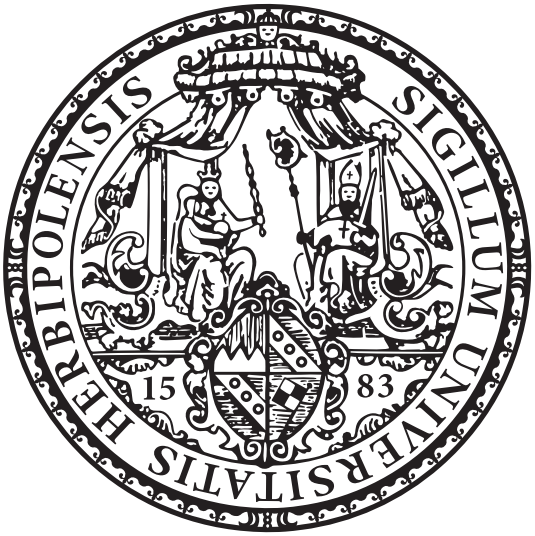
Masterarbeit

Storyline-Visualisierungen für wissenschaftliche Kollaborationsgraphen

Tim Herrmann

Abgabedatum:	10. Juni 2022
Betreuer:	Prof. Dr. Alexander Wolff
	Felix Klesen, M. Sc.
	Tim Hegemann, M. Sc.



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

Zusammenfassung

Bei einer Storyline-Visualisierung handelt es sich um eine Darstellung, die aufzeigt wie verschiedene Charaktere oder Entitäten über den zeitlichen Verlauf miteinander interagieren. Dabei wird jeder Charakter durch eine x-monotone Kurve repräsentiert. Interagieren zwei oder mehrere Charaktere miteinander, so werden sie in der Visualisierung zu den entsprechenden Zeitintervallen zu Nachbarn gemacht und nahe zusammengeführt oder verbunden. In dieser Arbeit wird ein Verfahren entwickelt, welches die Kollaboration von Wissenschaftlern mithilfe einer Storyline-Visualisierung aufzeigt, und die Anzahl der dabei entstehenden Blockkreuzungen minimiert. Dafür wird ein heuristisches Greedy-Verfahren entwickelt, das ein Problem mit k Wissenschaftlern und m Veröffentlichungen mithilfe einer Ähnlichkeitsmatrix in $O(mk^3)$ Zeit löst. Außerdem werden verschiedene Methoden zur Wahl der Startpermutation der Charaktere entwickelt und miteinander und der optimalen Lösung verglichen. Die Greedy-Verfahren, die mit einer zufälligen oder einer heuristisch gewählten Startpermutation beginnen, berechnen in sehr kurzer Zeit eine passende Storyline-Visualisierung, erzeugen dabei aber ein Vielfaches der benötigten Blockkreuzungen. Wenn der Greedy-Algorithmus zehn mal mit jeweils zufälliger Startpermutation durchgeführt wird und das beste dieser Ergebnisse zurückgegeben wird, ist die Anzahl der Blockkreuzungen deutlich kleiner. Die Laufzeit ist dabei wie erwartet etwas höher, aber für die meisten Anwendungen unproblematisch. Eine ähnliche Version, welche für alle möglichen Startpermutationen eine Lösung berechnet und die beste zurückgibt, erzeugt mit Abstand die wenigsten Blockkreuzungen. Die Berechnung benötigt allerdings faktorielle Laufzeit, weshalb dieser Ansatz nur für kleinere Problemstellungen geeignet ist und gegenüber exakten Lösungsmethoden nicht vorzuziehen ist.

Abstract

A storyline visualization shows how different characters or entities interact with each other over a period of time. Every character is represented by a x-monotone curve. If two or more characters interact with each other they will run close to each other. The objective of this thesis is to automatically generate storyline visualizations which display how different authors have collaborated while minimizing the amount of block crossings. In order to achieve this goal, we introduce a greedy heuristic algorithm which solves a problem with k authors and m publications in $O(mk^3)$ by using a similarity matrix. Furthermore, we develop different methods to choose fitting starting permutations. The greedy algorithms which start with random or heuristic starting permutations compute viable storylines quickly, but the amount of block crossing they generate are many times higher than necessary. The variation which executes the greedy algorithm ten times with a random starting permutation and returns the best result performs a lot better. As expected, the runtime is a bit longer, but still sufficiently fast. A similar version which simply tries every possible start permutation and returns the best result performs best. Since it runs in factorial time, it is only suitable for storylines with few characters and calculating the optimal solution is generally the better option.

Inhaltsverzeichnis

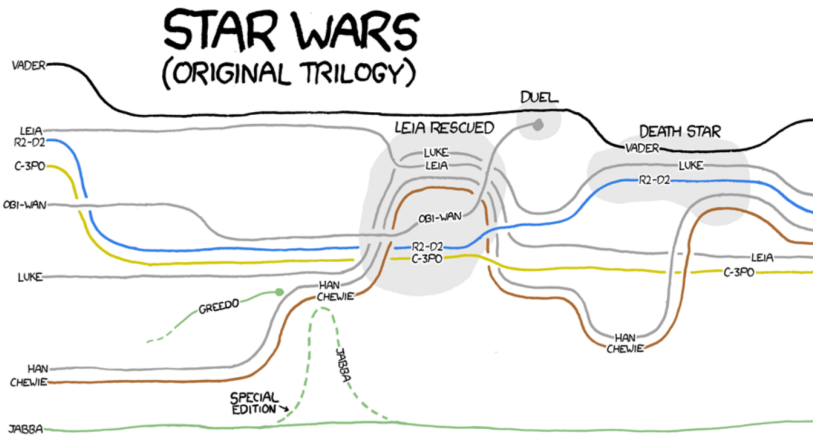
1. Einführung

Die Graphentheorie beschäftigt sich unter anderem mit der Visualisierung von Daten. Sie befasst sich beispielsweise mit dem Zeichnen von U-Bahnplänen. Dabei sollen alle notwendigen Informationen möglichst übersichtlich dargestellt werden. XKCD [Mun] veröffentlichte handgezeichnete Storyline-Visualisierungen, die aufzeigen wie unterschiedliche Charaktere aus Filmen miteinander interagieren. Diese Art von Darstellung eignet sich allerdings nicht nur für Filme, sondern auch für Geschichten generell, wie zum Beispiel Erzählungen, Theaterstücke, Bücher, Comics, Serien und mehr. Sie kann generell verschiedene Dinge darstellen, Vorraussetzung ist nur, dass es Entitäten gibt, die über einen gewissen Zeitraum hinweg miteinander interagieren. Dabei werden die Entitäten durch Linien dargestellt, die zusammengeführt werden, wenn sie miteinander interagieren. In dieser Arbeit soll die Kollaboration von Wissenschaftlern mithilfe von Storyline-Visualisierungen veranschaulicht werden. Bei den Wissenschaftlern handelt es sich dann um die Entitäten und zwei oder mehr Wissenschaftler interagieren miteinander, wenn sie an derselben Veröffentlichung gearbeitet haben.

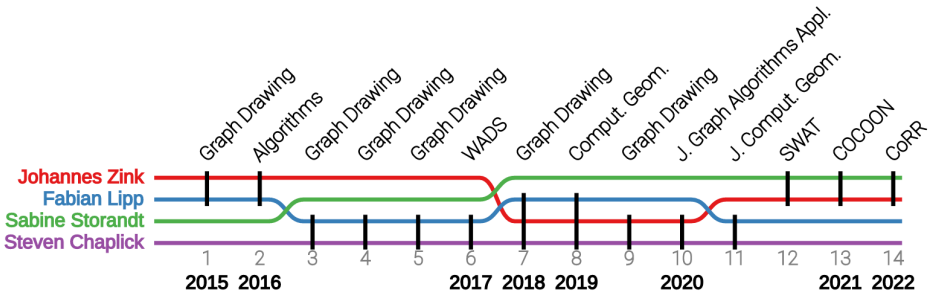
1.1. Storyline-Visualisierung

Eine Storyline-Visualisierung stellt dar, wie verschiedene Charaktere im Laufe der Zeit miteinander interagieren. Dabei wird der Verlauf der Zeit durch die x-Achse repräsentiert und jeder Charakter wird durch eine x-monotone Kurve dargestellt. Interagieren zwei oder mehr Charaktere miteinander, so werden sie zu dem passenden Zeitpunkt zueinander geführt oder durch eine vertikale Linie verbunden. Damit dabei keine zusätzlichen Charaktere miteinbezogen werden, muss sichergestellt werden, dass zwischen den Teilnehmern der Interaktion keine anderen Charaktere sind. Um dies zu erreichen kreuzen sich die Charakterlinien falls nötig. Obwohl Storyline-Visualisierungen neben dem Interagieren von Charakteren aus Geschichten auch andere Zusammenhänge darstellen können, werden in dieser Arbeit weiterhin die ursprünglichen Begrifflichkeiten verwendet und die Visualisierung enthält daher Charaktere, sie sich im Verlauf der Zeit gegebenenfalls treffen. Abbildung 1.1 veranschaulicht, wie verschiedene Storyline-Visualisierungen aussehen können.

Hinsichtlich der Überschaubarkeit solcher Darstellungen gibt es verschiedene Schwerpunkte, die berücksichtigt werden können. Häufig geht es um das Verringern von Kreuzungen, aber auch das generelle Design wurde untersucht. Unter anderem wurde von Tanahashi und Ma [TM12] das Vermeiden von großen leeren Flächen und die Form der Übergänge der Linien begutachtet. Silvia et al. [SAHW15] haben eine andere Vorgehensweise aufgeführt und ein Layout vorgestellt, das mithilfe eines Kräftesystems die einzelnen Elemente dynamisch setzt. Mithilfe von ganzzahliger linearen Programmierung



(a) Ausschnitt einer Storyline-Visualisierung aus XKCD [Mun].



(b) Storyline-Visualisierung für die Kollaboration von Wissenschaftlern.

Abb. 1.1.: Beispiele für verschiedene Arten von Storyline-Visualisierungen.

(Integer Linear Programming oder kurz ILP) minimieren Fröschl und Nöllenburg [FN17] die vertikalen Schwankungen der x-monotonen Linien. Des Weiteren kann dabei auch die Anzahl der Kreuzungen gewichtet mit minimiert werden. Auch Gronemann et al. [GJLM16] minimieren Kreuzungen mithilfe eines ILP und zeigen, dass es mittelgroße Daten mit akzeptabler Berechnungsdauer lösen kann. Falls alle Treffen einer Storyline aus immer genau zwei Charakteren bestehen und die Treffen als Baum dargestellt werden können, finden Kostitsyna et al. [KNP+15] für eine Storyline mit k Charakteren und m Treffen immer eine Visualisierung mit $O(k \log k)$ Kreuzungen. Des Weiteren zeigen Sie, dass es Storylines gibt, die diese Einschränkungen erfüllen und $\Omega(k \log k)$ Kreuzungen benötigen. Außerdem wird ein parametrisiertes Verfahren vorgestellt, welches die Anzahl der Kreuzungen für den allgemeinen Fall in $O(k!^2 k \log k + k!^2 m)$ Zeit minimiert.

1.2. Blockkreuzung

Bei einer Blockkreuzung handelt es sich um eine Kreuzung von zwei benachbarten Strängen von Linien. Dabei besteht jeder Strang aus einer Teilmenge der aktuellen Anordnung

der Charaktere, welche während der Kreuzung parallel zueinander bleiben. Durch das Nutzen von Blockkreuzungen können dementsprechend mehrere Charaktere in einem Schritt ihre Position wechseln, was oftmals weniger verwirrend ist als viele paarweise Kreuzungen. In Abbildung 1.2 wird das Verhalten einer Blockkreuzung veranschaulicht. Obwohl Blockkreuzungen oftmals übersichtlicher sind als viele paarweise Kreuzungen, ist es dennoch häufig sinnvoll auch die Anzahl der Blockkreuzungen zu minimieren (siehe Abbildung 1.3).

Van Dijk et al. [vDFF+17] haben zwei parametrisierte Algorithmen veröffentlicht, welche die Anzahl der Blockkreuzungen in einer Storyline-Visualisierung minimieren. Des Weiteren wurde ein heuristisches Greedy-Verfahren vorgestellt und ausgewertet, welches die Anzahl der Blockkreuzungen minimiert, falls jedes Treffen der Storyline aus genau zwei Charakteren besteht. Später modellieren Van Dijk et al. [vDLMW18] ein Erfüllbarkeitsproblem der Aussagenlogik (Boolean Satisfiability Problem oder kurz SAT) und lösen sie mit dafür entwickelten Algorithmen (SAT-Solvern) um eine optimale Lösung zu finden.

1.3. Eigene Arbeit

In dieser Arbeit wurde eine Webapplikation entwickelt, die für eine Suche von Wissenschaftlern automatisiert eine Storyline-Visualisierung erstellt, welche aufzeigt, wie diese bisher miteinander kollaboriert haben. Dabei soll die Anzahl der Blockkreuzungen der Visualisierung minimiert werden. Dafür wurde ein heuristischer Greedy-Algorithmus entwickelt, der mithilfe einer Ähnlichkeitsmatrix die auszuführenden Blockkreuzungen auswählt. Des Weiteren wurden verschiedene Methoden zur Auswahl der Startpermutation der Storyline-Visualisierung vorgestellt und die verschiedenen Ergebnisse miteinander und der optimalen Lösung verglichen. Die optimalen Lösungen wurden mithilfe des parametrisierten Algorithmus von van Dijk et al. [vDFF+17] berechnet und die notwendigen Informationen über die Autoren und ihre Veröffentlichungen wurden mithilfe der Publikationsdatenbank *dblp: computer science bibliography* [Ley] erfasst.

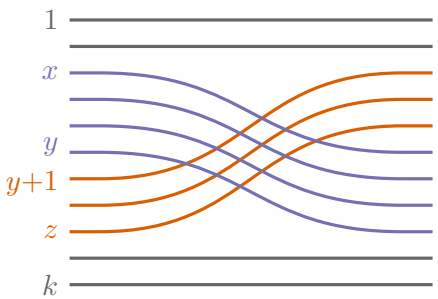
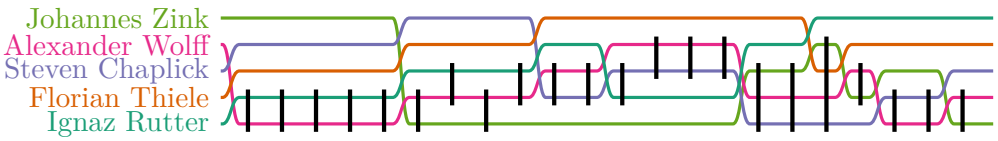
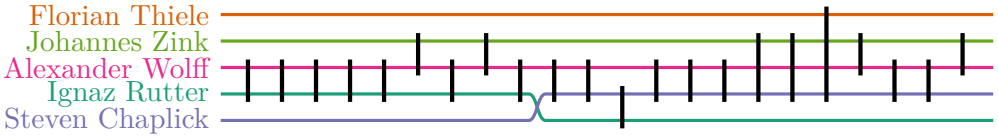


Abb. 1.2.: Die Stränge $(x, \ldots, y)$ und $(y+1, \ldots, z)$ kreuzen sich in der Blockkreuzung (x, y, z) (Abbildung aus [vDFF+17]).



(a) Storyline-Visualisierung mit vielen Blockkreuzungen.



(b) Storyline-Visualisierung mit weniger Blockkreuzungen.

Abb. 1.3.: Die Gegenüberstellung von zwei verschiedenen Visualisierungen der gleichen Storyline zeigt, dass das Minimieren der Anzahl der Blockkreuzungen die Lesbarkeit verbessern kann.

2. Grundlagen

Bevor die eigentlichen Arbeiten zur Lösung des Problems aufgeführt werden, sollen in diesem Kapitel die notwendigen Grundlagen geklärt werden. Dazu wird das Problem formal definiert und ein Algorithmus zur Lösung eines ähnlichen Problems vorgestellt. Viele der dabei verwendeten Bezeichnungen und Formulierungen stammen aus der wissenschaftlichen Veröffentlichung von van Dijk et al. [vDFF$^+$17].

2.1. Problembeschreibung

Eine Storyline sei definiert als $S = (C, M)$, wobei $C = \{1, \ldots, k\}$ die Menge der Charaktere und $M = (M_1, \ldots, M_m)$ mit $M_i \subseteq C$ und $|M_i| \geq 2$ für $i = 1, 2, \ldots, m$ eine Abfolge von Treffen aus mindestens zwei Charakteren darstellt. Des Weiteren handelt es sich bei der Permutation π um eine beliebige Anordnung der Charaktere. Sie wird verwendet um die vertikale Anordnung der Charaktere zu einem gewissen Zeitpunkt anzugeben. Demnach ist (c_2, c_4, c_3, c_1) eine der möglichen Permutationen der Charaktere $\{c_1, c_2, c_3, c_4\}$. Ein Treffen M_i kann dann zu einer Permutation π der k Charaktere stattfinden, wenn alle Charaktere aus M_i eine Teilpermutation in π bilden. Das heißt, M_i ist dann in der Permutation π möglich, falls eine beliebige Permutation von M_i Teil von π ist. Damit alle Treffen aus M stattfinden können, müssen sich die Charakterlinien der Visualisierung unter Umständen kreuzen. Anstatt viele paarweise Kreuzungen zu verwenden, kann die Visualisierung allerdings auch mithilfe von Blockkreuzungen gezeichnet werden. Diese sollen die Visualisierung übersichtlicher machen und können mithilfe von drei Parametern definiert werden. Eine Blockkreuzung $b = (x, y, z)$ sei daher definiert durch den Beginn des oberen Stranges x, dem Ende des oberen Stranges y und dem Ende des unteren Stranges z (siehe Abbildung 1.2).

Definition 2.1. *Die Problemstellung, bei der die Anzahl der Blockkreuzungen einer Storyline-Visualisierung minimiert werden soll, nennt sich Storyline Block Crossing Minimization (SBCM).*

Bei d-SBCM handelt es sich um spezielle Fälle, bei denen jedes Treffen aus höchstens d Charakteren besteht. Dabei ist d eine beliebige Konstante. Besonders für das Lösen von 2-SBCM sind mittlerweile einige Vorgehensweisen bekannt ([KNP$^+$15], [vDFF$^+$17]).

2.2. Greedy-Algorithmus für 2-SBCM

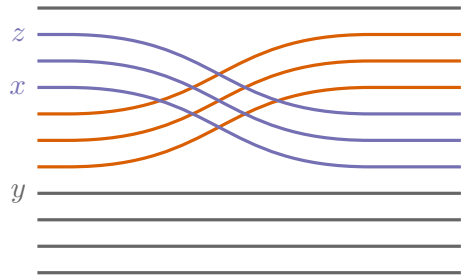
Van Dijk et al. [vDFF$^+$17] stellen unter anderem auch ein Greedy-Algorithmus für 2-SBCM vor. Dieser hat den Algorithmus zur Minimierung der Blockkreuzungen im allgemeinen Fall inspiriert und wird deshalb hier kurz vorgestellt.

Für eine Instanz $S = (C, M)$ wird eine Liste B erstellt, in der die nötigen Blockkreuzungen gespeichert werden, um alle Treffen aus M möglich zu machen. Zuerst beginnt der Algorithmus mit einer zufälligen Permutation π_0 der Charaktere aus C. In jedem Schritt i werden alle Treffen aus dem Beginn von M entfernt, die in der aktuellen Permutation π_i stattfinden können. Danach wird eine Blockkreuzung b ausgewählt, sodass die resultierende Permutation die maximale Anzahl der Treffen aus dem Beginn von M ermöglicht. Anschließend wird b zur Liste der Blockkreuzungen B hinzugefügt und die Iteration ist damit beendet. Dieser Prozess wird wiederholt bis M leer ist und (π_0, B) zurückgegeben wird. Die endgültige Storyline-Visualisierung kann dann durch Nachverarbeitung dieser Ergebnisse erhalten werden.

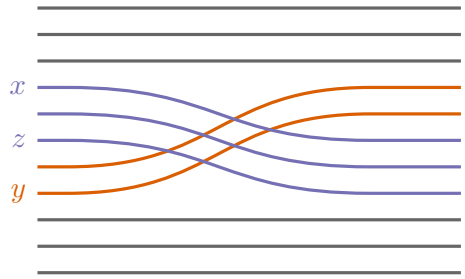
Um die Blockkreuzung zu finden, die die maximale Anzahl der bevorstehenden Treffen ermöglicht, überprüft der Algorithmus alle Blockkreuzungen, deren resultierende Permutation mindestens das nächste Treffen erlaubt. Sei $\{c_1, c_2\}$ das nächste Treffen in M, x und y die Position dieser beiden Charaktere in der aktuellen Permutation und $x < y$ (ohne Verlust der Allgemeingültigkeit), dann sind alle relevanten Blockkreuzungen in:

$$\{(z, x, y-1) : 1 \leq z \leq x\} \cup \{(x, z, y) : x \leq z < y\} \cup \{(x+1, y-1, z) : y \leq z \leq k\}$$

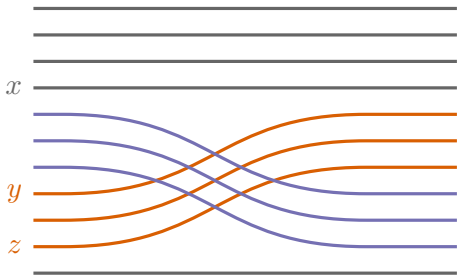
Abbildung 2.1 veranschaulicht die drei möglichen Möglichkeiten die Blockkreuzung zu wählen. Die Anzahl der relevanten Blockkreuzungen für jeden Schritt beträgt deshalb $k+1$. Mithilfe einer speziellen Datenstruktur können auf diese Weise die Blockkreuzungen der Storyline-Visualisierung in $O(km)$ Zeit gefunden werden.



(a) Blockkreuzung $(z, x, y-1)$ für $1 \leq z \leq x$.



(b) Blockkreuzung (x, z, y) für $x \leq z < y$.



(c) Blockkreuzung $(x+1, y-1, z)$ für $y \leq z \leq k$.

Abb. 2.1.: Visualisierung der drei möglichen Fälle um die zwei Charaktere mithilfe einer Blockkreuzung zusammenzuführen.

3. Greedy-Algorithmus

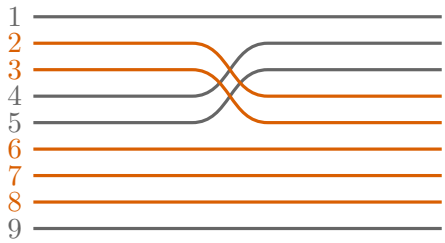
In diesem Kapitel wird der Greedy-Algorithmus beschrieben, der die Anzahl der Blockkreuzungen der Storyline-Visualisierung minimiert. Dazu wird zuerst die Idee vorgestellt und ein Ähnlichkeitsmaß eingeführt. Anschließend wird die Funktionsweise des Algorithmus mithilfe von Pseudocode und einem Beispiel erklärt und die Laufzeit analysiert. Zum Schluss werden noch Erweiterungen beschrieben, welche die Storyline-Visualisierung möglicherweise weiter verbessern.

3.1. Idee

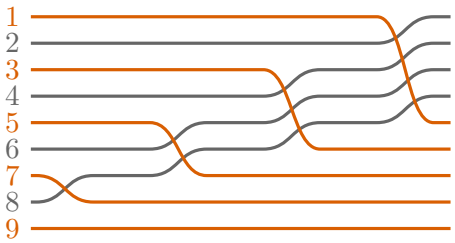
Ähnlich wie der Greedy-Algorithmus in Abschnitt 2.2 sollen in jeder Iteration die Treffen, die in der aktuellen Permutationen stattfinden können, übersprungen werden. Wird ein Treffen M_i erreicht, welches momentan nicht möglich ist, soll die Blockkreuzung gewählt werden, die in diesem Moment am besten erscheint. Im Gegensatz zu der Variante für 2-SBCM kann jedoch mit einer Blockkreuzung das nächste Treffen oft nicht ermöglicht werden. Demnach gibt es nicht nur $k + 1$ relevante Blockkreuzungen, sondern deutlich mehr Möglichkeiten das nächste Treffen zu ermöglichen.

Seien $\pi_{i,1}, \ldots, \pi_{i,g}$ die Teilpermutationen der aktuellen Permutation π, die aus den Charakteren bestehen, die in M_i sind, wobei jede Teilpermutation aus einem maximalen zusammenhängenden Intervall von Charakteren aus π besteht. Die Anzahl g solcher Teilpermutationen beträgt $1 < g \leq \lceil \frac{k}{2} \rceil$ und jede dieser Teilpermutationen wird in $G = (G_1, \ldots, G_g)$ gespeichert. Falls es genau eine solche Teilpermutation gibt, sind alle Teilnehmer des Treffens M_i beieinander und M_i kann stattfinden. Ansonsten sollen die Teilpermutationen nach und nach mithilfe von Blockkreuzungen zusammengeführt werden (siehe Abbildung 3.1).

Da mit jeder Blockkreuzung zwei Teilpermutationen vereinigt werden können, werden für jedes Treffen genau $g - 1 \leq \lceil \frac{k}{2} \rceil - 1$ Blockkreuzungen benötigt. Deshalb sollen die Blockkreuzungen so gewählt werden, dass die Anzahl der Teilpermutationen g für die kommenden Treffen möglichst klein ist. Alle möglichen Zusammensetzungen der g Teilpermutationen zu überprüfen kostet allerdings $O(k!)$ Zeit. Deshalb wird ein anderer Ansatz benötigt. Für alle möglichen Charakterpaare wird daher ein Ähnlichkeitsmaß berechnet, welches einen Richtwert geben soll, wie oft sie sich in nächster Zeit treffen. Anhand dieses Ähnlichkeitsmaßes wird dann entschieden, wie die g Gruppen zusammengeführt werden um das kommende Treffen zu ermöglichen. Dabei sollen zwei Charaktere zu Nachbarn gemacht werden, die sich in nächster Zeit häufig treffen, damit sie später nicht zwei Teilpermutationen erstellen, sondern bereits in der gleichen Teilpermutation sind.



(a) Die zwei orangefarbenen Teilpermutationen des Treffens $\{2, 3, 6, 7, 8\}$ werden mithilfe einer Blockkreuzungen zusammengeführt.



(b) Das Treffen $\{1, 3, 5, 7, 9\}$ erzeugt für diese Permutation die maximale Anzahl an Teilpermutationen. Mithilfe von vier Blockkreuzungen können die fünf Teilpermutationen zusammengeführt werden.

Abb. 3.1.: Zwei Beispiele zum Zusammenführen der Charaktere aus M_i.

3.2. Ähnlichkeitsmaß

Ist ein Treffen M_i in der aktuellen Permutation nicht möglich, dann wird anhand des Ähnlichkeitsmaßes entschieden, welche Teilpermutationen zusammengeführt werden sollen. Deshalb wird im Folgenden erklärt, wie die Ähnlichkeitsmatrix berechnet wird, die das Ähnlichkeitsmaß der Charakterpaare speichert. Außerdem wird gezeigt, wie die Matrix verwendet wird um das Charakterpaar auszuwählen, welches mit einer Blockkreuzung zu Nachbarn gemacht werden soll.

3.2.1. Berechnung der Ähnlichkeitsmatrix

Für die Berechnung der Ähnlichkeitsmatrix legen wir ein Lookahead t fest, der bestimmt, wie viele der nächsten Treffen für die Matrix relevant sind. Des Weiteren wird ein Ähnlichkeitsmaß $a_{(p,q)}$ für jedes Paar (p, q) mit $p, q \in C$ definiert und mit Null initialisiert. Da der Charakter p später weitere Bedingungen erfüllen muss, spielt die Reihenfolge der Tupel (p, q) eine Rolle. Um das Ähnlichkeitsmaß der Charakterpaare dann anzupassen, werden die nächsten t Treffen begutachtet. Ist t größer als die Anzahl der verbleibenden Treffen, so werden einfach alle restlichen Treffen betrachtet. Für jedes dieser zukünftigen Treffen M_j mit $i < j \leq \min(i + t, m - i)$ wird das Ähnlichkeitsmaß $a_{(p,q)}$ um $1/(j - i)$ erhöht, wenn sich p und q in M_j treffen. So soll sichergestellt werden, dass die Treffen, die früher stattfinden, priorisiert werden.

3.2.2. Finden eines Charakterpaares mithilfe der Ähnlichkeitsmatrix

Das Charakterpaar (p, q) mit dem höchsten Ähnlichkeitsmaß und ihre respektiven Teilpermutationen sollen mithilfe einer Blockkreuzung zusammengeführt werden. Demnach sollen sich die Charaktere in verschiedenen Teilpermutationen befinden, also sei $p \in G_v$ und $q \in G_w$ für $v \neq w$ und $v, w \in \{1, 2, \ldots, g\}$. Allerdings ist dies für beliebige Charakterpaare aus verschiedenen Teilpermutationen leider nicht immer möglich (siehe Abbildung 3.2). Seien $\text{top}(G_j)$ der oberste und erste Charakter und $\text{bot}(G_j)$ der letzte und

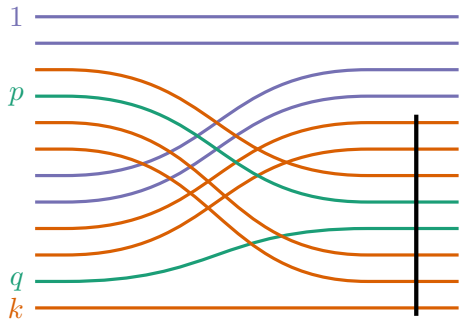


Abb. 3.2.: Es werden zwei Blockkreuzungen benötigt, um sowohl das Charakterpaar (p, q) als auch ihre jeweiligen orangefarbenen Teilpermutationen zusammenzuführen.

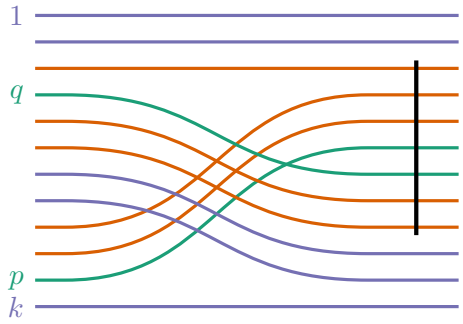


Abb. 3.3.: Befindet sich ein Charakter des ausgewählten Paares (p, q) am Rand seiner Teilpermutation, dann kann das Paar und die respektiven Teilpermutationen (orange) mit einer Blockkreuzung zusammengeführt werden.

unterste Charakter der Teilpermutation G_j für $j = 1, 2, \ldots, g$. Dann kann sichergestellt werden, dass nur eine Blockkreuzung benötigt wird und trotzdem zwei Teilpermutationen und zwei Charaktere mit möglichst hohem Ähnlichkeitsmaß zusammengeführt werden, wenn das Paar folgendermaßen ausgewählt wird:

$$\arg\max\{a_{(p,q)} \mid (p \in \mathrm{top}(G_v) \vee p \in \mathrm{bot}(G_v)) \wedge q \in G_w\}$$

Mit mindestens einem der beiden Charaktere (p) am Rand seiner Teilpermutation, ist sichergestellt, dass die ausgewählten Charaktere p, q und ihre respektiven Teilpermutationen mithilfe einer Blockkreuzung zusammengeführt werden können (vergleiche Abbildung 3.3).

3.3. Funktionsweise

Anhand des Pseudocodes in Algorithmus 1 und einem Beispiel soll die Vorgehensweise des Greedy-Algorithmus veranschaulicht werden. Dabei beziehen sich alle nun folgenden Zeilenangaben auf die Zeilen des Pseudocode. Die Storyline $S = (C, M)$, die als Beispiel

Algorithmus 1: GreedySBCM(Liste C, Liste M, int t)

Eingabe: Liste mit Charakteren C, Liste mit Treffen M, Lookahead t
Ausgabe: Liste mit Permutationen P

```
1  erstelle zufällige Startpermutation π der Charaktere
2  erstelle leere Liste von Permutationen P
3  P.append(π)
4  i=1
5  while i < M.length do
6      while isValidMeeting(π, M_i) do
7          if i == M.length then
8              return P
9          i++
10     A = buildSimilarityMatrix(π, M, i, t)
11     while true do
12         G = buildGroups(π, M_i)
13         pair = findPair(π, G, A)
14         π = adaptPermutation(pair, π, G, A)
15         P.append(π)
16         if isValidMeeting(π, M_i) then
17             break
18 return P
```

verwendet wird, sei gegeben durch:

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

Des Weiteren wird ein Lookahead von $t = 3$ verwendet.

Der Algorithmus bildet zuerst eine zufällige Startpermutation π der k Charaktere, eine Liste P, welche die Permutationen speichert, die durchlaufen werden, und fügt die Startpermutation π zu P hinzu (vergleiche Zeile 1–3). Damit das Beispiel übersichtlich bleibt, sei die zufällige Startpermutation gegeben als $\pi = (1, 2, 3, 4, 5, 6)$. In Zeile 4 und 5 wird eine Schleife gestartet, die dafür sorgt, dass alle Treffen aus M abgearbeitet werden.

Zu Beginn jeder Iteration i der Schleife (Zeile 6–9) werden die ersten Treffen aus dem Anfang der Liste M übersprungen, die in der aktuellen Permutation π stattfinden können und i dementsprechend erhöht. Wenn M abgearbeitet ist wurden alle Treffen ermöglicht und es wird P als Lösung ausgegeben. In unserem Beispiel können dementsprechend die Treffen $M_1 = \{2, 3\}$ und $M_2 = \{3, 4, 5\}$ übersprungen werden, da sie in der aktuellen Permutation π möglich sind.

Erreichen wir ein Treffen M_i, welches in der aktuellen Permutation nicht stattfinden kann, wird das Ähnlichkeitsmaß aller notwendiger Charakter-Paare, wie in Abschnitt 3.2.1 beschrieben, berechnet und in der Matrix A gespeichert (Zeile 10). Da M_3

(a) Die zwei Teilpermutation $\pi_{3,1} = (1, 2)$ und $\pi_{3,2} = (4, 5, 6)$ für das Treffen M_3 sind orangefarben gezeichnet.

(b) Das ausgewählte Charakterpaar $(1, 5)$ und ihre respektiven Gruppen können mit einer Blockkreuzung zusammengeführt werden.

Abb. 3.4.: Die zwei Teilpermutationen des Treffens M_3 werden erst veranschaulicht und anschließend mithilfe einer Blockkreuzung zusammengeführt.

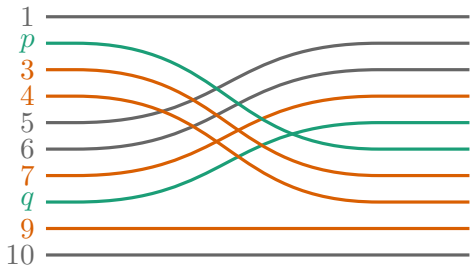
im Moment nicht stattfinden kann, wird jetzt die passende Ähnlichkeitsmatrix berechnet. Bei einem Lookahead von $t = 3$ und nur noch zwei verbleibenden Treffen werden alle noch folgenden Treffen berücksichtigt. Daher wird das Ähnlichkeitsmaß aller Charakterpaare des nächsten Treffens $M_4 = \{1, 5\}$ um eins, und anschließend das Ähnlichkeitsmaß aller Charakterpaare aus $M_5 = \{1, 2, 5\}$ um ½ erhöht. Die entstehende Ähnlichkeitsmatrix ist demnach:

$$A = \begin{pmatrix} 0 & 0{,}5 & 0 & 0 & 1{,}5 & 0 \\ 0{,}5 & 0 & 0 & 0 & 0{,}5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1{,}5 & 0{,}5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

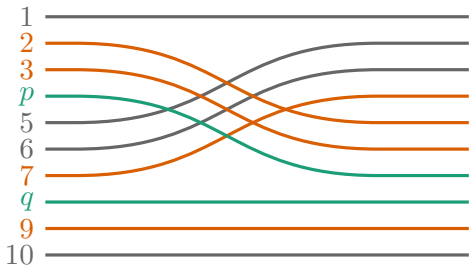
Danach beginnt eine weitere Schleife (Zeile 11), die in jedem Durchlauf eine Blockkreuzung durchführt bis das nächste Treffen M_i möglich wird. Dazu bildet der Algorithmus in Zeile 12 die Teilpermutationen $\pi_{i,1}, \ldots, \pi_{i,g}$, die ausschließlich aus Charakteren bestehen, die in M_i sind, wobei jede Teilpermutation aus einem maximalen zusammenhängenden Intervall von Charakteren aus π besteht, und speichert diese in der Liste G. In Abbildung 3.4a sind die Teilpermutation $\pi_{3,1} = (1, 2)$ und $\pi_{3,2} = (4, 5, 6)$ veranschaulicht, die bei dem Treffen $M_3 = \{1, 2, 4, 5, 6\}$ für die Permutation $\pi = (1, 2, 3, 4, 5, 6)$ entstehen.

Generell sollen zwei solcher Teilpermutationen in jedem Schritt verbunden werden, bis sich alle Charaktere aus M_i in einer Teilpermutation befinden und das Treffen demnach stattfinden kann. Dazu wird anschließend (Zeile 13), wie in Abschnitt 3.2.2 beschrieben, mithilfe der bereits berechneten Ähnlichkeitsmatrix das Charakterpaar (p, q) gewählt, das mit einer Blockkreuzungen zu Nachbarn gemacht werden soll. Für unser Beispiel ist das Ähnlichkeitsmaß für die Charakterpaare $(1, 5)$ und $(5, 1)$ am höchsten. Da sich die beiden Charaktere aus dem Paar $(1, 5)$ in verschiedenen Teilpermutationen ($\pi_{3,1}$ und $\pi_{3,2}$) befinden und p am Rand seiner respektiven Teilpermutation ist ($\text{top}(\pi_{3,1})$), kann das Paar und ihre respektiven Teilpermutationen mithilfe einer Blockkreuzung zusammengeführt werden (siehe Abbildung 3.4b). Das Charakterpaar $(5, 1)$ erfüllt diese Bedingungen nicht.

In Zeile 14 und 15 wird daraufhin die aktuelle Permutation, die sich aufgrund der



(a) Die Teilpermutation mit p wird hinter q geschoben, falls p das erste Element seiner Teilpermutation ist.



(b) Die Teilpermutation mit p wird vor q geschoben, falls p das letzte Element seiner Teilpermutation ist.

Abb. 3.5.: Das Verschieben der Teilpermutation ist abhängig von der Position des Charakters p innerhalb seiner Teilpermutation. Dabei spielt weder die Größe, noch die Position der Teilpermutationen eine Rolle.

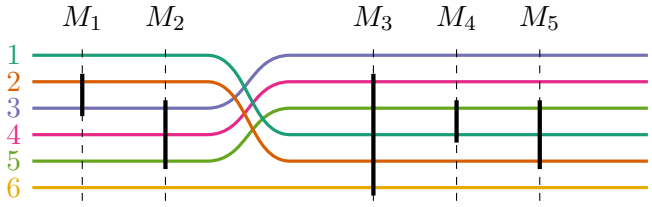


Abb. 3.6.: Für die Storyline aus dem Beispiel erstellt der Greedy-Algorithmus eine Visualisierung mit einer Blockkreuzung.

ausgewählten Blockkreuzung ändert, aktualisiert und zu der Liste der durchlaufenden Permutationen P hinzugefügt. Dazu wird die Teilpermutation, dessen ausgewählter Charakter p am Rand seiner Teilpermutation ist, zu dem anderen Charakter q verschoben. Dabei soll sich p auf der Position nach q befinden, falls er der erste Charakter seiner Teilpermutation ist, und auf der Position vor q befinden, wenn er der letzte Charakter seiner Teilpermutation ist. In Abbildung 3.5 sind diese zwei Fälle veranschaulicht. Die Permutation nach der ausgewählten Blockkreuzung ist in unserem Beispiel demnach $\pi = (3, 4, 5, 1, 2, 6)$, die anschließend zu P hinzugefügt wird.

Zum Schluss (Zeile 16-17) wird die innere Schleife unterbrochen, falls das Treffen M_i jetzt möglich ist. Das Treffen M_3 kann in diesem Fall jetzt stattfinden, die zweite innere Schleife wird demnach abgebrochen und mithilfe der äußeren Schleife werden die nächsten zwei Treffen M_4 und M_5 abgearbeitet, die beide ohne weitere Blockkreuzungen stattfinden können.

Sind beide Schleifen durchlaufen ist der Lösungsweg in P gespeichert und kann zurückgegeben werden. Die gefundene Lösung des Beispiels ist demnach:

$$P = ((1, 2, 3, 4, 5, 6), (3, 4, 5, 1, 2, 6))$$

Die dazu passende Storyline-Visualisierung ist in Abbildung 3.6 zu sehen.

3.4. Laufzeit

Die kleinen Vorbereitungen des Algorithmus benötigen $O(k)$ Zeit. Anschließend wird jedes Treffen in der äußeren Schleife abgearbeitet, das heißt sie wird für m Treffen genau m mal ausgeführt. Für jedes dieser Treffen M_i wird dann Folgendes abgearbeitet:

Die erste innere Schleife (Zeile 6) benötigt pro Durchlauf $O(k)$ Zeit, wird allerdings, unabhängig von der äußeren Schleife, insgesamt nur m mal ausgeführt. Anschließend wird in $O(t \cdot k^2)$ Zeit die Ähnlichkeitsmatrix berechnet. Um die Charaktere zusammenzuführen wird danach die zweite innere Schleife (Zeile 11) genau $g - 1 < k$ mal ausgeführt. Innerhalb dieser Schleife wird in $O(k)$ Zeit die Liste der Teilpermutationen G erstellt. Außerdem wird dort in $O(k^2)$ das Paar gefunden, das mit der aktuellen Blockkreuzung verbunden werden soll, und in $O(k)$ Zeit die Permutation π aktualisiert. Danach wird in konstanter Zeit die aktualisierte Permutation π zur Liste P hinzugefügt. Zum Schluss wird in $O(k)$ überprüft, ob das Treffen jetzt stattfinden kann und die zweite innere Schleife möglicherweise abgebrochen.

Nachdem diese beiden Schleifen durchlaufen wurden, sind alle Treffen abgearbeitet und es ergibt sich eine Laufzeit von:

$$O(k + m \cdot k + m \cdot (t \cdot k^2 + k \cdot (k + k^2 + k + k))) = O(m \cdot (tk^2 + k^3)) \text{ mit } t \leq m$$

Wenn der Parameter t in Abhängigkeit von k gewählt wird, kann die Laufzeit weiter zu $O(mk^3)$ vereinfacht werden.

3.5. Erweiterungen

Die Visualisierungen, die der Algorithmus aus Abschnitt 3.3 berechnet, sind teilweise noch etwas unübersichtlich. Deshalb soll das Design der Visualisierung in diesem Abschnitt weiterhin verbessert werden. Um dies zu erreichen soll auch die Anzahl der paarweisen Kreuzungen beachtet werden. Des Weiteren werden verschiedene Methoden vorgestellt um möglicherweise eine bessere Startpermutation zu wählen.

3.5.1. Priorisierung kleinerer Blockkreuzungen

Bisher wurde ausschließlich die Anzahl der Blockkreuzungen beachtet. Das soll auch weiterhin im Vordergrund stehen. Allerdings kann auch auf die Anzahl der paarweisen Kreuzungen, die durch die Blockkreuzungen entstehen, eingegangen werden. Bei der Auswahl des Charakterpaares, welches zusammengeführt werden soll, kann abgesehen von dem Ähnlichkeitsmaß zusätzlich auch die Größe der Blockkreuzungen berücksichtigt werden. Demnach wird bei jeder Auswahl des Charakterpaares (p, q) nach der Anzahl der entstehenden paarweisen Kreuzungen ausgewählt, falls das Ähnlichkeitsmaß der Paare gleich ist. Das Ähnlichkeitsmaß hat jedoch weiterhin Priorität, da es möglicherweise die Anzahl der Blockkreuzungen verringert.

3.5.2. Auswahl der Startpermutation

Anstatt die Startpermutation zufällig zu wählen, können andere Methoden verwendet werden um zu entscheiden, welche Charaktere zu Beginn benachbart sind. Dadurch können möglicherweise unnötige Blockkreuzungen vermieden werden. Besonders der Beginn der Visualisierung kann so unter Umständen übersichtlicher werden. Um die verschiedenen Varianten später referenzieren zu können werden sie hier auch gleich benannt. Die ursprüngliche Version, welche die Startpermutation zufällig wählt, sei durch die Bezeichnung GreedyStartRandom gegeben.

Heuristik basierend auf dem Ähnlichkeitsmaß (GreedyStartHeuristic)

Damit das erste Treffen ohne notwendige Kreuzungen möglich ist, werden die Charaktere, die an ihm teilnehmen zu einem leeren Tupel π_0 hinzugefügt. Anschließend wird die Ähnlichkeitsmatrix wie in Abschnitt 3.2.1 berechnet. In jedem nun folgenden Schritt wird mithilfe der Ähnlichkeitsmatrix ein verbleibender Charakter $c \notin \pi_0$ ausgewählt und an π_0 angehängt, bis alle Charaktere aus C in π_0 sind. Welcher Charakter an welche Seite von π_0 hinzugefügt wird, wird folgendermaßen entschieden:

$$\arg\max\{a_{(c,r)} \mid (r \in \text{top}(\pi_0) \vee r \in \text{bot}(\pi_0)) \wedge c \notin \pi_0\}$$

Demnach wird der Charakter ausgewählt, der der Ähnlichkeitsmatrix nach am besten an den Rand der aktuellen Startpermutation π_0 passt, und dort angehängt.

Wähle das beste Ergebnis aus mehreren Durchführungen des Greedy-Algorithmus mit jeweils zufälliger Startpermutation (GreedyRepetitionRandom)

Anstatt den Greedy-Algorithmus nur einmal mit einer zufälligen Startpermutation durchzuführen, wird er in dieser Variante mehrmals durchlaufen und das Ergebnis mit den wenigsten Blockkreuzungen zurückgegeben. Selbst bei relativ wenigen Wiederholungen kann die Anzahl der Blockkreuzung auf diese Weise oftmals deutlich verringert werden. In den Auswertungen dieser Arbeit wird der Greedy-Algorithmus in dieser Variante zehn mal durchgeführt.

Ausprobieren aller möglichen Startpermutationen (GreedyRepetitionAll)

Falls die Anzahl k der zu betrachtenden Charaktere nicht zu groß ist, ergibt sich die Möglichkeit alle möglichen Startpermutationen auszuprobieren, und das beste Ergebnis auszugeben. Die Lösung wird dann anhand der Anzahl der Blockkreuzungen und paarweisen Kreuzungen ausgewählt, wobei die Verringerung der Blockkreuzung weiter Vorrang hat. Da es $k!$ mögliche Startpermutationen gibt, eignet sich dieses Vorgehen allerdings nur für kleine Eingaben.

4. Implementierung

Es soll eine Webapplikation entwickelt werden, welche für eine Eingabe von Autoren automatisiert eine Storyline-Visualisierung mit möglichst kleiner Anzahl an Blockkreuzungen erstellt. Dabei soll dargestellt werden, wie die Autoren im Laufe der Zeit miteinander kollaboriert haben. Die dafür benötigten Informationen können unter anderem im XML- und JSON-Format aus der Publikationsdatenbank dblp abgefragt werden. Nach dem Aufbereiten der notwendigen Daten berechnet der Greedy-Algorithmus dann die passende Storyline-Visualisierung und das Ergebnis wird anschließend auf der Webseite dargestellt. Außerdem werden weitere Informationen zur Visualisierung und den einzelnen Veröffentlichungen angezeigt. So ist neben der Berechnungszeit auch die Anzahl der Autoren, Veröffentlichungen, Blockkreuzungen und paarweisen Kreuzungen zu sehen. Des Weiteren sind die Veröffentlichungen in der Legende unter der Visualisierung genauer beschrieben. In der Abbildung 4.1 wird gezeigt, wie die Webapplikation eine Storyline-Visualisierung für eine Eingabe von Autoren darstellt.

Für das Zeichnen der Visualisierung eignet sich die JavaScript Bibliothek `D3.js`[1]. Da JavaScript auch gut mit den JSON-Dateien umgehen kann und leicht auf den Servern der Universität umzusetzen ist, wurde der Greedy-Algorithmus in JavaScript implementiert. Das Abfragen der Informationen aus der Datenbank wird mithilfe der `Fetch API`[2] durchgeführt. Für den Aufbau der Webseite wird wie üblich HTML und CSS verwendet, deren Elemente mit JavaScript gegebenenfalls bearbeitet werden. Der Ablauf des Verfahrens ist in Abbildung 4.2 verdeutlicht.

[1] https://d3js.org/
[2] https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

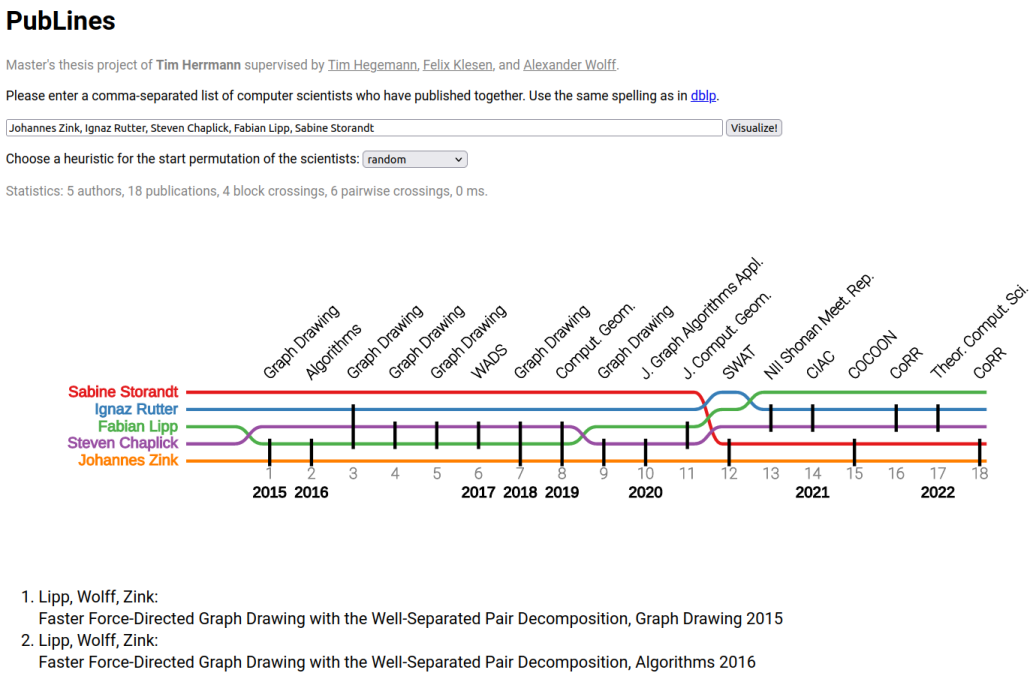


Abb. 4.1.: Die Webapplikation zeigt für eine Eingabe von Autoren eine Storyline-Visualisierung und weitere Informationen an.

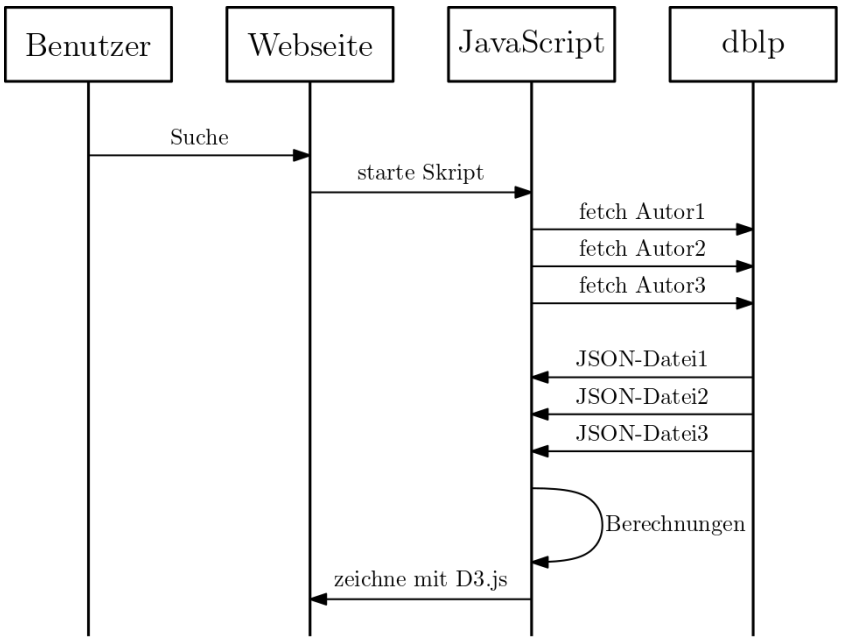


Abb. 4.2.: Das Sequenzdiagramm der Webapplikation zeigt den Ablauf des Greedy-Verfahrens an.

5. Experimentelle Auswertung

Um praxisnahe Ergebnisse zu erhalten, werden zum Auswerten direkte Beispiele mit Autoren von wissenschaftlichen Veröffentlichungen verwendet. Dies hat zur Folge, dass für unterschiedliche Sucheingaben gleicher Länge (gleiche Anzahl an Autoren) verschieden viele Veröffentlichungen existieren. Des Weiteren kann die Anzahl der Veröffentlichung im Verlauf der Zeit ansteigen, sodass direkte Wiederholungen der Versuche schwierig werden können. Genauere Informationen zu den einzelnen Sucheingaben und deren Anzahl an Veröffentlichungen befinden sich deshalb im Anhang A.

Der Algorithmus fragt die notwendigen Informationen von der Publikationsdatenbank dblp ab und bereitet sie vor, bevor das eigentliche Greedy-Verfahren beginnt. Alle folgenden Zeitmessungen verwenden die Web-API `performance.now()`[1], und erfassen ausschließlich die Zeit, die das Greedy-Verfahren benötigt. Die Zeit für das Sammeln, Vorbereiten und Anzeigen der Daten ist nicht miteinbezogen. Die Ergebnisse dieser Zeitmessung sind allerdings trotzdem nicht perfekt akkurat, da die verwendete API die Ergebnisse zu einem gewissen Grad rundet, um potentielle Sicherheitsgefahren abzuschwächen. Sie sollten aber ausreichend genau sein. Des Weiteren zählt der Algorithmus in konstanter Zeit die Anzahl der Blockkreuzungen mit, da sie für die Auswertung von Bedeutung sind. Alle Evaluationen wurden auf einem Computer mit Windows 10 Home 64-Bit, Intel(R) Core(TM) i7-4790 CPU, 12GB Arbeitsspeicher und einer NVIDIA GeForce GTX 970 ausgeführt. Bei dem verwendeten Browser handelt es sich um Google Chrome (Version 101.0.4951.54).

Nachdem alle Storyline-Visualisierungen der Auswertung die Kollaborationen von Autoren aufzeigen, werden in diesem Kapitel nicht mehr allgemein von Charakteren und Treffen gesprochen, sondern von Autoren und Veröffentlichungen. Im Folgenden wird die Auswirkung des Lookaheads t untersucht und die verschiedenen Varianten des Greedy-Algorithmus (vergleiche Abschnitt 3.5.2) miteinander und der optimalen Lösung verglichen.

5.1. Auswirkungen des Lookaheads

Der Greedy-Algorithmus entscheidet anhand einer Ähnlichkeitsmatrix, wie die benötigten Blockkreuzungen gewählt werden. Diese Matrix wird abhängig von den nächsten t Veröffentlichungen gebildet. Der Lookahead t kann in $O(k)$ gewählt werden, um die Laufzeit nicht signifikant zu erhöhen (vergleiche Abschnitt 3.4). Da es für eine steigende Anzahl von Autoren wahrscheinlich auch mehr Veröffentlichungen gibt, die publiziert wurden bevor ein Autor erneut an einer wissenschaftlichen Arbeit beteiligt ist, soll auch

[1] https://developer.mozilla.org/en-US/docs/Web/API/Performance/now

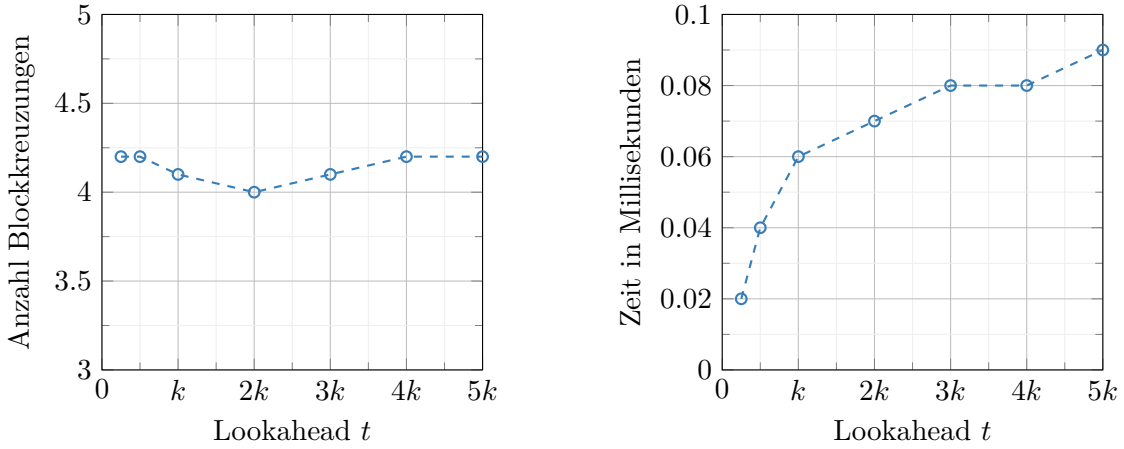


Abb. 5.1.: Für die Sucheingabe Q1.1 mit sechs Autoren und 44 Veröffentlichungen entstehen für $t = 2k$ am wenigsten Blockkreuzungen. Die Abweichungen für verschiedene t sind für diese Sucheingabe allerdings klein.

der Lookahead mit der Anzahl der Autoren steigen. Um einen guten Wert für t zu finden wurden sechs unterschiedlich große Sucheingaben mehrfach getestet. Jedes Beispiel wurde 1000 Mal mit zufälliger Startpermutation durchgeführt und anschließend wurde der Mittelwert der Berechnungsdauer und der Anzahl der Blockkreuzungen gebildet.

Die Ergebnisse sind in Abbildungen 5.1–5.6 visualisiert. Die Berechnungsdauer steigt mit wachsendem t wie erwartet relativ gleichmäßig, während die Anzahl der Blockkreuzungen häufig ein tief für $t = 1/2k$ (siehe Abbildung 5.2, 5.4 und 5.6) besitzt. Auch für $t = 3k$, $t = 4k$ und $t = 5k$ ist die Anzahl der Blockkreuzungen teilweise klein (siehe Abbildung 5.3 und 5.5). Da die Versuche mit einem Lookahead von $1/2k$ in den meisten Fällen gut abschneiden und schneller sind als die anderen drei Optionen, wählen wir $t = 1/2k$. Damit für kleinere Eingaben trotzdem noch ein ausreichender Lookahead verwendet wird, sei $t = \max(6, 1/2k)$. Auf diese Art und Weise werden immer mindestens die nächsten sechs Veröffentlichungen betrachtet, sofern sie existieren.

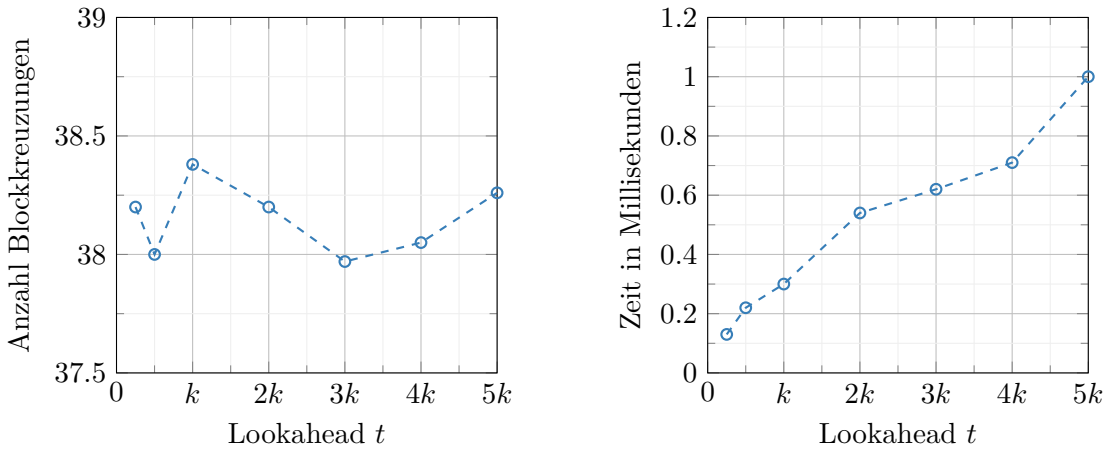


Abb. 5.2.: Für die Sucheingabe Q1.2 mit acht Autoren und 86 Veröffentlichungen entstehen für $t = 1/2k$ und $t = 3k$ am wenigsten Blockkreuzungen.

5.2. Auswertung der verschiedenen Varianten des Greedy-Algorithmus

Um die verschiedenen Varianten des Greedy-Algorithmus zu testen, wurden vier verschiedene Beispiele mit den unterschiedlichen Versionen des Greedy-Algorithmus (vergleiche Abschnitt 3.5.2) durchgeführt. Dabei wurden die Ergebnisse der Varianten, die im Abschnitt 3.5.2 vorgestellt wurden (GreedyStartRandom, GreedyStartHeuristic, GreedyRepetitionRandom und GreedyRepetitionAll) miteinander verglichen. Für jede dieser vier Varianten wird jede der Sucheingaben zuerst mit drei Autoren durchgeführt, und nach und nach werden weitere Autoren hinzugefügt und k somit erhöht, um das Verhalten über unterschiedlich große Eingaben zu testen. Da Visualisierungen

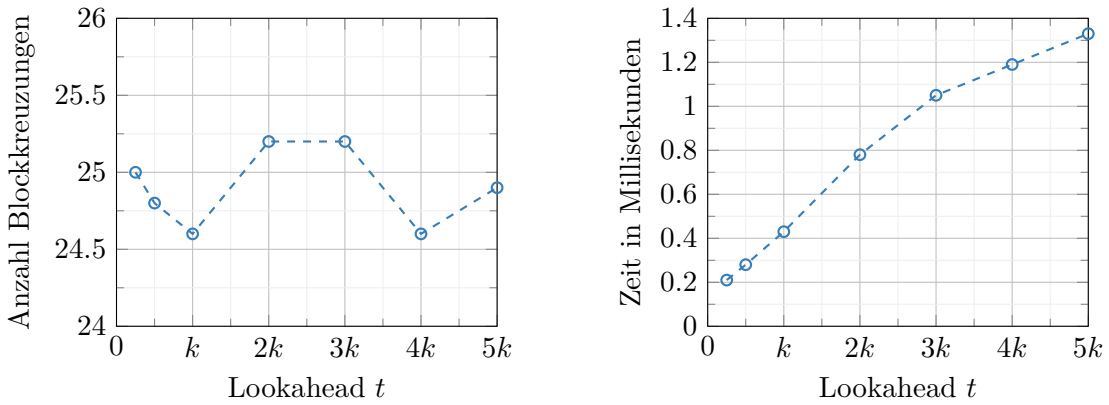


Abb. 5.3.: Für die Sucheingabe Q1.3 mit 13 Autoren und 101 Veröffentlichungen entstehen für $t = k$ und $t = 4k$ am wenigsten Blockkreuzungen. Die Anzahl an Blockkreuzungen für $t = 1/2k$ und $t = 5k$ sind aber nicht sehr viel höher.

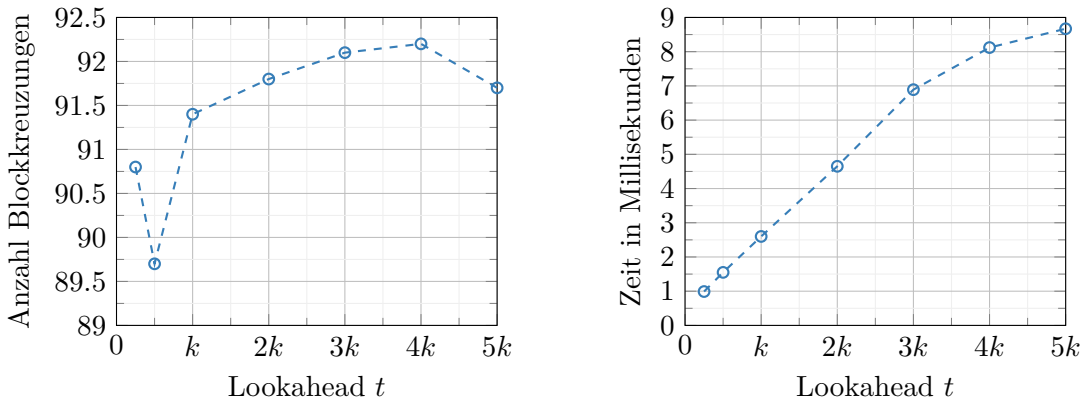


Abb. 5.4.: Die Auswertung des Lookaheads t anhand der Sucheingabe Q1.4 mit 19 Autoren und 192 Veröffentlichungen zeigt, dass die Anzahl der Blockkreuzungen für $t = 1/2k$ am kleinsten ist.

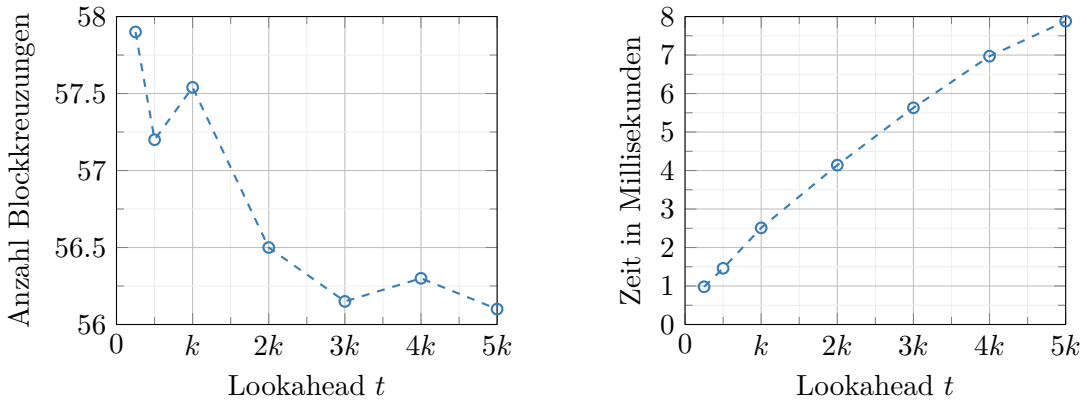


Abb. 5.5.: Für die Sucheingabe Q1.5 mit 23 Autoren und 207 Veröffentlichungen schneiden die größeren Lookaheads ($3k$–$5k$) besser ab.

mit zwei oder weniger Autoren keine Kreuzungen benötigen, sind sie für diese Arbeit uninteressant. Die meisten Ergebnisse werden wie zuvor über 1000 Durchläufe gemittelt. Nur das Durchführen des Greedy-Verfahrens, welche alle möglichen Startpermutationen ausprobiert, wurde aufgrund der hohen Laufzeit für Sucheingaben mit acht oder mehr Autoren nicht mehr wiederholt getestet.

Die Ergebnisse dieser Versuche sind in Abbildungen 5.7–5.10 veranschaulicht. Sie zeigen, dass es für Eingaben der Länge fünf oder weniger, das heißt die Anzahl der zu betrachtenden Autoren beträgt höchstens fünf, am besten ist GreedyRepetitionAll zu verwenden, und demnach einfach alle möglichen Startpermutationen durchzutesten. Dieses Verfahren benötigt zwar ein vielfaches länger als die anderen, liefert aber das beste Ergebnis und braucht trotzdem noch nicht zu viel Zeit. Kleinere Sucheingaben mit fünf Autoren und 57 Veröffentlichungen benötigen ungefähr acht Millisekunden, während Beispiele mit fünf Autoren und 123 Veröffentlichungen 20 Millisekunden Berechnungs-

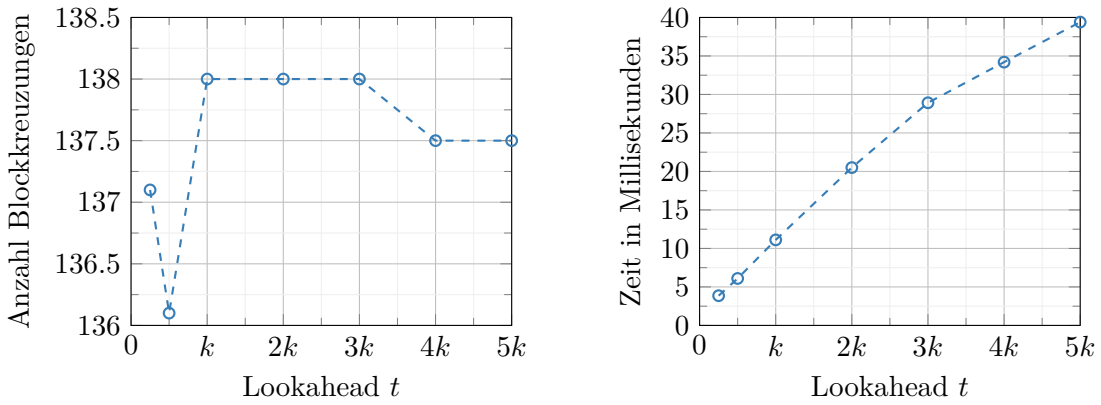


Abb. 5.6.: Die Auswertung des Lookaheads t anhand der Sucheingabe Q1.6 mit 34 Autoren und 345 Veröffentlichungen zeigt, dass die Anzahl der Blockkreuzungen für $t = 1/2k$ am kleinsten ist.

Tab. 5.1.: Berechnungsdauer von GreedyRepetitionAll in Sekunden für verschiedene Eingaben mit k Autoren und m Veröffentlichungen.

k	2	3	4	5	6	7	8	9	10
m	10	22	108	123	135	136	136	140	189
Zeit	<0,0001	0,0001	0,0022	0,021	0,2341	2,1193	20,3	247	4435

zeit brauchen. Eingaben bis zur Länge sieben sind auch kein Problem, allerdings sind da kleine Wartezeiten bemerkbar. Die meisten Beispiele wurden allerdings unter drei Sekunden berechnet. Aufgrund der faktoriellen Komplexität sind für mittelgroße und größere Eingaben allerdings andere Verfahren zu empfehlen (siehe Tabelle 5.1).

Die Variante GreedyStartHeuristic bildet die Startpermutation mithilfe der Ähnlichkeitsmatrix und erweist sich in ein paar Fällen als einfache Möglichkeit die Anzahl der Blockkreuzungen gegenüber GreedyStartRandom ein wenig zu vermindern (siehe Abbildung 5.10). Besonders für kleinere Eingaben mit nicht zu vielen Veröffentlichungen ist öfter eine kleine Verbesserung erkennbar (siehe Abbildung 5.8). Auch für große Eingaben ist dieses Vorgehen teilweise GreedyStartRandom vorzuziehen. Allerdings hat die Startpermutation dann weniger Auswirkungen auf die Anzahl der Blockkreuzungen insgesamt, da ein Großteil dieser zu weit von der Startpermutation entfernt entstehen. Deshalb kann diese Methode, je nach Beispiel, auch schlechter abschneiden und ist daher generell nicht zuverlässig (vergleiche Abbildung 5.9). Sowohl GreedyStartHeuristic, als auch GreedyStartRandom finden sehr schnell eine Visualisierung für die Storyline, erzeugen dabei aber die meisten Blockkreuzungen.

Im Vergleich dazu findet GreedyRepetitionRandom Lösungen, die nahezu immer weniger Blockkreuzungen enthalten. Dafür ist die Berechnungsdauer zwar auch deutlich größer, sie steigt aber im Gegensatz zu GreedyRepetitionAll, nicht faktoriell an. Deshalb ist diese Variante für die meisten größeren Eingaben die beste.

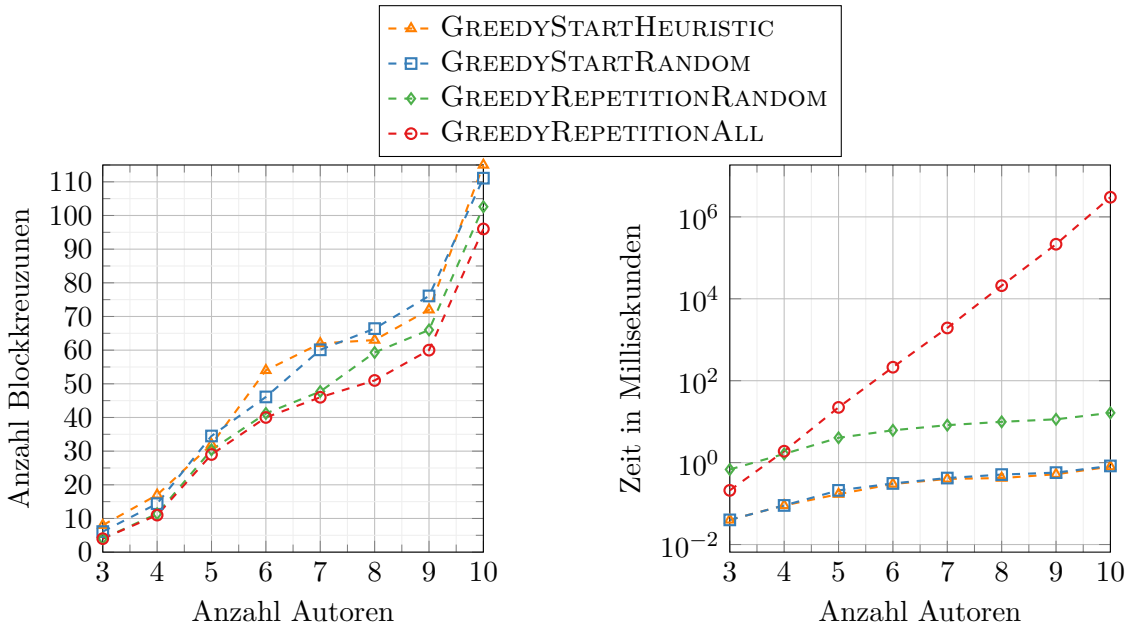


Abb. 5.7.: Verhalten der unterschiedlichen Varianten für die Sucheingaben aus Q2.1. Die Zeitachse ist logarithmisch.

Falls eine spezielle Anordnung der Autoren bevorzugt ist oder gewisse Tendenzen der Kollaborationen bereits bekannt sind, ist es aber möglicherweise sinnvoll dem Greedy-Algorithmus eine erwünschte Startpermutation vorzugeben.

5.3. Vergleich des Greedy-Algorithmus mit der optimalen Lösung

Ähnlich wie in Abschnitt 5.2 beginnen die Eingaben hier mit drei Autoren und es werden nach und nach weitere Autoren hinzugefügt. Die optimalen Lösungen wurden mit dem FTP-Algorithmus von van Dijk et al. [vDFF+17] berechnet und anschließend mit den Ergebnissen des Greedy-Algorithmus verglichen. Da dieser allerdings in C++ programmiert ist können die Zeitmessungen nicht direkt mit denen des Greedy-Algorithmus verglichen werden. Um den generellen Zeitverlauf für die Berechnung von genauen Lösungen aufzuzeigen, wurden die Zeitmessung trotzdem in die Ergebnisse der Evaluation mit aufgenommen. Für die Startpermutation des Greedy-Verfahrens wurden wieder die Varianten aus Abschnitt 3.5.2 verwendet, wobei die Anzahl der Blockkreuzungen der zwei zufälligen Vorgehen wieder über 1000 Versuche gemittelt wurden.

Die Ergebnisse von acht verschiedenen Sucheingaben sind in Abbildungen 5.11–5.18 abgebildet und zeigen, dass die Varianten des Greedy-Algorithmus, die am schnellsten eine Visualisierung berechnen (GreedyStartRandom und GreedyStartHeuristic), oftmals viel mehr Blockkreuzungen erzeugen als notwendig (siehe Abbildung 5.16). Wenn GreedyRepetitionAll verwendet wird können die Ergebnisse häufig erstaunlich gut

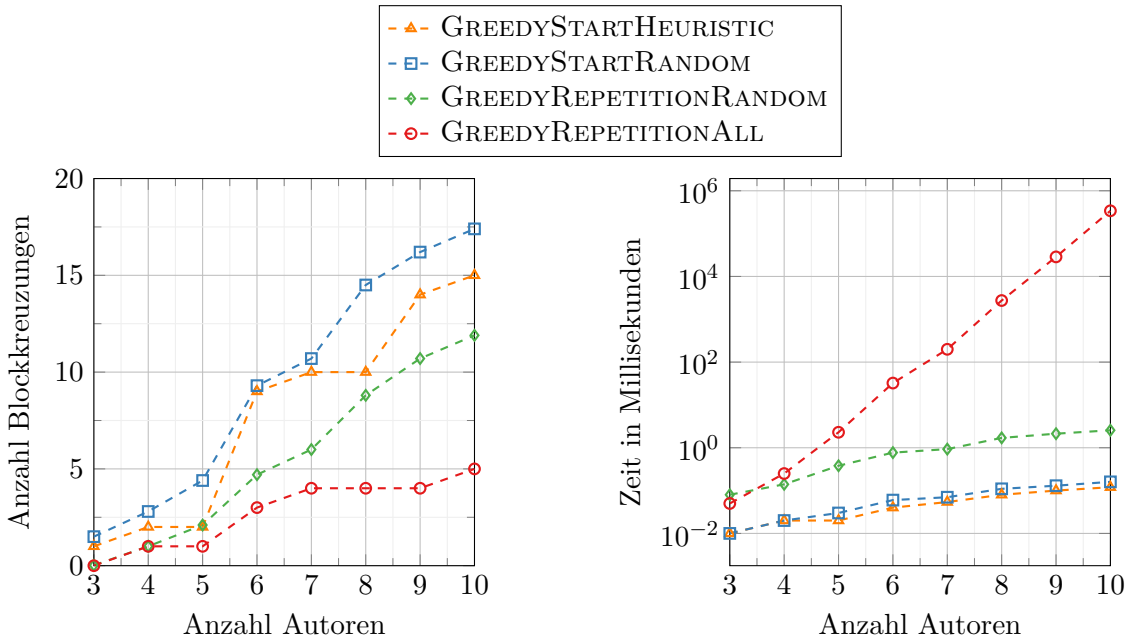


Abb. 5.8.: Verhalten der unterschiedlichen Varianten für die Sucheingaben aus Q2.2. Die Zeitachse ist logarithmisch.

mit der optimalen Lösung mithalten (siehe Abbildung 5.13 und 5.14). Auch für Eingaben, in denen diese Variante schlechter abschneidet, findet sie meistens trotzdem ein Storyline-Visualisierung, die weniger als doppelt so viele Blockkreuzungen enthält als die optimale Lösung (siehe Abbildungen 5.15 und 5.16). Da er allerdings auch faktorielle Laufzeit besitzt, ist er gegenüber exakten Berechnungen nicht vorzuziehen. Demnach sind für Storylines mit wenigen Charakteren meistens andere Ansätze, wie beispielsweise parametrisierte Verfahren, vorzuziehen. Unter den Varianten, die sich auch noch für Storylines mit vielen Autoren eignen, schneidet GreedyRepetitionRandom am besten ab. Schon bei nur zehn Wiederholungen findet dieses Verfahren Lösungen, die nicht sehr viel mehr Blockkreuzungen enthalten als die Ergebnisse von GreedyRepetitionAll (siehe Abbildungen 5.11 und 5.16). Deshalb eignet es sich für Anwendungen, die für Eingaben mit vielen Charakteren eine schnelle Lösung benötigen, generell am besten.

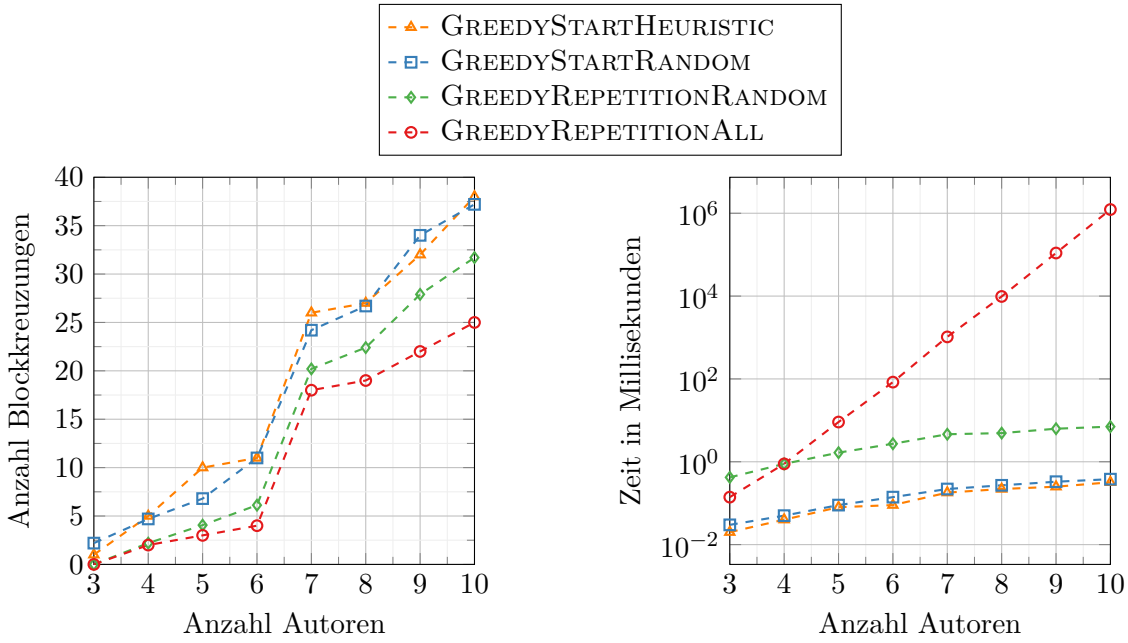


Abb. 5.9.: Verhalten der unterschiedlichen Varianten für die Sucheingaben aus Q2.3. Die Zeitachse ist logarithmisch.

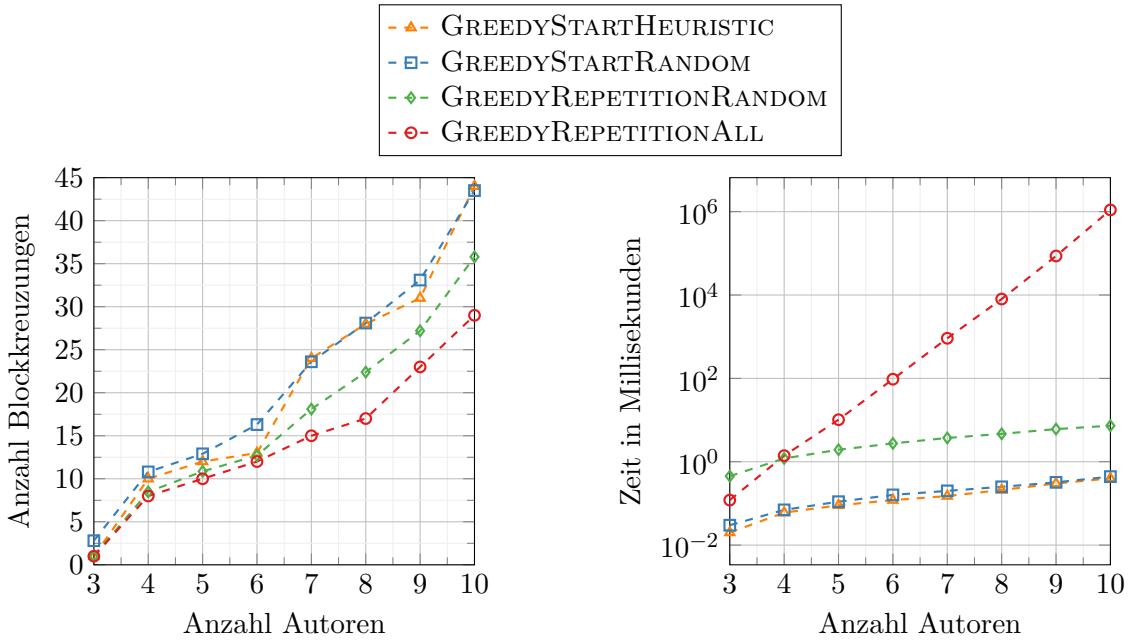


Abb. 5.10.: Verhalten der unterschiedlichen Varianten für die Sucheingaben aus Q2.4. Die Zeitachse ist logarithmisch.

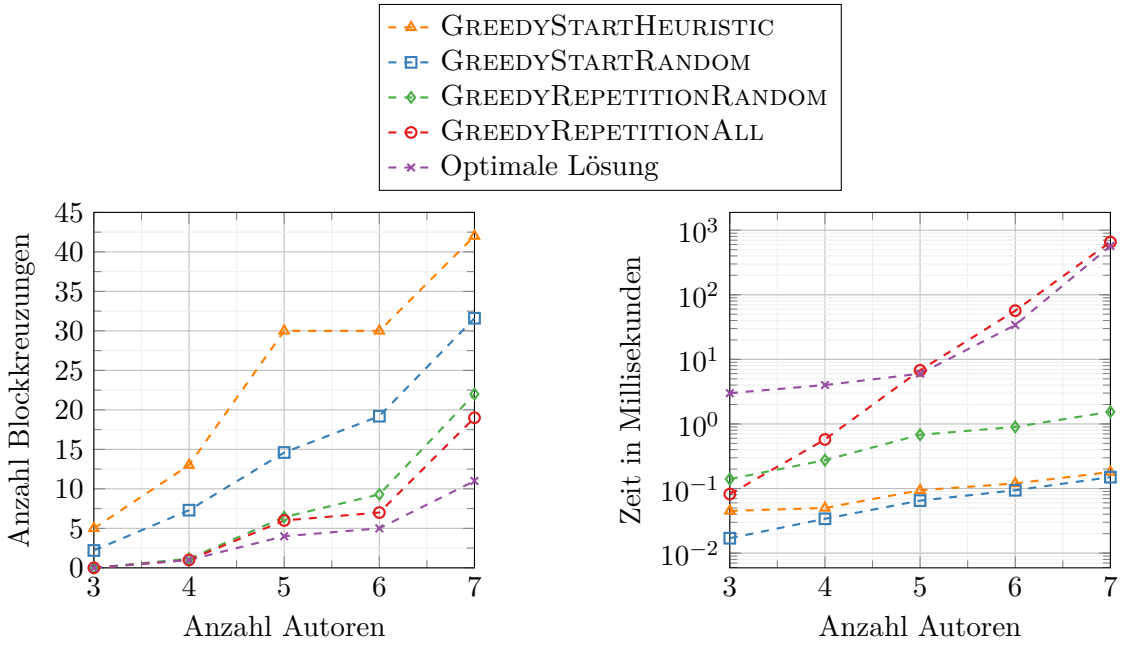


Abb. 5.11.: Vergleich der verschiedenen Greedy-Varianten mit der optimalen Lösung anhand der Sucheingaben aus Q3.1. Die Zeitachse ist logarithmisch.

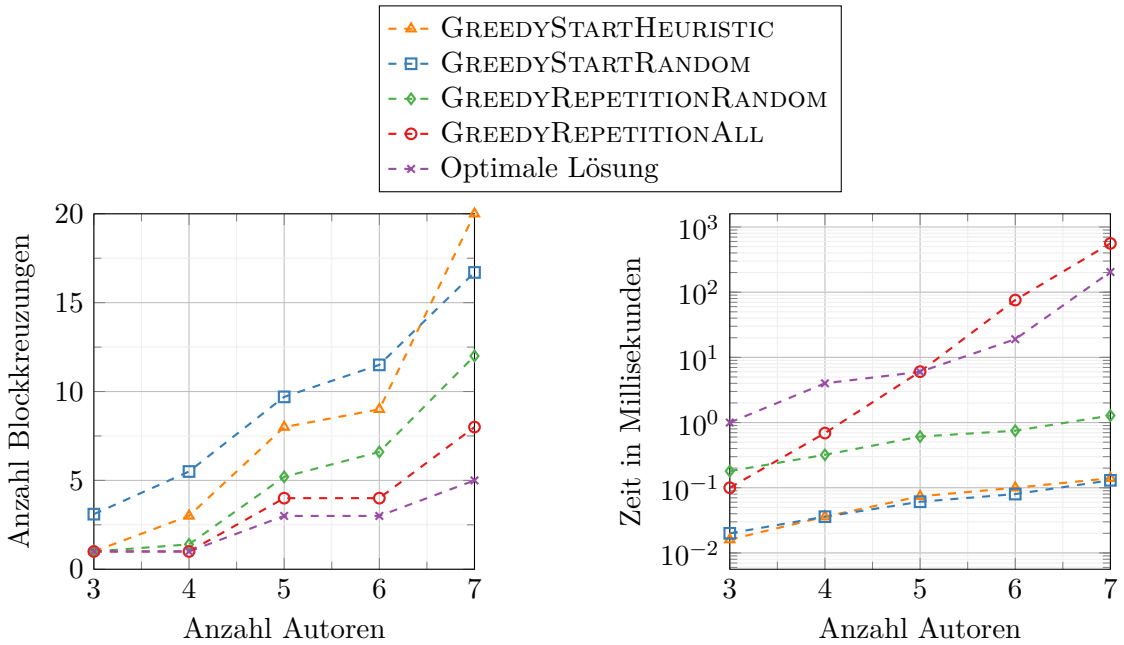


Abb. 5.12.: Vergleich der verschiedenen Greedy-Varianten mit der optimalen Lösung anhand der Sucheingaben aus Q3.2. Die Zeitachse ist logarithmisch.

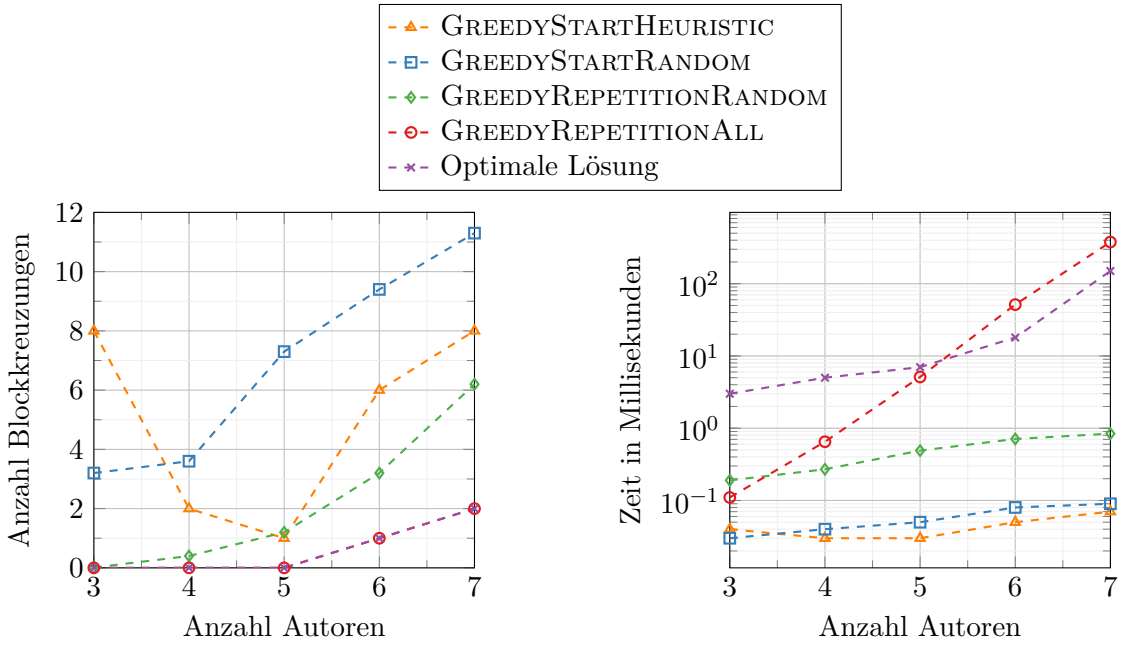


Abb. 5.13.: Vergleich der verschiedenen Greedy-Varianten mit der optimalen Lösung anhand der Sucheingaben aus Q3.3. Die Zeitachse ist logarithmisch.

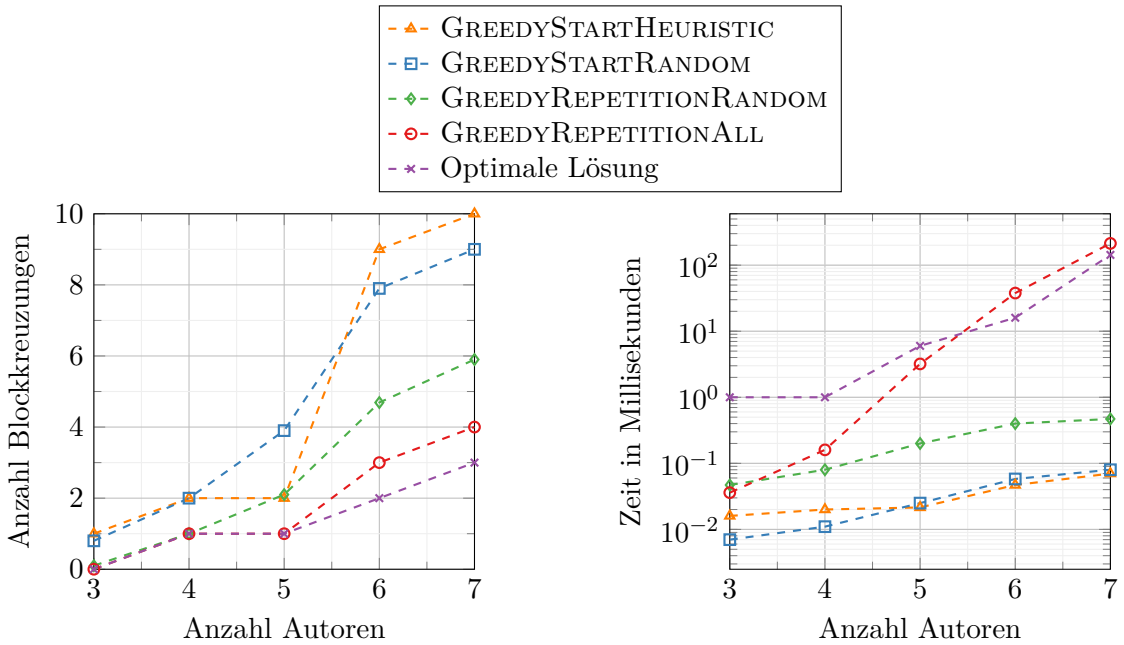


Abb. 5.14.: Vergleich der verschiedenen Greedy-Varianten mit der optimalen Lösung anhand der Sucheingaben aus Q3.4. Die Zeitachse ist logarithmisch.

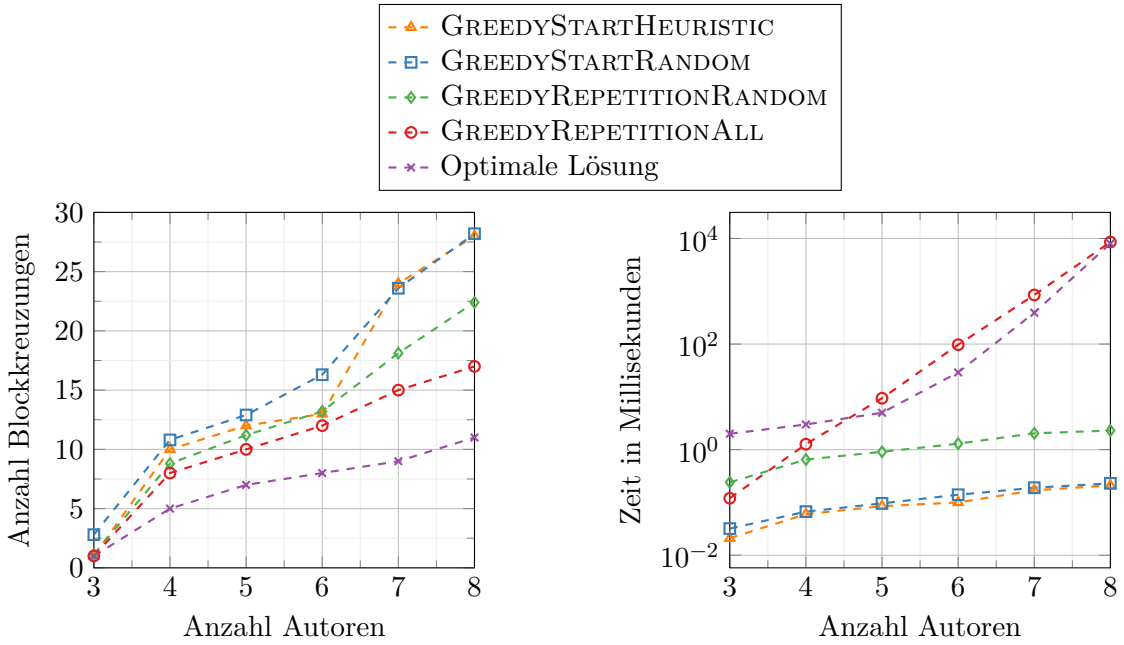


Abb. 5.15.: Vergleich der verschiedenen Greedy-Varianten mit der optimalen Lösung anhand der Sucheingaben aus Q3.5. Die Zeitachse ist logarithmisch.

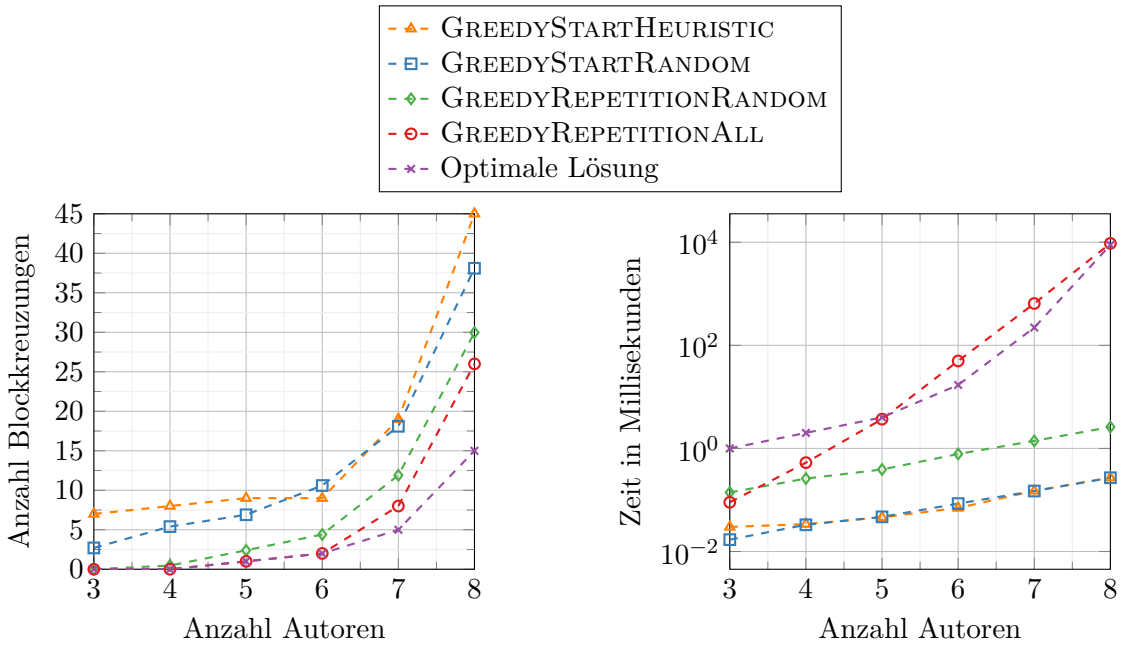


Abb. 5.16.: Vergleich der verschiedenen Greedy-Varianten mit der optimalen Lösung anhand der Sucheingaben aus Q3.6. Die Zeitachse ist logarithmisch.

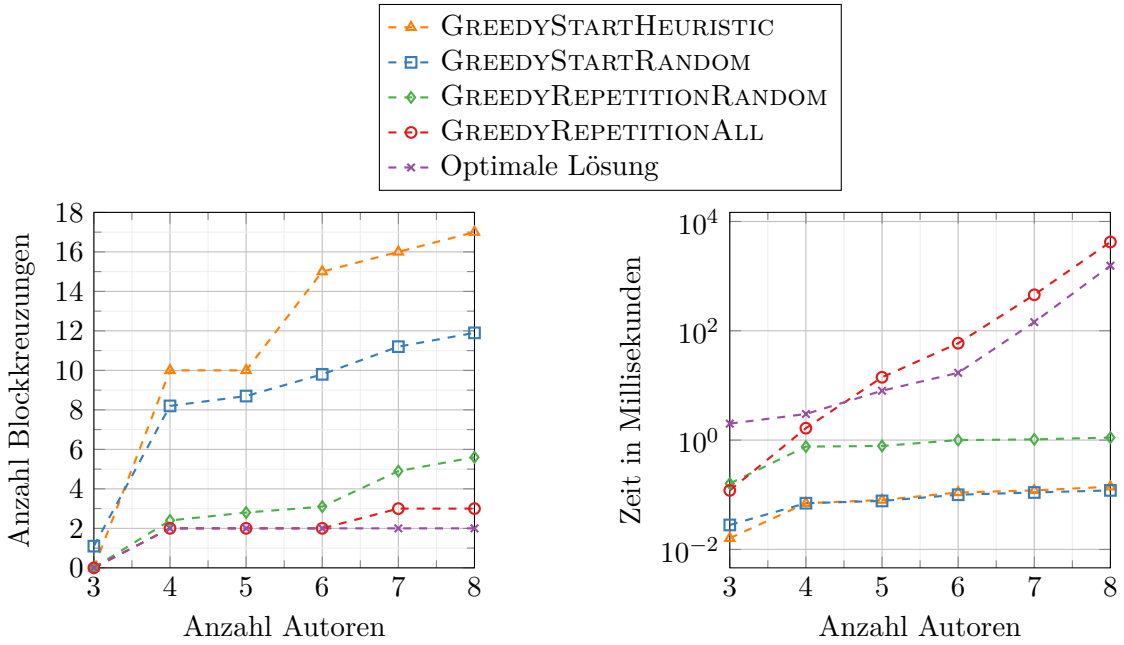


Abb. 5.17.: Vergleich der verschiedenen Greedy-Varianten mit der optimalen Lösung anhand der Sucheingaben aus Q3.7. Die Zeitachse ist logarithmisch.

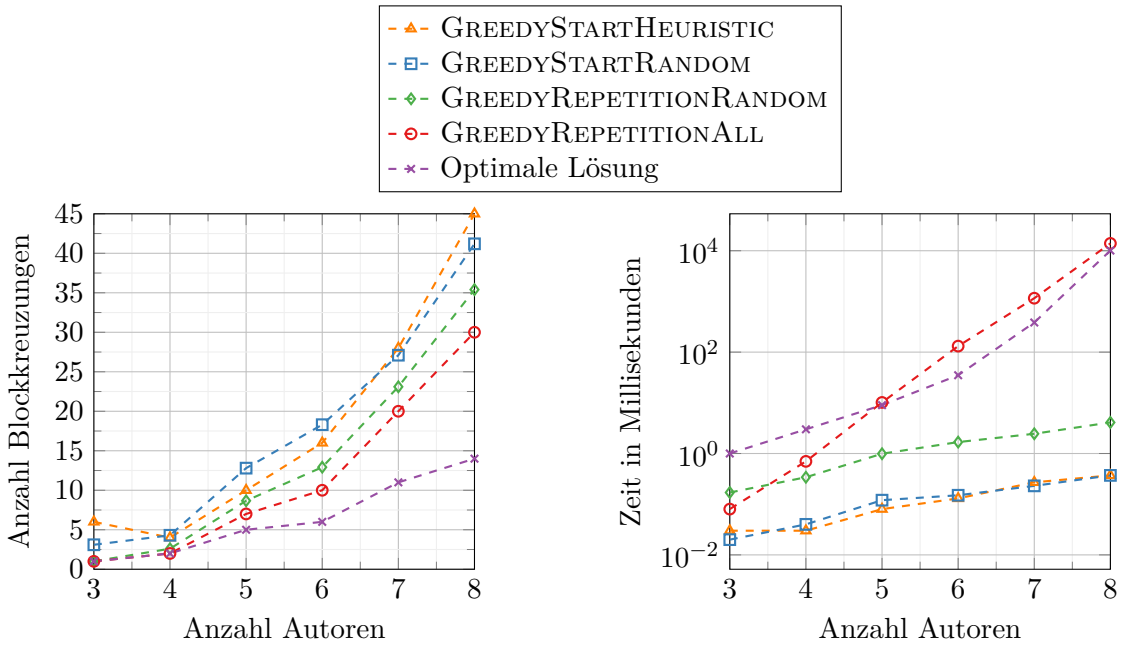


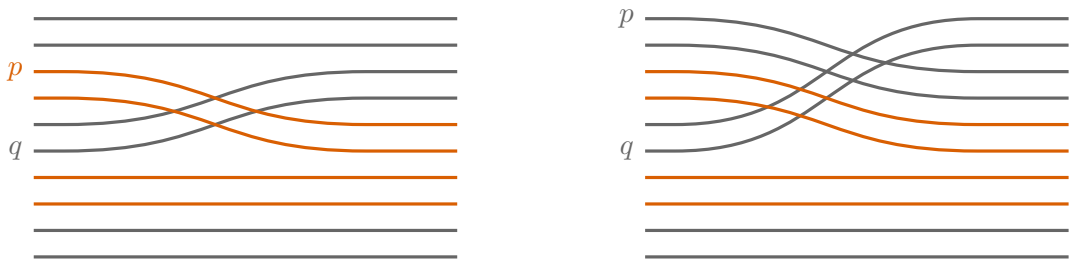
Abb. 5.18.: Vergleich der verschiedenen Greedy-Varianten mit der optimalen Lösung anhand der Sucheingaben aus Q3.8. Die Zeitachse ist logarithmisch.

6. Fazit

Wir haben einen heuristischen Greedy-Algorithmus konstruiert, der mithilfe einer Ähnlichkeitsmatrix schnell Storyline-Visualisierungen berechnet. Des Weiteren wurde eine Webseite erstellt, die zu einer Eingabe von Wissenschaftlern Daten aus der Datenbank dblp abfragt, mithilfe des Greedy-Verfahrens eine passende Storyline-Visualisierung berechnet und diese mit D3.js anzeigt. Dabei kann die Startpermutation manuell gewählt oder randomisiert werden. Außerdem kann auch über die Ähnlichkeitsmatrix eine Startpermutation berechnet werden. Diese betrachtet allerdings nur die Veröffentlichungen zu Beginn der Storyline und kann deshalb für längere Visualisierungen schlechter abschneiden als die ursprüngliche randomisierte Version. Wenn das beste Ergebnis aus zehn Durchführungen des Algorithmus mit jeweils zufälliger Startpermutation gewählt wird, entstehen im Vergleich zu den beiden bisherigen Ansätzen deutlich weniger Blockkreuzungen. Die durchschnittliche Berechnungsdauer ist dafür zwar um ein Vielfaches größer, sie wächst im Gegensatz zu den faktoriellen Verfahren aber für größere Eingaben nicht zu stark an und eignet sich daher am besten für die meisten Storylines mit vielen zu beobachtenden Charakteren. Die Variante, die für jede mögliche Startpermutation den Greedy-Algorithmus durchführt und das beste Ergebnis zurückgibt, ist erstaunlicherweise oftmals nicht sehr viel schlechter als die optimale Lösung. Sie hat aber faktorielle Laufzeit und ist deshalb exakten Lösungsansätzen nicht überlegen. Alle anderen hier bisher vorgestellten Ansätze sind aufgrund der Greedy-Heuristik zwar schnell, erstellen aber Visualisierungen mit ungefähr drei bis sechs mal so vielen Blockkreuzungen wie notwendig und sind daher nur dann relevant, wenn die Zeit eine große Rolle spielt oder die Visualisierung eine gewisse Menge an Charakterlinien überschreitet.

Weiterhin wäre es interessant zu untersuchen, ob die Variante, die eine Visualisierung für alle möglichen Startpermutationen berechnet, mithilfe von Pruning beschleunigt werden kann. Dabei könnten eventuell Lösungen gefunden werden, die nicht all zu viel schlechter sind als das Optimum, allerdings deutlich weniger Berechnungszeit benötigen. Auch das wiederholende Ausführen kann weiter untersucht werden. Anstatt den Algorithmus zehn mal mit zufälliger Startpermutation durchzuführen können vielleicht zusätzlich noch verschiedene andere Heuristiken gewählt und getestet werden um ein besseres Ergebnis zu finden.

Außerdem kann die Wahl der Charakterpaare (p, q), welche mit einer Blockkreuzung zu Nachbarn gemacht werden, überarbeitet werden. Für ein ausstehendes Treffen M_i werden dafür im Moment ausschließlich Charaktere ausgewählt, die Teil dieses Treffens sind $(p, q \in M_i)$. Dieses Vorgehen war sehr intuitiv, da auf diese Art und Weise immer genau eine Teilpermutation mit einer Blockkreuzung verschoben wird. Es können allerdings auch einige andere Charakterpaare betrachtet werden, die mithilfe einer Blockkreuzung, welche zwei Teilpermutationen zusammenführt, benachbart werden. Demnach sollen bei



(a) Während die zwei orangefarbenen Teilpermutationen zusammengeführt werden, werden zwei Charaktere p und q zu Nachbarn gemacht. Dabei ist nur einer der beiden Charaktere in einer Teilpermutation.

(b) Während die zwei orangefarbenen Teilpermutationen zusammengeführt werden, werden zwei Charaktere p und q, die beide in keiner Teilpermutation sind, zu Nachbarn gemacht.

Abb. 6.1.: Für ein Treffen M_i können bei der Wahl des Charakterpaars (p, q) auch Charaktere betrachtet werden, die in keiner Teilpermutation $\pi_{i,j}$ für $j = 1, 2, \ldots, g$ sind.

der Wahl der Charakterpaare nicht nur die möglichen Paare mit $p, q \in M_i$ (vergleiche Abschnitt 3.2.2), sondern auch die anderen möglichen Paare berücksichtigt werden (siehe Abbildung 6.1). Dadurch könnten bessere Charakterpaare gewählt werden, sodass die Anzahl der Blockkreuzungen weiter verringert wird.

Literaturverzeichnis

[FN17] Theresa Fröschl und Martin Nöllenburg: Minimizing wiggles in storyline visualizations. *Graph Drawing and Network Visualization*, Seiten 585–587, 2017.

[GJLM16] Martin Gronemann, Michael Jünger, Frauke Liers und Francesco Mambelli: Crossing minimization in storyline visualization. In: *International Symposium on Graph Drawing and Network Visualization*, Seiten 367–381. Springer, 2016.

[KNP+15] Irina Kostitsyna, Martin Nöllenburg, Valentin Polishchuk, André Schulz und Darren Strash: On Minimizing Crossings in Storyline Visualizations. September 2015, ISBN 978-3-319-27260-3, 10.1007/978-3-319-27261-0_16.

[Ley] Michael Ley: dblp: computer science bibliography. `https://dblp.org/`.

[Mun] Randall Munroe: Movie Narrative Charts. `https://xkcd.com/657/`.

[SAHW15] Shejuti Silvia, June Abbas, Sam Huskey und Chris Weaver: Storyline visualization with force directed layout. *IEEE VIS Posters*, 2015.

[TM12] Yuzuru Tanahashi und Kwan Liu Ma: Design considerations for optimizing storyline visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2679–2688, 2012.

[vDFF+17] Thomas C. van Dijk, Martin Fink, Norbert Fischer, Fabian Lipp, Peter Markfelder, Alexander Ravsky, Subhash Suri und Alexander Wolff: Block Crossings in Storyline Visualizations. *J. Graph Alg. Appl.*, 21(5):873–913, 2017, 10.7155/jgaa.00443.

[vDLMW18] Thomas C. van Dijk, Fabian Lipp, Peter Markfelder und Alexander Wolff: Computing Storylines with Few Block Crossings. In: Fabrizio Frati und Kwan Liu Ma (Herausgeber): *Proc. 25th Int. Symp. Graph Drawing & Network Vis. (GD'17)*, Band 10692 der Reihe *LNCS*, Seiten 365–378. Springer, 2018, 10.1007/978-3-319-73915-1_29.

A. Verwendete Sucheingaben zur Evaluation

In der Publikationsdatenbank dblp werden Autoren, die den gleichen Namen haben, mithilfe einer vierstellige ID unterschieden. Diese ID wird, falls notwendig, in den folgenden Tabellen mit angegeben. In der Tabelle A.1 sind die unterschiedlichen Informationen zu den Sucheingaben aufgeführt, die für das Auswerten des Lookaheads t verwendet wurden (Abschnitt 5.1). Die anderen Sucheingaben aus Abschnitt 5.2 und 5.3 sind in Tabelle A.2 zu finden. Da diese für verschiedene Eingabelängen ausgeführt wurden, befinden sich in Tabelle A.3 weitere Informationen zur Anzahl der Veröffentlichungen.

Tab. A.1.: Verwendete Sucheingaben mit k Autoren und m Veröffentlichungen.

Name	k	m	Sucheingabe
Q1.1	6	44	Reiner Kolla, Paul Molitor, Uwe Hinsberger, Armin Runge, Günter Hotz, Frank Wolz
Q1.2	8	86	Johannes Zink, Alexander Wolff 0001, Ignaz Rutter, Sabine Storandt, Florian Thiele, Steven Chaplick, Fabian Lipp, Martin Nöllenburg
Q1.3	13	101	Andreas Nüchter, Dorit Borrmann, Rainer Koch, Jens Pottebaum, Stefan May, Markus Kühn, Philipp Koch, Thomas C. van Dijk, Benedikt Budig, Jan-Henrik Haunert, Benjamin Niedermann, Alexander Ravsky, Oleg Verbitsky
Q1.4	19	192	Andreas Nüchter, Dorit Borrmann, Rainer Koch, Jens Pottebaum, Stefan May, Markus Kühn, Philipp Koch, Therese Friberg, Stephan Prödel, Johannes Zink, Alexander Wolff 0001, Ignaz Rutter, Sabine Storandt, Florian Thiele, Steven Chaplick, Fabian Lipp, Martin Nöllenburg, Thomas C. van Dijk, Stephen G. Kobourov
Q1.5	23	207	Frank Puppe, Joachim Baumeister, Maximilian Ertl, Markus Krug, Martin Atzmüller, Stefan Störk, Christian Reul, Reiner Kolla, Paul Molitor, Uwe Hinsberger, Armin Runge, Steffen Hölldobler, Rüdiger Reischuk, Abraham Bernstein, Andreas Nüchter, Dorit Borrmann, Rainer Koch, Jens Pottebaum, Stefan May, Markus Kühn, Philipp Koch, Therese Friberg, Stephan Prödel
Q1.6	34	345	Johannes Zink, Alexander Wolff 0001, Ignaz Rutter, Sabine Storandt, Florian Thiele, Steven Chaplick, Fabian Lipp, Martin Nöllenburg, Thomas C. van Dijk, Stephen G. Kobourov, Frank Puppe, Joachim Baumeister, Maximilian Ertl, Markus Krug, Martin Atzmüller, Stefan Störk, Christian Reul, Jochen Reutelshoefer, Reiner Kolla, Paul Molitor, Uwe Hinsberger, Armin Runge, Steffen Hölldobler, Rüdiger Reischuk, Abraham Bernstein, Andreas Nüchter, Dorit Borrmann, Rainer Koch, Jens Pottebaum, Stefan May, Markus Kühn, Philipp Koch, Therese Friberg, Stephan Prödel

Tab. A.2.: Weitere verwendete Sucheingaben mit k Autoren und m Veröffentlichungen.

Name	k	m	Sucheingabe
Q2.1	10	164	Alexander Wolff 0001, Philipp Kindermann, Steven Chaplick, Martin Nöllenburg, Stephen G. Kobourov, Joachim Spoerhase, Alexander Ravsky, Fabian Lipp, Chan-Su Shin, Michael Kaufmann 0001
Q2.2	10	45	Reiner Kolla, Paul Molitor, Uwe Hinsberger, Armin Runge, Bernd Becker 0001, Günter Hotz, Frank Wolz, Hans-Georg Osthof, Winfried Nöth, Marcel Baunach
Q2.3	10	101	Andreas Nüchter, Dorit Borrmann, Joachim Hertzberg, Kai Lingemann, Hartmut Surmann, Jan Elseberg, Stefan May, Rainer Koch, Christian Pfitzner, Philipp Koch
Q2.4	10	83	Klaus W. Wagner, Heribert Vollmer, Christian Glaßer, Victor L. Selivanov, Ulrich Hertrampf, Stephen D. Travers, Sven Kosub, Lane A. Hemaspaandra, Ronald V. Book, Eric Allender
Q3.1	7	104	Frank Puppe, Joachim Baumeister, Maximilian Ertl, Markus Krug, Martin Atzmüller, Stefan Störk, Christian Reul
Q3.2	7	47	Thomas C. van Dijk, Benedikt Budig, Jan-Henrik Haunert, Benjamin Niedermann, Alexander Ravsky, Oleg Verbitsky, Martin Fink 0001
Q3.3	7	60	Andreas Nüchter, Dorit Borrmann, Rainer Koch, Jens Pottebaum, Stefan May, Markus Kühn, Philipp Koch
Q3.4	7	39	Reiner Kolla, Paul Molitor, Uwe Hinsberger, Armin Runge, Bernd Becker 0001, Günter Hotz, Frank Wolz
Q3.5	8	74	Klaus W. Wagner, Heribert Vollmer, Christian Glaßer, Victor L. Selivanov, Ulrich Hertrampf, Stephen D. Travers, Sven Kosub, Lane A. Hemaspaandra
Q3.6	8	86	Johannes Zink, Alexander Wolff 0001, Ignaz Rutter, Sabine Storandt, Florian Thiele, Steven Chaplick, Fabian Lipp, Martin Nöllenburg
Q3.7	8	43	Vijay Kumar B. G, Ian D. Reid 0001, Gustavo Carneiro, Thanh-Toan Do, Ioannis Patras, Raja Bala, Varnith Chordia, Shaobo Fang
Q3.8	8	91	Boglárka G.-Tóth, Leocadio G. Casado, José Fernández 0001, Eligius M. T. Hendrix, Inmaculada García, Guillermo Aparicio, Blas Pelegrín, Juana López Redondo

Tab. A.3.: Anzahl der Veröffentlichungen m für die ersten k' Autoren aus den Sucheingaben Q2.1–Q3.8.

Name	k'							
	3	4	5	6	7	8	9	10
Q2.1	37	57	88	96	101	104	116	164
Q2.2	14	19	28	35	39	41	43	45
Q2.3	58	71	75	77	97	97	99	101
Q2.4	29	46	48	62	69	74	76	83
Q3.1	54	69	89	91	104	-	-	-
Q3.2	16	30	34	41	47	-	-	-
Q3.3	37	48	57	57	60	-	-	-
Q3.4	14	19	28	35	39	-	-	-
Q3.5	29	46	48	62	69	74	-	-
Q3.6	22	25	25	38	42	86	-	-
Q3.7	24	39	39	42	43	43	-	-
Q3.8	13	35	61	61	73	91	-	-