l \in \{1, \dots, \mu\},\ m \in T_l, \text{ falls } i \in C_m, \tag{4.4}$$

$$\sum_{l \in \mathcal{G}_i} q_l^r \geq 1 - d_i^r \qquad \text{für alle } r \in \{1, \dots, \lambda\},\ i \in \mathcal{K}. \tag{4.5}$$

Falls auf einer Permutation π^r keine Treffgruppe aktiv ist, müssen die Charaktere der voranstehenden Permutation π^{r-1} existieren, da sonst durch Entfernen und anschließendem wieder Hinzufügen aller Charaktere jede beliebige Permutation ohne Blockkreuzungen erreicht werden kann. Es gilt daher $\sum_{l=1}^{\mu} q_l^r = 0 \Rightarrow d_i^r = d_i^{r-1}$, bzw. als Beschränkungen

$$\sum_{l=1}^{\mu} q_l^r \geq d_i^r - d_i^{r-1} \qquad \text{für alle } r \in \{2, \ldots, \lambda\},\ i \in \mathcal{K}, \tag{4.6}$$

$$\sum_{l=1}^{\mu} q_l^r \geq -d_i^r + d_i^{r-1} \qquad \text{für alle } r \in \{2, \ldots, \lambda\},\ i \in \mathcal{K}. \tag{4.7}$$

Wir verwenden Blockkreuzungen ähnlich wie im ersten Ansatz, weswegen wir viele Beschränkungen mit nur kleinen Modifikationen übernehmen können. So muss lediglich garantiert werden, dass Beschränkungen für einen Charakter i nur gelten, wenn dieser auch existiert. Dies erreicht man durch Addition der Variable d_i^r um die Beschränkungen entsprechend zu entschärfen. So werden die Beschränkungen 3.1 und 3.2 zu

$$x_{ij}^r + x_{ji}^r \leq 1 + d_i^r + d_j^r \qquad \text{für alle } i, j \in \mathcal{K},\ r \in \{1, \ldots, \lambda\}, \text{ falls } i \neq j, \tag{4.8}$$

$$x_{ij}^r + x_{ji}^r + d_i^r + d_j^r \geq 1 \qquad \text{für alle } i, j \in \mathcal{K},\ r \in \{1, \ldots, \lambda\}, \text{ falls } i \neq j, \tag{4.9}$$

$$x_{ij}^r + x_{jk}^r + x_{ki}^r \leq 2 + d_i^r + d_j^r \qquad \text{für alle } i, j, k \in \mathcal{K},\ r \in \{1, \ldots, \lambda\}, \text{ falls } i \neq j, i \neq k, k \neq j. \tag{4.10}$$

Wir verfahren mit den Beschränkungen 3.3 bis 3.15 analog und erhalten so die folgenden Beschränkungen:

$$\chi_{ij}^r + d_i^r + d_j^r \geq x_{ij}^r - x_{ij}^{r+1} \qquad \text{für alle } r \in \{1, \ldots, \lambda - 1\},\ i, j \in \mathcal{K},\ \text{falls } i \neq j, \tag{4.11}$$

$$\chi_{ij}^r + d_i^r + d_j^r \geq x_{ij}^{r+1} - x_{ij}^r \qquad \text{für alle } r \in \{1, \ldots, \lambda - 1\},\ i, j \in \mathcal{K},\ \text{falls } i \neq j, \tag{4.12}$$

$$\chi_{ij}^r \leq x_{ij}^r + x_{ij}^{r+1} + d_i^r + d_j^r \qquad \text{für alle } r \in \{1, \ldots, \lambda - 1\},\ i, j \in \mathcal{K},\ \text{falls } i \neq j, \tag{4.13}$$

$$\chi_{ij}^r \leq x_{ji}^r + x_{ji}^{r+1} + d_i^r + d_j^r \qquad \text{für alle } r \in \{1, \ldots, \lambda - 1\},\ i, j \in \mathcal{K},\ \text{falls } i \neq j, \tag{4.14}$$

$$a_i^r + b_i^r + c_i^r \leq 1 + 2 \cdot d_i^r \qquad \text{für alle } r \in \{1, \ldots, \lambda\},\ i \in \mathcal{K}, \tag{4.15}$$

$$a_i^r + b_i^r + c_i^r + d_i^r \geq 1 \qquad \text{für alle } r \in \{1, \ldots, \lambda\},\ i \in \mathcal{K}, \tag{4.16}$$

$$c_j^r + b_i^r - 1 \leq x_{ij}^r + d_i^r + d_j^r \qquad \text{für alle } r \in \{1, \ldots, \lambda\},\ i, j \in \mathcal{K},\ \text{falls } i \neq j, \tag{4.17}$$

$$x_{ij}^r + x_{jk}^r + b_i^r + b_k^r - 3 \leq b_j^r + d_i^r + d_j^r + d_k^r \qquad \text{für alle } r \in \{1, \ldots, \lambda\}, \quad i, j, k \in \mathcal{K}, \text{ falls } i \neq j, i \neq k, j \neq k, \tag{4.18}$$

$$x_{ij}^r + x_{jk}^r + c_i^r + c_k^r - 3 \leq c_j^r + d_i^r + d_j^r + d_k^r \qquad \text{für alle } r \in \{1, \ldots, \lambda\}, \quad i, j, k \in \mathcal{K}, \text{ falls } i \neq j, i \neq k, j \neq k, \tag{4.19}$$

$$-(x_{ij}^r + x_{jk}^r + b_i^r + c_k^r) + 4 + d_i^r + d_j^r + d_k^r \geq a_j^r \qquad \text{für alle } r \in \{1, \ldots, \lambda\}, \quad i, j, k \in \mathcal{K}, \text{ falls } i \neq j, i \neq k, j \neq k, \tag{4.20}$$

$$\chi_{ij}^r + d_i^r + d_j^r \geq b_i^r + c_j^r - 1 \qquad \text{für alle } r \in \{1, \ldots, \lambda - 1\}, \quad i, j \in \mathcal{K}, \text{ falls } i \neq j, \tag{4.21}$$

$$\chi_{ij}^r \leq 1 - a_i^r + d_i^r + d_j^r \qquad \text{für alle } r \in \{1, \ldots, \lambda - 1\}, \quad i \in \mathcal{K}, \tag{4.22}$$

$$\chi_{ij}^r \leq -(b_i^r + b_j^r) + 2 + d_i^r + d_j^r \qquad \text{für alle } r \in \{1, \ldots, \lambda - 1\}, \quad i, j \in \mathcal{K}, , \text{ falls } i \neq j \tag{4.23}$$

$$\chi_{ij}^r \leq -(c_i^r + c_j^r) + 2 + d_i^r + d_j^r \qquad \text{für alle } r \in \{1, \ldots, \lambda - 1\}, \quad i, j \in \mathcal{K}, \text{ falls } i \neq j. \tag{4.24}$$

Falls mehrere Treffgruppen, mit einer unterschiedlichen Menge von existierenden Charakteren, auf π^r aktiv sind, kommt es zu einem Konflikt, da nicht mehr eindeutig ist, welche Charaktere auf π^r existieren. In einem solchen Fall kann es sinnvoll sein, weitere Permutationen einzufügen. Abbildung 5a zeigt dafür ein Beispiel. In der Abbildung wird allerdings auch ersichtlich, dass dieselbe Beispielinstanz mit zwei Permutationen und einer Blockkreuzung ebenfalls lösbar ist. Da wir die Anzahl der Permutationen vorgeben, kann es hier zu unterschiedlichen Ergebnissen mit einer unterschiedlichen Anzahl von Blockkreuzungen kommen, wie es in Abbildung 5b zu sehen ist.

Um dem entgegen zu wirken erzwingen wir, dass zwei Treffgruppen T_l und T_o nicht auf derselben Permutation π^r aktiv sein dürfen, falls ihre Treffen unterschiedliche Charaktere enthalten. Es gilt dann $q_l^r \Rightarrow \neg q_o^r$ und $q_o^r \Rightarrow \neg q_l^r$, was sich darstellen lässt als

$$q_l^r + q_o^r \leq 1 \qquad \text{für alle } r \in \{1, \ldots, \lambda\}, \quad l, o \in \{1, \ldots, \mu\} \text{ ,falls } l \neq o \text{ und } T_l, T_o \text{ nicht dieselben Charaktere enthalten.} \tag{4.25}$$

Wir müssen verhindern, dass diese *speziellen Permutationen* für Blockkreuzungen missbraucht werden, wie es in Abb. 6 gezeigt wird. Dazu wird eine Beschränkung benötigt,

die keine Kreuzung zwischen zwei Permutationen π^r und π^{r+1} zulässt, falls es einen Charakter i gibt mit $d_i^r \neq d_i^{r+1}$. Die Ausnahme dabei sind Charaktere die auf mindestens einer dieser Permutationen nicht existieren.

$$\chi_{jk}^r \leq d_i^r - d_i^{r+1} + d_j^r + d_j^{r+1} + d_k^r + d_k^{r+1} + 1 \qquad \text{für alle } i,j,k \in \mathcal{K}, \quad r \in \{1,\ldots,\lambda-1\}, \quad \text{falls } i \neq j, i \neq k, k \neq j, \tag{4.26}$$

$$\chi_{jk}^r \leq -d_i^r + d_i^{r+1} + d_j^r + d_j^{r+1} + d_k^r + d_k^{r+1} + 1 \qquad \text{für alle } i,j,k \in \mathcal{K}, \quad r \in \{1,\ldots,\lambda-1\}, \quad \text{falls } i \neq j, i \neq k, k \neq j. \tag{4.27}$$

Zuletzt werden noch Beschränkungen benötigt, welche erzwingen, dass Linien jedes Treffens einer Treffgruppe T_l auf r zusammen liegen, wenn $q_l^r = 1$.

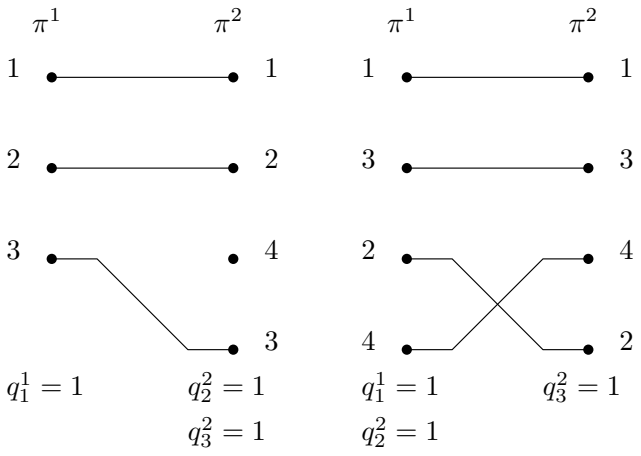
$$q_l^r \leq x_{ij}^r - x_{kj}^r + 1 \qquad \text{für alle } i,j,k \in \mathcal{K},\ r \in \{1,\ldots,\lambda\},\ l \in \{1,\ldots,\mu\},\ m \in T_l, \quad \text{falls } i,k \in C_m \text{ und } j \notin C_m, \tag{4.28}$$

$$q_l^r \leq x_{kj}^r - x_{ij}^r + 1 \qquad \text{für alle } i,j,k \in \mathcal{K},\ r \in \{1,\ldots,\lambda\},\ l \in \{1,\ldots,\mu\},\ m \in T_l, \quad \text{falls } i,k \in C_m \text{ und } j \notin C_m. \tag{4.29}$$

Durch Kombination der linearen Beschränkungen 4.1 bis 4.29 erhält man das gesuchte ILP.

Satz 4.1. *Falls der zweite ILP Ansatz eine Lösung mit λ Permutationen findet und keine Lösung mit $\lambda - 1$ Permutationen, so ist die Anzahl der Blockkreuzungen in der Lösung minimal.*

Beweis. Durch Beschränkung 4.25 wird für jedes Paar (T_l, T_{l+1}) eine spezielle Permutation benötigt, falls die Treffen von T_l und T_{l+1} nicht dieselben Charaktere enthalten. Sei λ_{min} die Anzahl solcher Permutationen. Da die Folge der Treffgruppen $\mathcal{T}$ gegeben ist, ist λ_{min} für eine Instanz konstant. Durch Beschränkungen 4.26 und 4.27 dürfen zwischen zwei Permutationen keine Blockkreuzung stattfinden, falls eine dieser Permutationen eine spezielle Permutation ist. Es verbleiben folglich $\lambda - \lambda_{min}$ Permutationen bzw. Blockkreuzungen, um eine Lösung zu finden, denn zwischen zwei „normalen“ Permutationen ist höchstens eine Blockkreuzung erlaubt. Findet das ILP keine Lösung mit $\lambda - 1$ Permutationen, so existiert folglich keine Lösung mit $\lambda - \lambda_{min} - 1$ Blockkreuzungen. Findet das ILP eine Lösung mit λ Permutationen, so existiert folglich eine Lösung mit $\lambda - \lambda_{min}$ Blockkreuzungen. Damit ist $\lambda - \lambda_{min}$ die minimale Anzahl von Blockkreuzungen. □



(a) Ohne Blockkreuzung

(b) Mit einer Blockkreuzung

Abb. 5: Sei $T_1 = \{(1,2,3)\}$, $T_2 = \{(2,4),(1),(3)\}$ und $T_3 = \{(3,4),(1),(2)\}$. Alle drei Treffgruppen passen nicht auf eine Permutation. Bei zwei Permutationen gibt es allerdings mehrere gültige Lösungen mit verschiedener Anzahl von Blockkreuzungen

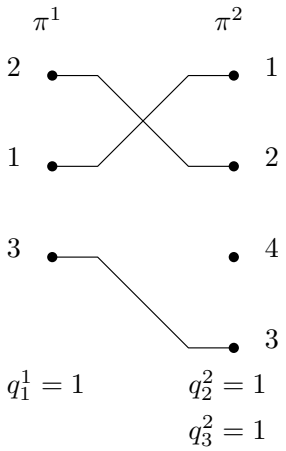


Abb. 6: Sei $T_1 = \{(1,2,3)\}$, $T_2 = \{(2,4),(1),(3)\}$ und $T_3 = \{(3,4),(1),(2)\}$. Die spezielle Permutation π^2 wird für eine Blockkreuzung missbraucht

4.3 Analyse

Um den ersten mit dem zweiten Ansatz vergleichen zu können, möchten wir auch die Anzahl der Variablen des zweiten Ansatzes zählen. Die Variablen a, b, c, x und χ verhalten sich dabei identisch. Da wir die Anzahl der Permutationen λ fest vorgeben, ändert sich die Anzahl der obigen Variablen wie folgt:

$$\begin{aligned} |a| = |b| = |c| &= \mathcal{O}(\lambda \cdot \kappa). \\ |\chi| &= \mathcal{O}((\lambda - 1) \cdot \kappa^2) = \mathcal{O}(\lambda \cdot \kappa^2). \\ |x| &= \mathcal{O}(\lambda \cdot \kappa^2). \end{aligned}$$

Die Variable q_i^r entscheidet für jede Treffgruppe $T_i \in \mathcal{T}$ und jede Permutation π^r, ob T_i auf π^r aktiv ist. Ihre Anzahl ist also gegeben durch

$$|q| = \mathcal{O}(\lambda \cdot |\mathcal{T}|).$$

Die Variable d_i^r entscheidet für jeden Charakter i und jede Permutation π^r, ob i auf π^r existiert.

$$|d| = \mathcal{O}(\lambda \cdot \kappa).$$

Die Gesamtanzahl der Variablen ist dann

$$\mathcal{V} = \mathcal{O}(2 \cdot \lambda \cdot \kappa^2 + 2 \cdot \lambda \cdot \kappa + \lambda \cdot |\mathcal{T}|) = \mathcal{O}(\lambda \cdot (\kappa + \kappa^2 + |\mathcal{T}|)).$$

Im Vergleich mit dem ersten ILP Ansatz steigt die Anzahl der Variablen weniger stark mit einer steigenden Anzahl von Charakteren. Da wir die Anzahl der Permutationen λ vorgeben und annehmen, dass sich selbst komplizierte Instanzen mit wenig Permutationen lösen können, erhalten wir generell weniger Variablen als beim ersten ILP-Ansatz.

5 Formulierung als SAT-Instanz

Das ILP ist für die Optimierung realer Instanzen ungeeignet, da Laufzeit und Speicherverbrauch zu hoch sind. Da dieses allerdings nur binäre Variablen enthält und der zweite Ansatz keine Funktion optimiert, ist eine Formulierung als SAT Entscheidungsproblem naheliegend. Der Wert der Variablen ist dabei intuitiv. So wird 0 zu False und 1 zu True. Im Folgenden werden wir die Beschränkungen des zweiten ILP Ansatzes zu Klauseln übersetzten.

Die Eindeutigkeit der Ordnung (Beschränkungen 4.8 und 4.9) zweier Linien i und j, also $x_{ij}^r \Leftrightarrow \neg x_{ji}^r$, lässt sich durch zwei Klauseln darstellen.

$$x_{ij}^r \vee x_{ji}^r \vee d_i^r \vee d_j^r \qquad \text{für alle } i,j \in \mathcal{K}, \; r \in \{1,\dots,\lambda\},$$

$$\neg x_{ij}^r \vee \neg x_{ji}^r \vee d_i^r \vee d_j^r \qquad \text{für alle } i,j \in \mathcal{K}, \; r \in \{1,\dots,\lambda\}.$$

Die Ordnung von drei Linien i,j und k darf keinen Kreis enthalten (Beschränkung 4.10). Das bedeuted x_{ij}^r, x_{jk}^r und x_{ki}^r dürfen nicht alle auf True bzw. alle auf False gesetzt sein.

$$x_{ij}^r \vee x_{jk}^r \vee x_{ki}^r \vee d_i^r \vee d_j^r \vee d_k^r \qquad \text{für alle } i,j,k \in \mathcal{K}, \; r \in \{1,\dots,\lambda\},$$

$$\neg x_{ij}^r \vee \neg x_{jk}^r \vee \neg x_{ki}^r \vee d_i^r \vee d_j^r \vee d_k^r \qquad \text{für alle } i,j,k \in \mathcal{K}, \; r \in \{1,\dots,\lambda\}.$$

Die Varibale χ_{ij}^r soll eine Kreuzung von i und j zwischen π^r und π^{r+1} erwingen (Beschränkungen 4.11 und 4.12). Allerdings soll $-\chi_{ij}^r$ ebenso sicher stellen, dass sich eben diese Linien nicht überkreuzen (Beschränkungen 4.13 und 4.14). Es gilt also die Äquivalenz $\chi_{ij}^r \Leftrightarrow x_{ij}^r \neq x_{ij}^{r+1}$, welche sich wie folgt durch vier Klauseln darstellen lässt.

$$\chi_{ij}^r \vee x_{ij}^r \vee \neg x_{ij}^{r+1} \vee d_i^r \vee d_j^r \qquad \text{für alle } i,j \in \mathcal{K}, \; r \in \{1,\dots,\lambda-1\},$$

$$\chi_{ij}^r \vee \neg x_{ij}^r \vee x_{ij}^{r+1} \vee d_i^r \vee d_j^r \qquad \text{für alle } i,j \in \mathcal{K}, \; r \in \{1,\dots,\lambda-1\},$$

$$\neg\chi_{ij}^r \vee x_{ij}^r \vee x_{ij}^{r+1} \vee d_i^r \vee d_j^r \qquad \text{für alle } i,j \in \mathcal{K}, \; r \in \{1,\dots,\lambda-1\},$$

$$\neg\chi_{ij}^r \vee x_{ji}^r \vee x_{ji}^{r+1} \vee d_i^r \vee d_j^r \qquad \text{für alle } i,j \in \mathcal{K}, \; r \in \{1,\ldots,\lambda-1\}.$$

Jede Linie i muss auf jeder Permutation π^r genau einem Blockkreuzungsintervall zugeordnet werden (Beschränkungen 4.15 und 4.16). Dies lässt sich durch fünf Klauseln garantieren.

$$\begin{aligned}
a_i^r \vee b_i^r \vee c_i^r \vee d_i^r &\qquad \text{für alle } i \in \mathcal{K}, \; r \in \{1,\ldots,\lambda\},\\
-a_i^r \vee -b_i^r \vee -c_i^r \vee d_i^r &\qquad \text{für alle } i \in \mathcal{K}, \; r \in \{1,\ldots,\lambda\},\\
-a_i^r \vee -b_i^r \vee c_i^r \vee d_i^r &\qquad \text{für alle } i \in \mathcal{K}, \; r \in \{1,\ldots,\lambda\},\\
a_i^r \vee -b_i^r \vee -c_i^r \vee d_i^r &\qquad \text{für alle } i \in \mathcal{K}, \; r \in \{1,\ldots,\lambda\},\\
-a_i^r \vee b_i^r \vee -c_i^r \vee d_i^r &\qquad \text{für alle } i \in \mathcal{K}, \; r \in \{1,\ldots,\lambda\}.
\end{aligned}$$

Die Bedingung, dass jede Linie i mit $b_i^r = True$ über jeder Linie j mit $c_j^r = True$ steht, also $b_i^r \wedge c_j^r \Rightarrow x_{ij}^r$, lässt sich wie folgt ausdrücken (Beschränkung 4.17).

$$-c_j^r \vee -b_i^r \vee x_{ij}^r \vee d_i^r \vee d_j^r \qquad \text{für alle } i,j \in \mathcal{K}, \; r \in \{1,\ldots,\lambda\}.$$

Um nur ordentliche Blockkreuzungen zuzulassen, haben wir bei dem ILP gefordert, dass die Mengen $\mathcal{B}$ und $\mathcal{C}$ zusammenhängen (Beschränkungen 4.18 und 4.19). Schon bei den linearen Beschränkungen ist ersichtlich, dass hierbei $x_{ij}^r \wedge x_{jk}^r \wedge b_i^r \wedge b_k^r \Rightarrow b_j^r$ und analog $x_{ij}^r \wedge x_{jk}^r \wedge c_i^r \wedge c_k^r \Rightarrow c_j^r$ gelten muss, was sich als Klauseln ausdrücken lässt.

$$\begin{aligned}
-x_{ij}^r \vee -x_{jk}^r \vee -b_i^r \vee -b_k^r \vee b_j^r \vee d_i^r \vee d_j^r \vee d_k^r &\qquad \text{für alle } i,j,k \in \mathcal{K}, \; r \in \{1,\ldots,\lambda\},\\
-x_{ij}^r \vee -x_{jk}^r \vee -c_i^r \vee -c_k^r \vee c_j^r \vee d_i^r \vee d_j^r \vee d_k^r &\qquad \text{für alle } i,j,k \in \mathcal{K}, \; r \in \{1,\ldots,\lambda\}.
\end{aligned}$$

Die Mengen $\mathcal{B}$ und $\mathcal{C}$ müssen ebenfalls benachbart sein, d.h. zwischen zwei Linien auf einer Permuatation π^r, von der eine in $\mathcal{B}$ und die andere in $\mathcal{C}$ liegt, darf keine Linie i mit $a_i^r = True$ liegen (Beschränkung 4.20). Es gilt also $x_{ij}^r \wedge x_{jk}^r \wedge b_i^r \wedge c_k^r \Rightarrow \neg a_j^r$ bzw. als Klausel:

$$-x_{ij}^r \vee -x_{jk}^r \vee -b_i^r \vee -c_k^r \vee -a_j^r \vee d_i^r \vee d_j^r \vee d_k^r \qquad \text{für alle } i,j,k \in \mathcal{K}, \; r \in \{1,\ldots,\lambda\}.$$

Zwei Linien i und j sollen sich zwischen zwei Permutationen π^r und π^{r+1} nur genau dann überkreuzen dürfen, falls $b_i^r = True$ und $c_j^r = True$, also $b_i^r \wedge c_j^r \Leftrightarrow \chi_{ij}^r$ (Beschränkungen 4.21, 4.22, 4.23 und 4.24).

$$\chi_{ij}^r \vee -b_i^r \vee -c_j^r \vee d_i^r \vee d_j^r \qquad \text{für alle } i,j \in \mathcal{K}, \; r \in \{1,\ldots,\lambda-1\},$$

$$-\chi_{ij}^r \vee -a_i^r \vee d_i^r \vee d_j^r \qquad \text{für alle } i,j \in \mathcal{K}, \; r \in \{1,\ldots,\lambda-1\},$$

$$-\chi_{ij}^r \vee -b_i^r \vee -b_j^r \vee d_i^r \vee d_j^r \qquad \text{für alle } i,j \in \mathcal{K}, \; r \in \{1,\ldots,\lambda-1\},$$

$$-\chi_{ij}^r \vee -c_i^r \vee -c_j^r \vee d_i^r \vee d_j^r \qquad \text{für alle } i,j \in \mathcal{K}, \; r \in \{1,\ldots,\lambda-1\}.$$

Jede Treffgruppe T_l muss auf genau einer Permutation π^r aktiv sein (Beschränkung 4.1). Dafür stellen wir zuerst durch eine Klaussel sicher, dass T_l mindestens einmal aktiv ist.

$$\bigvee_{r=1}^{\lambda} q_l^r \qquad \text{für alle } l \in \{1,\ldots,\mu\}.$$

Weiterhin darf es kein Paar (q_l^r, q_l^p) geben mit $q_l^r \wedge q_l^p = True$.

$$\neg q_l^r \vee \neg q_l^p \qquad \text{für alle } l \in \{1,\ldots,\mu\}, \; r,p \in \{1,\ldots,\lambda\}.$$

Wir müssen die Ordnung der Treffgruppen garantieren, d.h. eine Treffgruppe T_l darf nur auf einer Permutation π^r aktiv sein, falls T_{l-1} bereits auf einer Permutation π^p mit $p \leq r$ aktiv war (Beschränkungen 4.2 und 4.3).

$$\bigvee_{j=1}^{r} q_{l-1}^j \vee \neg q_l^r \qquad \text{für alle } l \in \{2,\ldots,\mu\}, \; r,p \in \{1,\ldots,\lambda\},$$

$$q_1^1.$$

Ein Charakter i existiert nur auf einer Permutation π^r, falls er in einem Treffen m ist, welches Teil einer Treffgruppe T_l ist und diese Treffgruppe durch $q_l^r = True$ auf π^r aktiv ist (Beschränkungen 4.4 und 4.5). Es gilt daher $q_l^r \Leftrightarrow \neg d_i^r$. Wir verwenden wieder $\mathcal{G}_i$ als die Menge aller l, für welche gilt: Der Charakter i kommt in einem Treffen aus T_l vor. Dann werden die Beschränkungen zu den Klauseln

$$\neg q_l^r \vee \neg d_i^r \qquad \text{für alle } l \in \{1,\ldots,\mu\}, \; r \in \{1,\ldots,\lambda\}, \; m \in T_l, \; i \in \mathcal{K}, \; \text{,falls } i \in C_m,$$

$$\bigvee_{l \in \mathcal{G}_i} q_l^r \vee d_i^r \qquad \text{für alle } r \in \{1, \ldots, \lambda\},\ i \in \mathcal{K}.$$

Falls auf einer Permutation π^r keine Treffgruppe aktiv ist, d.h. es gilt $\sum_{l=1}^{\mu} q_l^r = 0$ so müssen alle Charaktere existieren, welche auf der vorherigen Permutation π^{r-1} existieren (Beschränkungen 4.6 und 4.7).

$$\bigvee_{l=1}^{\mu} q_l^r \vee d_i^r \vee \neg d_i^{r-1} \qquad \text{für alle } r \in \{2, \ldots, \lambda\},\ i \in \mathcal{K},$$

$$\bigvee_{l=1}^{\mu} q_l^r \vee \neg d_i^r \vee d_i^{r-1} \qquad \text{für alle } r \in \{2, \ldots, \lambda\},\ i \in \mathcal{K}.$$

Wir fordern, dass zwei Treffgruppen T_l und T_o nicht auf derselben Permutation aktiv sein dürfen, falls für sie verschiedene Charaktere existieren, um die in Abbildung 5 und Abbildung 6 gezeigten Konflikte zu vermeiden (Beschränkung 4.25). Diese Beschränkung lässt sich darstellen als

$$q_l^r \vee q_o^r \qquad \text{für alle } r \in \{1, \ldots, \lambda\}, l, o \in \{1, \ldots, \mu\}, \text{ falls } l \neq o \text{ und } T_l, T_o \text{ nicht dieselben Charaktere enthalten,}$$

$$\neg q_l^r \vee \neg q_o^r \qquad \text{für alle } r \in \{1, \ldots, \lambda\}, l, o \in \{1, \ldots, \mu\}, \text{ falls } l \neq o \text{ und } T_l, T_o \text{ nicht dieselben Charaktere enthalten.}$$

Permutationen, welche nur für das Hinzufügen bzw. Entfernen erstellt wurden, dürfen nicht für Blockkreuzungen missbraucht werden (Beschränkungen 4.26 und 4.27). Dies wird durch folgende Klauseln sichergestellt.

$$\neg \chi_{jk}^r \vee d_i^r \vee \neg d_i^{r+1} \vee d_j^r \vee d_j^{r+1} \vee d_k^r \vee d_k^{r+1} \qquad \text{für alle } i, j, k \in \mathcal{K}, r \in \{1, \ldots, \lambda - 1\}, \text{ falls } i \neq j, i \neq k, k \neq j.$$

$$\neg \chi_{jk}^r \vee \neg d_i^r \vee d_i^{r+1} \vee d_j^r \vee d_j^{r+1} \vee d_k^r \vee d_k^{r+1} \qquad \text{für alle } i, j, k \in \mathcal{K}, r \in \{1, \ldots, \lambda - 1\}, \text{ falls } i \neq j, i \neq k, k \neq j.$$

Es bleibt nun nurnoch zu garantieren, dass Linien in einem Treffen zusammengeschoben werden (Beschränkungen 4.28 und 4.29).

$\neg q_l^r \vee x_{ij}^r \vee \neg x_{kj}^r$	für alle $i, j, k \in \mathcal{K},\ r \in \{1, \ldots, \lambda\}$, $l \in \{1, \ldots, \mu\},\ m \in T_l$, falls $i, k \in m$ und $j \notin m$,
$\neg q_l^r \vee \neg x_{kj}^r \vee x_{ij}^r$	für alle $i, j, k \in \mathcal{K},\ r \in \{1, \ldots, \lambda\}$, $l \in \{1, \ldots, \mu\},\ m \in T_l$, falls $i, k \in m$ und $j \notin m$.

Da die Beschränkungen des zweiten ILP Ansatzes lediglich zu Klauseln übersetzt wurden, folgt die Korrektheit der SAT-Instanz aus Satz 4.1.

6 Experimentelle Ergebnisse

Wir haben drei Verfahren zur Minimierung von Blockkreuzungen auf Storylines entwickelt, die wir nun im Bezug auf ihre Laufzeit testen werden. Die beiden ganzzahlig linearen Programme, welche wir als ILP1 und ILP2 bezeichnen, werden mithilfe des *Gurobi Optimizer 7.0.1*[6] in Kombination mit dem *Gurobi Python Interface* gelöst. Für die SAT-Instanz verwenden wir den *MINISAT SAT solver 2.2.0*[3]. Alle Tests wurden auf einer Intel Core i5-4670K CPU mit 4 Kernen und 8GB RAM durchgeführt.

6.1 Vergleich der Verfahren

Um die Implementierungen zu vergleichen, testen wir sie in drei unterschiedlichen Szenarien, welche sich durch die Anzahl der Charaktere und der Anzahl der Treffen unterscheiden. Dabei soll jedes Szenario die maximale Größe einer Instanz abschätzen, für welche eine Implementierung in hinnehmbarer Zeit eine optimale Lösung findet. Für jedes Szenario werden fünf voneinander unabhängige Instanzen getestet. Wir möchten dabei möglichst realistische Instanzen wählen, weswegen diese wie folgt generiert werden.

Eine Instanz wird aus der Anzahl der Charaktere κ, der Anzahl der Treffen ν und einem Zeitraum $\mathcal{Z} = \{1, \ldots, t_{max}\}$ erstellt. Es gilt dabei $s_m \geq 1$ und $e_m \leq t_{max}$ für die Start- und Endzeitpunkte eines jeden Treffens m. Der Zeitraum ist somit ein Maß für die Dichte der Treffen. Der Vorgang zum Erstellen eines Treffens verläuft wie folgt: Für ein Treffen m werden zufällige Start- und Endzeitpunkte gewählt. Wir fordern, dass ein Treffen nur über einen kleinen Zeitraum stattfinden darf, da zu lange Treffen die betroffenen Charaktere „blockieren“. Ist diese Bedingung nicht erfüllt, oder gilt $s_m \geq e_m$, so werden zwei neue Zeitpunkte zufällig ausgewählt. Wir wiederholen diesen Vorgang, bis zwei geeignete Zeitpunkte gefunden wurden. Das Auswählen der Charaktere für ein Treffen verläuft etwas komplizierter. Um reale Instanzen zu simulieren, nutzen wir eine Beobachtung. In echten Instanzen lassen sich die Charaktere meistens in *Gruppen* einteilen. Dies sind Charaktere, welche sich häufig untereinander, aber nur selten mit Charakteren anderer Gruppen treffen. Häufig können zwei gegenüberstehende Fraktionen identifiziert werden. So steht „Gut“ gegen „Böse“ oder „Helden“ gegen „Schurken“. Aus diesem Grund teilen wir alle Charaktere in zwei etwa gleich große Gruppen auf. Jeder Charakter ist in genau einer Gruppe. Treffen werden dann „gefüllt“, indem zuerst die Größe des Treffens g mit $g \leq \kappa$ zufällig bestimmt wird. Dann wählen wir zufällig einen Charakter i. Danach werden $g - 1$ weitere Charaktere gewählt, wobei die Wahrscheinlichkeit einen Charakter aus der Gruppe von i zu wählen deutlich erhöht ist. Sobald g Charaktere Teil des Treffens sind, wird überprüft, ob das Treffen zulässig ist. Falls ein zuvor generiertes Treffen m_1 zur selben Zeit stattfindet und $C_{m_1} \cap C_{m_2} \neq \emptyset$ gilt, ist das Treffen m_2 ungültig und wird verworfen. Wir wiederholen diesen Prozess, bis ν Treffen generiert wurden. Wir gehen

Tab. 1: Vergleich zwischen ILP1, ILP2 und SAT (z. l. bedeutet, dass die Berechnung der optimalen Lösung länger als 180 Sekunden benötigt).

$\kappa = 5, \nu = 10, t_{max} = 30$			
		V	Zeit
1	ILP1		0,39
	ILP2	1	0
	SAT	1	0,015
2	ILP1		0,34
	ILP2	3	0,01
	SAT	3	0
3	ILP1		2,81
	ILP2	4	0,19
	SAT	4	0,03
4	ILP1		1,12
	ILP2	4	0,16
	SAT	4	0,03
5	ILP1		30,66
	ILP2	4	0,01
	SAT	4	0,015

$\kappa = 7, \nu = 25, t_{max} = 90$			
		V	Zeit
1	ILP1		z.l.
	ILP2	5	16,84
	SAT	5	0,107
2	ILP1		z.l.
	ILP2	5	152,74
	SAT	5	0,123
3	ILP1		z.l.
	ILP2	5	2,51
	SAT	5	0,093
4	ILP1		z.l.
	ILP2	5	46,18
	SAT	5	0,06
5	ILP1		z.l.
	ILP2		z.l.
	SAT	7	0,231

$\kappa = 10, \nu = 50, t_{max} = 180$			
		V	Zeit
1	ILP1		z.l.
	ILP2		z.l.
	SAT	14	22,86
2	ILP1		z.l.
	ILP2		z.l.
	SAT	9	3,99
3	ILP1		z.l.
	ILP2		z.l.
	SAT	11	13,25
4	ILP1		z.l.
	ILP2		z.l.
	SAT	10	4,90
5	ILP1		z.l.
	ILP2		z.l.
	SAT	10	4,79

weiterhin davon aus, dass ein Charakter vom Beginn des Treffens, in dem er zuerst vorkommt, bis zum Ende des Treffens, in dem er zuletzt vorkommt, existiert, da es so in den meisten realen Instanzen ist. Möglichst realitätsnahe Instanzen zu generieren ist schwierig, weswegen das Lösen so erstellter Instanzen auch für kleinere κ and ν deutlich länger benötigt, als das Lösen wirklicher realer Instanzen. Dennoch erlaubt diese Methode einen Vergleich zwischen den drei Ansätzen.

Da ILP2 und die SAT-Instanz nicht die minimale Anzahl von Blockkreuzungen liefern, sondern lediglich entscheiden, ob eine Instanz mit λ Permutationen lösbar ist, müssen beide mehrmals ausgeführt werden. Wir verwenden folgende Strategie beginnend mit $\lambda = 1$: Wird keine den Test erneut mit $\lambda = \lambda + 2$. Sobald eine Lösung gefunden wurde, versuchen wir es mit $\lambda - 1$ erneut. Findet ILP2 bzw. SAT nun eine Lösung, ist $\lambda - 1$ die minimale Anzahl an Permutationen. Falls nicht, ist λ optimal. Wir bestimmen die CPU-Zeit von ILP2 und SAT durch Summieren der CPU-Zeiten aller Versuche.

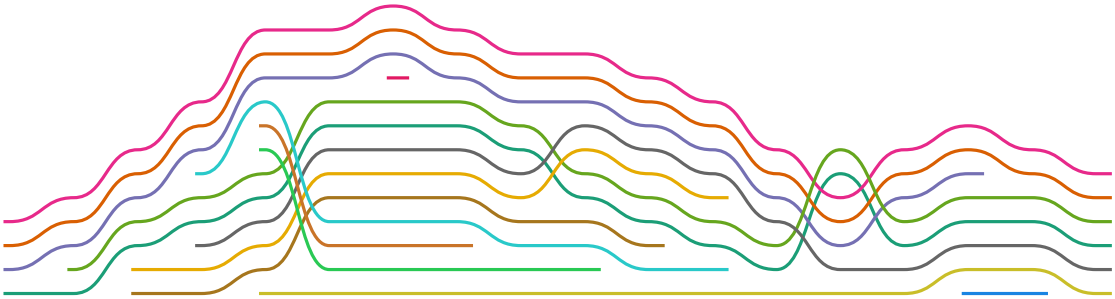
Wir betrachten neben der Berechnungszeit auch die Anzahl der Versuche (V) bei ILP2 bzw. SAT. Der Vergleich ist in Tabelle 1 dargestellt. Um die Einteilung der Charaktere in zwei Gruppen zu rechtfertigen, wurde auch eine Instanz mit $\kappa = 10$, $\nu = 50$ und $t_{max} = 180$ erstellt, welche auf diese Einteilung verzichtet. ILP1 und ILP2 konnten diese Instanz nicht lösen und es war auch nicht möglich, die benötigte CPU-Zeit abzuschätzen. SAT lieferte nach 13 Versuchen eine optimale Lösung in 63,5 Sekunden.

6.2 Lösen realer Instanzen

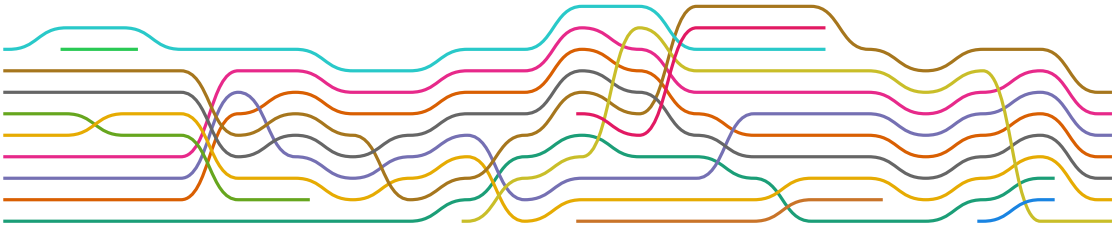
Wir verwenden für unsere Tests die drei Filminstanzen [15] „The Matrix“, die originale Trilogie von „Star Wars“ und „Inception“. Diese wurden auch von Gronemann et al. [5] verwendet und gezeichnet. Dabei ist zu Beachten, dass sie neben der gegebenen Instanz von „Inception“ auch eine Lösung für „Inception-sf“ lieferten, um Vergleiche mit vorherigen Arbeiten [17] [16] [11] zu erhalten. Dies ist eine modifizierte Version, mit den folgenden drei Änderungen:

Die Linie des Charakters „Mal“ darf Abkürzungen in der Storyline nehmen, da der Charaktere über einen langen Zeitraum existiert, allerdings nur selten an Treffen teilnimmt. In diesem Zeitraum wird er daher als nicht existent gesehen. Weiterhin gibt die Instanz aus [15] Treffen zum Ende des Films vor, die so im Film allerdings nicht vorkommen, weswegen die Treffen am Ende entsprechend angepasst wurden. Zuletzt enthält die Instanz aus [15] die zwei zusätzlichen Nebencharaktere „Arch“ und „Asian“, welche in „Inception-sf“ nicht berücksichtigt werden.

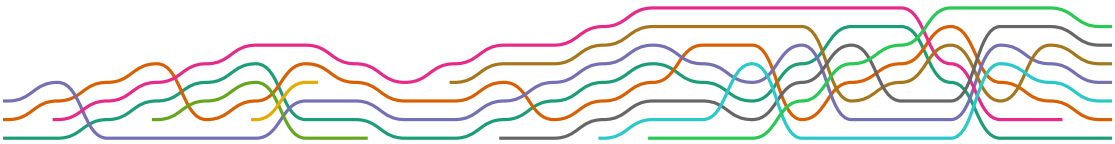
Diese vier Instanzen wurden von uns mit dem SAT-Ansatz mit minimalen Blockkreuzungen gelöst. Die Ergebnisse sind in Abbildung 7 dargestellt.



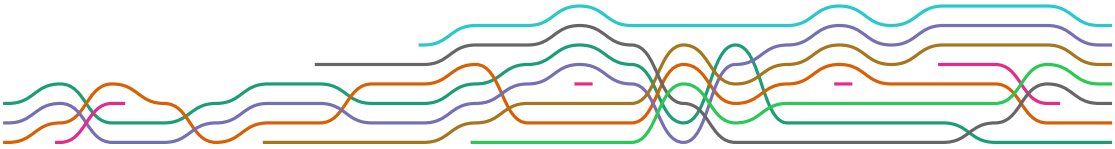
(a) „The Matrix“ mit 4 Blockkreuzungen. Berechnet mit dem SAT-Ansatz in 6,36 Sekunden.



(b) Die originale Trilogie von „Star Wars“ mit 10 Blockkreuzungen. Berechnet mit dem SAT-Ansatz in 6,67 Sekunden.



(c) „Inception“ mit 12 Blockkreuzungen. Berechnet mit dem SAT-Ansatz in 4,25 Sekunden.



(d) „Inception-sf“ mit 9 Blockkreuzungen. Berechnet mit dem SAT-Ansatz in 4,11 Sekunden.

Abb. 7: Die vier Filminstanzen mit minimalen Blockkreuzungen.

Literaturverzeichnis

[1] Fractal art: Beauty and mathematics. http://www.ams.org/mathimagery/thumbnails.php?album=13, 2016. accessed 16.02.2017.

[2] V. Bafna and P. A. Pevzner. Sorting by transpositions. *SIAM J. Discr. Math.*, 11(2):224–240, 1998.

[3] N. Eén and N. Sörensson. Minisat sat solver. http://minisat.se, 2003.

[4] M. Fink, S. Pupyrev, and A. Wolff. Ordering metro lines by block crossings. *J. Graph Algorithms Appl.*, 19(1):111–153, 2015.

[5] M. Gronemann, M. Jünger, F. Liers, and F. Mambelli. Crossing minimization in storyline visualization. In Y. Hu and M. Nöllenburg, editors, *Graph Drawing and Network Visualization: 24th International Symposium, GD 2016, Athens, Greece, September 19-21, 2016, Revised Selected Papers*, pages 367–381, Cham, 2016. Springer International Publishing.

[6] Gurobi Optimization. Gurobi optimizer. http://www.gurobi.com.

[7] N. W. Kim, S. K. Card, and J. Heer. Tracing genealogical data with timenets. In *Proceedings of the International Conference on Advanced Visual Interfaces*, AVI '10, pages 241–248, New York, NY, USA, 2010. ACM.

[8] A. Koop and H. Moock. *Lineare Optimierung - eine anwendungsorientierte Einführung in Operations Research.* Spektrum Akademischer Verlag, 2007.

[9] I. Kostitsyna, M. Nöllenburg, V. Polishchuk, A. Schulz, and D. Strash. On minimizing crossings in storyline visualizations. In E. D. Giacomo and A. Lubiw, editors, *Proc. 23rd Int. Symp. Graph Drawing (GD'15)*, volume 9411 of *LNCS*, pages 192–198. Springer, Heidelberg, 2015.

[10] M. Krzywinski. The art of pi. http://mkweb.bcgsc.ca/pi/art/, 2013. accessed 16.02.2017.

[11] S. Liu, Y. Wu, E. Wei, M. Liu, and Y. Liu. Storyflow: Tracking the evolution of stories. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2436–2445, 2013.

[12] C. J. Minard. The russian campaign 1812−1813. http://visual.ly/napoleons-march-moscow-war-1812, 2013. accessed 21.02.2017.

[13] C. W. Muelder, T. Crnovrsanin, A. Sallaberry, and K.-L. Ma. Egocentric storylines for visual analysis of large dynamic graphs. In *Big Data, 2013 IEEE International Conference on*, pages 56–62. IEEE, 2013.

[14] R. Munroe. Movie narrative charts. `https://xkcd.com/657/`, 2013. accessed 16.02.2017.

[15] Y. Tanahashi. Movie data set. `https://vis.cs.ucdavis.edu/~tanahashi/data_downloads/storyline_visualizations/story_data.tar`, 2013.

[16] Y. Tanahashi, C.-H. Hsueh, and K.-L. Ma. An efficient framework for generating storyline visualizations from streaming data. *IEEE transactions on visualization and computer graphics*, 21(6):730–742, 2015.

[17] Y. Tanahashi and K.-L. Ma. Design considerations for optimizing storyline visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2679–2688, 2012.

[18] T. C. van Dijk, M. Fink, N. Fischer, F. Lipp, P. Markfelder, A. Ravsky, S. Suri, and A. Wolff. Block crossings in storyline visualizations. In Y. Hu and M. Nöllenburg, editors, *Graph Drawing and Network Visualization: 24th International Symposium, GD 2016, Athens, Greece, September 19-21, 2016, Revised Selected Papers*, pages 382–398, Cham, 2016. Springer International Publishing.

Erklärung

Hiermit versichere ich die vorliegende Abschlussarbeit selbstständig verfasst zu haben, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, und die Arbeit bisher oder gleichzeitig keiner anderen Prüfungsbehörde unter Erlangung eines akademischen Grades vorgelegt zu haben.

Würzburg, den 23. Februar 2017

. . . . . .
Peter Markfelder