TU WIEN Informatics

Crossing Minimization in One-Sided Time-Interval Storylines with a Protagonist

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Felix Kainz
Matrikelnummer 11808817

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Inform. Dr.rer.nat. Martin Nöllenburg
Mitwirkung: Projektass. Dipl.-Ing. Alexander Dobler

Wien, 31. Oktober 2025

Felix Kainz

Martin Nöllenburg

TU WIEN Informatics

Crossing Minimization in One-Sided Time-Interval Storylines with a Protagonist

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Business Informatics

by

Felix Kainz
Registration Number 11808817

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Inform. Dr.rer.nat. Martin Nöllenburg
Assistance: Projektass. Dipl.-Ing. Alexander Dobler

Vienna, October 31, 2025

Felix Kainz

Martin Nöllenburg

Erklärung zur Verfassung der Arbeit

Felix Kainz

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 31. Oktober 2025

Felix Kainz

Danksagung

Mein besonderer Dank gilt meinem Betreuer und Mentor Martin Nöllenburg, der mich nicht nur während dieser Bachelorarbeit begleitet hat, sondern mir auch darüber hinaus den Zugang zur geometrischen Algorithmik eröffnet und mich stets motiviert hat. Ebenfalls danken möchte ich meinem Co-Betreuer Alexander Dobler, der mich durch die gesamte Arbeit unterstützt und mit vielen hilfreichen Anregungen vorangebracht hat. Für die regelmäßigen Treffen, die konstruktiven Diskussionen und den kontinuierlichen Fortschritt bin ich beiden sehr dankbar.

Ein weiterer Dank gilt meinem Bruder Simon Seelig, der mir mit ausführlichem inhaltlichem und formalem Feedback geduldig zur Seite stand. Außerdem möchte ich Franziska Seigner dafür danken, dass sie mir mit sprachlicher Präzision und einem guten Auge für flüssige Formulierungen geholfen hat.

Zuletzt möchte ich meiner Familie und meinen Freund*innen danken, die mich während der Arbeit und des gesamten Studiums unterstützt und motiviert haben.

Acknowledgements

I would like to express my sincere gratitude to my supervisor and mentor Martin Nöllenburg, who not only supported me throughout this Bachelor's thesis, but also introduced me to geometric algorithms and continuously encouraged my academic development. I am also very grateful to my co-supervisor Alexander Dobler, whose guidance and many helpful suggestions significantly contributed to the progress of this work. I thank both for the regular meetings, the constructive discussions, and the steady advancement they enabled.

My thanks further go to my brother Simon Seelig, who provided patient and extensive feedback on both content and structure. I would also like to thank Franziska Seigner for helping me improve the fluency and clarity of my English writing.

Lastly, I want to thank my family and my friends, who have supported and motivated me throughout this thesis and my entire studies.

Kurzfassung

Storyline-Visualisierungen stellen Interaktionen zwischen Charakteren über die Zeit hinweg dar, wobei x-monotone Kurven die Charaktere repräsentieren, die bei Interaktionen zusammenlaufen. Allerdings beeinträchtigt eine große Zahl sich kreuzender Linien schnell die Lesbarkeit. Diese Arbeit untersucht das *One-Sided Crossing Minimization in Time Interval Storylines with a Protagonist* (1-SCM-TI-P), eine Variante, bei der ein zentraler Charakter fixiert ist und Interaktionen innerhalb desselben Zeitintervalls ohne vorgegebene Reihenfolge umgeordnet werden dürfen. Wir entwickeln eine exakte ILP-Formulierung zur Kreuzungsminimierung sowie eine untere Schranke und entwerfen ein skalierbares heuristisches Verfahren, das eine TSP-basierte Initialisierung mit Simulated Annealing und strukturierten Nachbarschaften kombiniert. Experimente auf Basis von Kollaborationsnetzwerken aus DBLP zeigen, dass die Heuristiken in Millisekunden nahezu optimale Lösungen erreichen, während das ILP auf kleine oder nicht stark miteinander verbundene Instanzen beschränkt bleibt. Unsere Ergebnisse zeigen, dass hochwertige protagonistenzentrierte Storylines auch für große Instanzen effizient erzeugt werden können.

Abstract

Storyline visualizations encode interactions between characters over time, with x-monotone curves representing characters that converge when they interact. However, excessive line crossings quickly reduce readability. This thesis investigates the *One-Sided Crossing Minimization in Time Interval Storylines with a Protagonist* (1-SCM-TI-P), a variant where one main character remains fixed while interactions within the same temporal group may be reordered freely, as their order is not predetermined. We introduce an exact ILP formulation for crossing minimization together with a fast lower bound and develop a scalable heuristic framework combining a TSP-based initialization with Simulated Annealing using structured neighborhood operations. Experiments on real-world research collaboration networks derived from DBLP show that the heuristics achieve near-optimal solutions within milliseconds while the ILP is limited to small or sparse instances. The results demonstrate that high-quality layouts are feasible even for large instances, supporting efficient and readable protagonist-centered storylines in practical applications.

Contents

CHAPTER 1

Introduction

Stories often revolve around characters whose paths intertwine over time. When visualizing such narratives, a straightforward approach is to draw each character as a curve that moves through time from left to right. Whenever characters appear together in the story, their curves are drawn vertically next to each other, making their interactions easy to spot in the visualization.

This style of drawing, now referred to as a *storyline visualization*, became recognized through Randall Munroe's xkcd comic in 2009 [Mun09], which depicts the events of Tolkien's *The Lord of the Rings* as a single visual timeline (Figure 1.1).

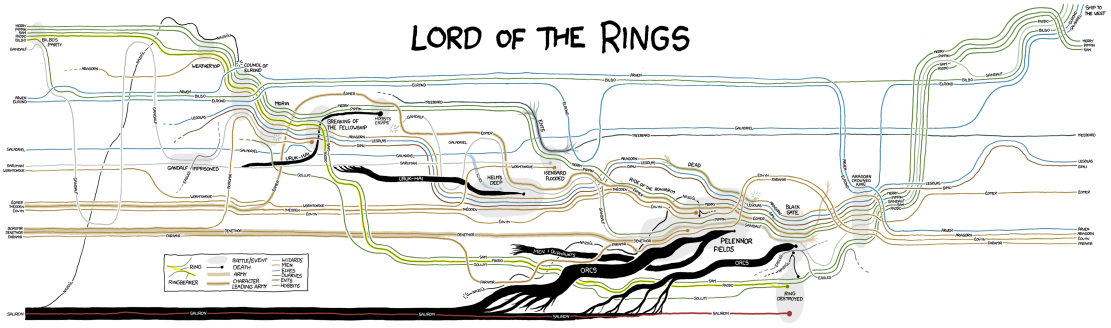


Figure 1.1: Storyline visualization of the events in *The Lord of the Rings*, illustrated by Munroe [Mun09].

Storyline visualizations provide an intuitive and visually appealing way to explore temporal relationships. However, as the number of characters and interactions grows, the drawing can quickly become cluttered and hard to read. In particular, each time two curves cross, the viewer must visually track which line continues where. Minimizing such crossings is therefore essential for preserving readability and supporting the narrative flow of the visualization.

While the idea is visually simple, constructing high-quality storyline layouts is computationally challenging. Each *interaction* imposes a local constraint on the vertical ordering of character curves at the corresponding point in time, and every swap of adjacent characters can introduce an additional crossing. As a result, finding a layout with as few curve crossings as possible becomes a challenging combinatorial optimization problem.

The core concept, first explored for software evolution visualization [OM10] was formally addressed by Tanahashi and Ma [TM12], who defined the aesthetic goal of minimizing line crossings. This has motivated several studies on crossing minimization in storyline drawings, connecting the problem to established concepts from graph drawing and layered layouts [Gro+16]. These works show that even simplified versions of storyline layouts are computationally hard, demonstrating the need for efficient heuristics when dealing with real-world data.

In many real-world scenarios, the visualization is not designed to treat all characters equally. A single central actor often drives the narrative, while others appear only for specific events. This is especially true in the academic collaboration setting considered in this thesis, where one researcher (the *protagonist*) is visualized together with their coauthors across years of publications. In such cases, the viewer's attention naturally follows the protagonist, and their position in the layout should remain consistent over time. This perspective leads to layouts in which the protagonist's vertical position is fixed, while the remaining characters may change position when they appear or disappear. Restricting the protagonist to appear on the top of the drawing, the *One-Sided Storyline Crossing Minimization with a Protagonist (1-SCM-P)* has recently been introduced and investigated by Hegemann and Wolff [HW24].

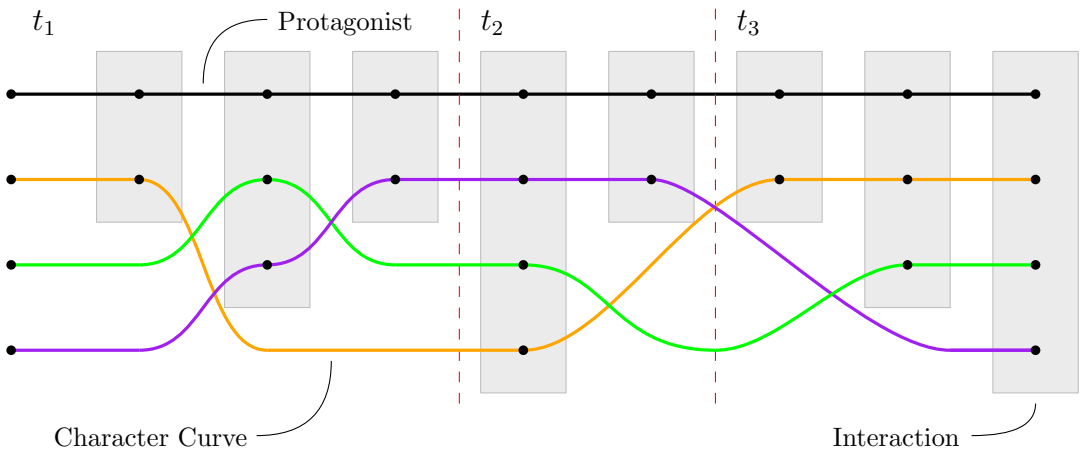


Figure 1.2: Exemplary, non-optimized storyline instance with 6 crossings. The black line on top is the protagonist. Interactions are marked with gray boxes. The order of interactions within a timestamp (separated by red dashed lines) and the vertical order of non-protagonist characters is subject to optimization.

We extend this one-sided protagonist storyline model by allowing *partial orderings* on

the time axis. Collaboration events are grouped into discrete *timestamps*, representing publication years in our researcher setting. Within each timestamp, several interactions may occur independently, and the exact left-to-right ordering of these events is not predetermined. This freedom allows us to reorder interactions within a timestamp in order to reduce both local crossings and crossings with interactions in neighboring timestamps. Partial-order formulations of storyline layouts have recently been investigated by Dobler et al. [Dob+23] for the non-protagonist general case.

We combine both settings, investigating the *One-Sided Crossing Minimization in Time Interval Storylines with a Protagonist (1-SCM-TI-P)*. An example of a problem instance is shown in Figure 1.2.

While the general storyline crossing minimization problem was shown to be NP-hard by Kostitsyna et al. [Kos+15] even without partial-order formulations, 1-SCM-P is efficiently solvable. To evaluate the effect of partially changing the order of interactions in 1-SCM-TI-P instances, we build on the 1-SCM-P algorithm proposed by Hegemann and Wolff [HW24]. This algorithm efficiently computes a minimal pairwise crossing configuration given a total ordering of interactions in $O(k^2 n)$ time, where k is the number of characters and n is the number of interactions. For our purposes, we treat the 1-SCM-P algorithm as a black box throughout the thesis.

It is an open problem whether 1-SCM-TI-P is NP-hard or efficiently solvable. The aim of this thesis is to investigate this problem from different, applied viewpoints. Chapter 2 goes more into detail on how this problem fits into surrounding research, and Chapter 3 introduces the concepts used for our approaches. Chapter 4 formally defines 1-SCM-TI-P, followed by exact approaches towards a solution in Chapter 5. In Chapter 6, we present a heuristic framework that aims to find near-optimal solutions. Chapter 7 evaluates the different approaches on real-world instances. We discuss our findings in Chapter 8 before concluding our work and giving an outlook on future work in Chapter 9.

CHAPTER 2

Related Work

Storyline visualizations encode entities as time-monotone curves and depict interactions by bringing curves into proximity, with readability primarily enhanced by minimizing crossings and supported by secondary criteria such as bends and whitespace, see Tanahashi and Ma [TM12]. The connection to layered drawing makes vertical permutations the central degrees of freedom for clarity and transforms storyline layouts to ordering problems in graph drawing, as formalized by Gronemann et al. [Gro+16].

From a computational viewpoint, several storyline variants are hard, which motivates restricted models and specific formulations, shown by Kostitsyna et al. [Kos+15]. To remain scalable, some approaches limit the interaction model or exploit domain structures, while others, such as Gronemann et al. [Gro+16], separate ordering from geometric fine-tuning.

Instead of focusing on pairwise crossings, an alternative objective counts *block crossings*, which are intersections between bundled groups of curves. The idea is to enhance readability by grouping similar crossings at once. Minimizing block crossings is NP-hard even under restricted interaction sizes, with tractable results only under additional parameters such as bounded bundle size or constrained interactions, documented by van Dijk et al. [Dij+16].

Time-interval storylines formalize the common situation where multiple events fall into the same time grouping (e.g., a publication year), permitting permutations *within* each interval to reduce crossings, as demonstrated by Dobler et al. [Dob+23]. This framework is especially well suited for data that lacks a precise, total ordering.

Another approach takes a protagonist- or ego-centric perspective to focus on one actor and their changing neighborhood over time, introduced by Muelder et al. [Mue+13]. Extending this idea with broader, application-driven design choices, SpreadLine presents a storyline-based framework for egocentric dynamic influence with case studies in epidemiology, social platforms, and academic careers, presented by Kuo et al. [KLM25]. These systems

highlight analytical value and visual stability for a single actor, providing a motivation for one-sided models that keep the protagonist visually consistent.

On the optimization side of protagonist layouts, *Storylines with a Protagonist* formalize one-sided models and provide efficient algorithms when the interaction order is fixed, distinguishing pairwise from block crossings and demonstrating practical performance on publication data, shown by Hegemann and Wolff [HW24]. They also consider the two-sided variant, in which the protagonist remains fixed while other characters may appear either above or below, but efficient procedures apply only when a crossing-free arrangement exists.

To conclude, looking beyond narrative settings, storyline concepts have been adapted to evolving networks where ordering and alignment choices reduce clutter while preserving temporal coherence. This line of work further underscores that discrete-time ordering is the main lever for readability in long sequences, highlighted by Arendt and Blaha [AB14].

CHAPTER 3

Preliminaries

Before formally defining the main problem investigated in this thesis, we provide a short overview of the fundamental optimization concepts used throughout this work, namely *Integer Linear Programming*, the *Traveling Salesperson Problem*, and *Simulated Annealing*. The goal is not to introduce these methods in full detail, but to summarize their functionality and notation as used in later chapters.

3.1 Integer Linear Programming

An *Integer Linear Program* (ILP) is an optimization model where both the objective function and the constraints are linear, but all decision variables are required to take integer values. Formally, we seek

$$\min_{x \in \mathbb{Z}^n} c^\top x \quad \text{s.t. } Ax \leq b,$$

where $x \in \mathbb{Z}^n$ denotes the vector of decision variables, $c \in \mathbb{R}^n$ the objective coefficients, and $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ define the linear constraints.

Restricting the decision variables to integer values turns linear programming (polynomial-time solvable) into an NP-hard problem. Consequently, no polynomial-time algorithm is known, and solving such models may require exponential time in the worst case [NW88].

ILPs are often used on small and medium-sized instances, providing exact benchmarks for heuristics and approximation algorithms. On larger instances, solvers may find feasible solutions but fail to prove optimality within time limits. The quality of the current solution can be inferred from the *optimality gap*, i.e., the relative difference between the best known feasible objective (upper bound) and the best known lower bound from the relaxation and the search tree.

After defining the problem formally in Chapter 4, we will extensively discuss the application of integer linear programming to our problem in Chapter 5, and the heuristic results

for our small and medium-sized instances in Chapter 7 will be compared against our ILP results.

3.2 The Traveling Salesperson Problem

A classical combinatorial optimization problem structurally related to our storyline optimization approach is the *Traveling Salesperson Problem* (TSP). Given a set of nodes $V = \{1, \ldots, n\}$ and a cost matrix $C = (c_{ij})$ representing the cost of traveling from node i to node j, the TSP seeks a minimum-cost route that visits each node exactly once and returns to the starting node. In the symmetric case, $c_{ij} = c_{ji}$. In the asymmetric case, travel costs depend on direction.

The TSP is NP-hard in the optimization sense and NP-complete in its decision variant [Law95]. Nevertheless, exact solvers such as *Concorde* exploit branch-and-cut techniques and polyhedral properties to optimally solve instances with thousands of nodes [Tri08].

Approximating our problem as a TSP instance will be the starting point for our heuristic approaches shown in Chapter 6. We will then evaluate how closely the resulting TSP-based solutions approximate the true optimum in Chapter 7, where we also discuss different options to transform our problem into a valid TSP formulation.

3.3 Simulated Annealing

Once a valid solution has been found, different strategies can be used to iteratively improve it. *Simulated Annealing* (SA) is a metaheuristic that explores the solution space by occasionally accepting worsening moves, allowing the search to escape local minima [KGV83]. Alternative candidate solutions are generated by applying small modifications to the current one, which together define its *neighborhood*. Chapter 6 introduces several such neighborhood operators, which are then compared with each other in Chapter 7.

A proposed neighbor with cost difference Δ relative to the current solution is accepted with probability

$$\Pr[\text{accept}] = \begin{cases} 1, & \Delta < 0, \\ \exp(-\Delta/T), & \Delta \geq 0, \end{cases}$$

where $T > 0$ is the temperature parameter. The higher the remaining temperature, the greater the chance of accepting a worse solution. Following standard practice, we use geometric cooling $T_{k+1} = \alpha T_k$, where the cooling factor $\alpha \in (0, 1)$ directly controls the rate at which this probability decreases over time.

CHAPTER 4

Problem Setting

We formally define the *One-Sided Crossing Minimization in Time Interval Storylines with a Protagonist (1-SCM-TI-P)*. In our storyline visualization, a designated *protagonist* character appears in every interaction and is always placed at the top of the drawing. The objective is to order the interactions and characters such that the total number of crossings between non-protagonist character curves is minimized.

4.1 Input Representation

An instance is given by a triple

$$\mathcal{S} = (\mathcal{C}, \mathcal{I}, \mathcal{T}),$$

where

- $\mathcal{C} = \{c_P, c_1, \ldots, c_n\}$ is the set of characters, where c_P denotes the fixed *protagonist* and n the number of non-protagonists.
- $\mathcal{T} = \{t_1, \ldots, t_p\}$ is a totally ordered set of p timestamps
- $\mathcal{I} = \{I_1, \ldots, I_m\}$ is a set of m interactions, each of the form

  $$I_j = (C_j, t_j),$$

  where $C_j \subseteq \mathcal{C}$ is the subset of characters participating in interaction I_j, and $t_j \in \mathcal{T}$ is the associated timestamp.

By definition, the protagonist participates in every interaction, i.e., $c_P \in C_j$ for all $I_j \in \mathcal{I}$.

The visualization is organized into vertical *columns*, one for each interaction, which are then placed in a left-to-right order according to their timestamp in $\mathcal{T}$. All interactions

with the same timestamp are grouped together, but their internal order is not fixed and must be determined as part of the solution. Within each column, the protagonist c_P is placed at the top, and all participants of the corresponding interaction I_j appear consecutively below c_P. Characters not participating in a given interaction may appear in arbitrary order below the participants. Formally, let $\mathcal{A} = (A_1, \ldots, A_m)$ denote the ordered sequence of all columns, obtained by arranging interactions according to the chosen timestamp and within-timestamp order.

4.2 Formal Problem Definition

The task is to compute

1. a *total ordering* of interactions within each timestamp $t \in \mathcal{T}$, and
2. a *vertical ordering* of all characters in each column $A \in \mathcal{A}$

such that the total number of crossings among non-protagonist character curves is minimized.

Each column A_i corresponds to a vertical slice of the storyline and is associated with an interaction $I_j = (C_j, t_j)$. The vertical ordering $\pi_i : \mathcal{C} \to \{1, \ldots, |\mathcal{C}|\}$ assigns each character a position.

To ensure interaction continuity, all participants of an interaction I_j must occupy contiguous positions in each column associated with timestamp t_j, i.e.,

$$\max_{c_k \in C_j} \pi_i(c_k) - \min_{c_k \in C_j} \pi_i(c_k) = |C_j| - 1 \quad \text{for all } A_i \text{ of } t_j.$$

Given two adjacent columns A_1 and A_2 (either consecutive within a timestamp or the last column of t and the first column of $t+1$), a crossing between two non-protagonist characters $c_i, c_j \in \mathcal{C} \setminus \{c_P\}$ occurs if their relative vertical order differs in A_1 and A_2. The overall crossing count is obtained by summing over all adjacent column pairs.

4.3 Crossing Definition

Let $\pi(A_1)$ and $\pi(A_2)$ denote the vertical orderings π_i induced by columns A_1 and A_2, respectively. We measure the number of crossings between two adjacent columns A_1 and A_2 using the Kendall-tau distance

$$d(A_1, A_2) := K\big(\pi(A_1), \pi(A_2)\big),$$

which counts the number of pairwise order disagreements between $\pi(A_1)$ and $\pi(A_2)$ on the fixed ground set $\mathcal{C} \setminus \{c_P\}$. Equivalently, a crossing between c_i and c_j occurs if c_i is above c_j in A_1 but below c_j in A_2, or vice versa. Summing $d(A_1, A_2)$ over all adjacent column pairs yields the total number of crossings.

Objective. The goal of 1-SCM-TI-P is to find an interaction ordering and character orderings minimizing

$$\text{Total Crossings} = \sum_{\text{adjacent } (A_1, A_2)} d(A_1, A_2).$$

Since the protagonist c_P is always at the top and participates in every interaction, no other curve can cross c_P, and all crossings involve only non-protagonist pairs.

While the routing and displaying of individual curves can be handled separately, the number of pairwise crossings is fully determined by the vertical orderings chosen at each timestamp. In this work, we therefore focus exclusively on optimizing these orderings, while the final curve drawing is delegated to the SCMP algorithm by Hegemann and Wolff [HW24].

CHAPTER 5

Exact Approaches

Before moving towards efficient, but heuristic implementations, we describe several exact approaches for solving or bounding 1-SCM-TI-P. The main contribution is an integer linear program (ILP) that calculates optimal solutions for small- and medium-sized instances and serves as a benchmark for the heuristic methods discussed in Chapter 7. In addition, we include an extended ILP variant with *active character ranges*, which allows characters to join the storyline at their first appearance, and lets them leave as soon as their last interaction has occurred. Finally, we describe a simple algorithmic lower bound that provides fast benchmarks against which heuristic solutions can be compared.

5.1 Base ILP Formulation

The following ILP formulation extends the model (*ILP1*) proposed by Dobler et al. [Dob+23], which minimizes crossings for instances without a protagonist under a fixed timestamp order, but optimizes the interaction order within each timestamp. Our formulation relies on three types of binary variables to model the problem:

1. *Assignment variables* $y_{t,p,I}$ to encode the placement of each interaction I at a particular column p of timestamp t;
2. *Ordering variables* x_{t,p,c_i,c_j} to encode the vertical order of characters within each column;
3. *Crossing variables* z_{t,p,c_i,c_j} to indicate crossings between adjacent columns.

Interaction Assignment. For each timestamp $t \in \mathcal{T}$, let $\mathcal{I}_t \subseteq \mathcal{I}$ be the set of interactions occurring at t. We introduce binary variables $y_{t,p,I}$ for $p = 1, \ldots, |\mathcal{I}_t|$, indicating whether interaction I is assigned to column p at timestamp t.

Each interaction must be assigned to exactly one column, and each column must host exactly one interaction:

$$\sum_{p=1}^{|\mathcal{I}_t|} y_{t,p,I} = 1, \qquad \forall t \in \mathcal{T}, I \in \mathcal{I}_t, \tag{5.1}$$

$$\sum_{I \in \mathcal{I}_t} y_{t,p,I} = 1, \qquad \forall t \in \mathcal{T}, p = 1, \ldots, |\mathcal{I}_t|. \tag{5.2}$$

Character Ordering. For each column (t, p) and each ordered pair of distinct characters $c_i, c_j \in \mathcal{C}$ with $i < j$, we introduce binary variables x_{t,p,c_i,c_j} indicating whether c_i is placed above c_j in column (t, p). To ensure that the ordering in each column is a total order, we set up transitivity constraints:

$$0 \leq x_{t,p,c_i,c_j} + x_{t,p,c_j,c_h} - x_{t,p,c_i,c_h} \leq 1, \quad \forall c_i, c_j, c_h \in \mathcal{C}, i < j < h. \tag{5.3}$$

Without loss of generality, we index the protagonist c_P as the smallest character (i.e., index $c_P = c_1$). The protagonist is always placed at the top of every column:

$$x_{t,p,c_P,c} = 1, \quad \forall t \in \mathcal{T}, p = 1, \ldots, |\mathcal{I}_t|, \forall c \in \mathcal{C} \setminus \{c_P\}. \tag{5.4}$$

Finally, participants of the current interaction must appear above all non-participating characters, ensuring contiguity:

$$x_{t,p,c_i,c_j} \geq y_{t,p,I}, \qquad \forall t \in \mathcal{T}, p = 1, \ldots, |\mathcal{I}_t|, I \in \mathcal{I}_t, c_i \in C_I, c_j \notin C_I, i < j, \tag{5.5}$$

$$1 - x_{t,p,c_i,c_j} \geq y_{t,p,I}, \qquad \forall t \in \mathcal{T}, p = 1, \ldots, |\mathcal{I}_t|, I \in \mathcal{I}_t, c_i \notin C_I, c_j \in C_I, i < j. \tag{5.6}$$

Crossings. The key ingredient of the formulation is the linearization of curve crossings between pairs of characters in adjacent columns. Since the protagonist will never switch positions, we define $\mathcal{C}' := \mathcal{C} \setminus \{c_P\}$. For each ordered pair of characters $c_i, c_j \in \mathcal{C}'$ with $i < j$ and each pair of adjacent columns, we introduce a binary variable

$$z_{t,p,c_i,c_j} = \begin{cases} 1 & \text{if the relative order of } c_i \text{ and } c_j \text{ is reversed between the two columns,} \\ 0 & \text{otherwise.} \end{cases}$$

The crossing constraints distinguish between adjacent columns within the same timestamp and columns between two consecutive timestamps.

For two adjacent columns (t, p) and $(t, p+1)$ within the same timestamp t, crossings are enforced by

$$z_{t,p,c_i,c_j} \geq x_{t,p,c_i,c_j} - x_{t,p+1,c_i,c_j}, \quad \forall t \in \mathcal{T},\ p = 1, \ldots, |\mathcal{I}_t| - 1, c_i, c_j \in \mathcal{C}', i < j, \tag{5.7}$$

$$z_{t,p,c_i,c_j} \geq x_{t,p+1,c_i,c_j} - x_{t,p,c_i,c_j}, \quad \forall t \in \mathcal{T},\ p = 1, \ldots, |\mathcal{I}_t| - 1, c_i, c_j \in \mathcal{C}', i < j. \tag{5.8}$$

For the boundary between two consecutive timestamps t and $t+1$, in other words the last column of t and the first column of $t+1$, we add

$$z_{t,|\mathcal{I}_t|,c_i,c_j} \geq x_{t,|\mathcal{I}_t|,c_i,c_j} - x_{t+1,1,c_i,c_j}, \quad \forall t \in \mathcal{T}, c_i, c_j \in \mathcal{C}', i < j,\, t < |\mathcal{T}|, \tag{5.9}$$

$$z_{t,|\mathcal{I}_t|,c_i,c_j} \geq x_{t+1,1,c_i,c_j} - x_{t,|\mathcal{I}_t|,c_i,c_j}, \quad \forall t \in \mathcal{T}, c_i, c_j \in \mathcal{C}', i < j,\, t < |\mathcal{T}|. \tag{5.10}$$

Together with the minimization objective, these four constraints ensure that, in an optimal solution, $z_{t,p,c_i,c_j} = 1$ if and only if the relative vertical order of c_i and c_j differs between the two columns, which exactly matches the definition of a crossing.

Objective. The objective function minimizes the total number of crossings by summing over all of the crossing variables:

$$\text{minimize} \sum_{t \in \mathcal{T}} \sum_{p=1}^{|\mathcal{I}_t|-1} \sum_{\substack{c_i,c_j \in \mathcal{C}' \\ i<j}} z_{t,p,c_i,c_j} + \sum_{t=1}^{|\mathcal{T}|-1} \sum_{\substack{c_i,c_j \in \mathcal{C}' \\ i<j}} z_{t,|\mathcal{I}_t|,c_i,c_j}. \tag{5.11}$$

This is equivalent to minimizing the total number of pairwise order disagreements between consecutive columns, i.e. the total crossing count in the visualization.

5.2 Extension: Active Character Ranges

While the remaining thesis deals with storylines where all character curves are apparent throughout all interactions, it is interesting to also investigate the variant where character curves are only drawn over the time span in which they appear. We will now provide a modified variant of the ILP that supports this modification. Intuitively, each character c becomes active at its first appearance and remains active until its last appearance; outside this range the character does not participate in crossings. We adapt our previously introduced constraints to model this new behavior. The added constraints are inspired by the active variant (*ILP2*) of the same work [Dob+23], adapted here to our protagonist-based setting.

Activation Variables. For each character $c \in \mathcal{C}$ and each column (t, p), we introduce a binary variable $a_{c,t,p}$ indicating whether c is active in column (t, p). If interaction I is assigned to column (t, p) and $c \in C_I$, then c must be active:

$$a_{c,t,p} \geq y_{t,p,I}, \quad \forall\, t \in \mathcal{T}, p = 1, \ldots, |\mathcal{I}_t|, \forall\, I \in \mathcal{I}_t, \forall\, c \in C_I. \tag{5.12}$$

Contiguous Activity. We linearly order columns lexicographically by timestamp and in-timestamp index:

$$(t, p) < (t', p') \iff (t < t') \text{or} (t = t' \text{ and } p < p').$$

For any triple of columns $(t, p) < (t', p') < (t'', p'')$, we enforce that activity is contiguous:

$$a_{c,t',p'} + 1 \geq a_{c,t,p} + a_{c,t'',p''}, \quad \forall\, c \in \mathcal{C}. \tag{5.13}$$

Crossings with Activity. Crossings are counted only when both characters are active in the two columns that define the adjacency. Compared to the base formulation, we add an offset that activates the constraints if and only if all four activity indicators are 1. These constraints replace constraints (5.7) to (5.10) from the base formulation.

For adjacent columns within the same timestamp t (columns p and $p+1$):

$$z_{t,p,c_i,c_j} \geq x_{t,p,c_i,c_j} - x_{t,p+1,c_i,c_j} - 4 + a_{c_i,t,p} + a_{c_i,t,p+1} + a_{c_j,t,p} + a_{c_j,t,p+1}, \quad \forall t \in \mathcal{T}, p = 1, \ldots, |\mathcal{I}_t| - 1, i < j, \tag{5.14}$$

$$z_{t,p,c_i,c_j} \geq x_{t,p+1,c_i,c_j} - x_{t,p,c_i,c_j} - 4 + a_{c_i,t,p} + a_{c_i,t,p+1} + a_{c_j,t,p} + a_{c_j,t,p+1}, \quad \forall t \in \mathcal{T}, p = 1, \ldots, |\mathcal{I}_t| - 1, i < j. \tag{5.15}$$

For the boundary between timestamps t and $t+1$ (last column of t, first of $t+1$):

$$z_{t,|\mathcal{I}_t|,c_i,c_j} \geq x_{t,|\mathcal{I}_t|,c_i,c_j} - x_{t+1,1,c_i,c_j} - 4 + a_{c_i,t,|\mathcal{I}_t|} + a_{c_i,t+1,1} + a_{c_j,t,|\mathcal{I}_t|} + a_{c_j,t+1,1}, \quad \forall t \in \mathcal{T}, t < |\mathcal{T}|, i < j, \tag{5.16}$$

$$z_{t,|\mathcal{I}_t|,c_i,c_j} \geq x_{t+1,1,c_i,c_j} - x_{t,|\mathcal{I}_t|,c_i,c_j} - 4 + a_{c_i,t,|\mathcal{I}_t|} + a_{c_i,t+1,1} + a_{c_j,t,|\mathcal{I}_t|} + a_{c_j,t+1,1}, \quad \forall t \in \mathcal{T}, t < |\mathcal{T}|, i < j. \tag{5.17}$$

The objective remains to minimize the sum of z-variables as in the base formulation (5.11). The introduced activity terms only control whether a crossing can contribute.

5.3 Algorithmic Lower Bound

As ILPs are not suitable for solving large instances, we alternatively describe an algorithmic approach that efficiently computes a weak lower bound by counting crossings that are unavoidable when reducing the storyline problem to pairs of interacting characters.

The idea is to run a dynamic program (DP) on each unordered pair of (non-protagonist) characters (a, b) independently. For each pair and each timestamp t, we construct a small 2×2 matrix costIntra_t that captures the minimal number of flips (required crossings) between the start and end state of the pair at this timestamp. The two possible states correspond to the relative vertical order of characters a and b: state 0 indicates a is above b, state 1 indicates b is above a. Rows correspond to the start state, columns to the end state.

Depending on whether only a participates at timestamp t, only b, both, or neither, costIntra_t takes one of the four fixed forms shown in Figure 5.1.

For each pair (a, b), we then run a simple dynamic program over the sequence of timestamps. The optimal initial state is chosen by the DP itself. The DP maintains, for each possible end state at timestamp t, the minimal number of required crossings up to t. At each step, the DP transitions between start and end states using the corresponding costIntra_t matrix. The final value is the minimum over the two possible end states after

the last timestamp. Summing these pairwise minima over all unordered character pairs yields a mandatory minimum on the total number of crossings.

The DP runs independently for each unordered pair of non-protagonists and scans all interactions across timestamps. Its overall complexity is

$$O\left(\binom{n}{2}\sum_{t\in\mathcal{T}}|\mathcal{I}_t|\right).$$

While this approach is very time-efficient, it is rather limited in its application. As we ignore the (sometimes necessarily large) distances between a pair of characters, we are severely undercounting mandatory crossings in instances. We will discuss the gravity of the so-induced quality loss in Chapter 7.

	0	1
0	0	1
1	1	0

(a) Neither exclusive

	0	1
0	0	∞
1	1	∞

(b) Only a exclusive

	0	1
0	∞	1
1	∞	0

(c) Only b exclusive

	0	1
0	2	1
1	1	2

(d) Both exclusive

Figure 5.1: Intra-cost matrices for the four possible participation patterns of a pair (a, b) at timestamp t. Rows indicate the start state (0: a above b, 1: b above a), columns the end state.

CHAPTER 6

Heuristics

In the previous chapter, we introduced an exact but slow solving procedure and a fast yet weak lower-bound dynamic program. Both are unsuitable for practical applications involving larger instances. We now turn to heuristic approaches. First, we generate an initial solution using an efficient Traveling Salesperson Problem (TSP) solver. Building on that, we investigate optimization methods based on a generic Simulated Annealing (SA) framework, with particular focus on the definition and generation of solution neighborhoods. Finally, we present a postprocessing step that further reduces crossings wherever possible.

6.1 TSP-Based Initial Solution Generation

As already discussed in Chapter 1, once the order in which interactions are visualized is fixed, 1-SCM-TI-P with a protagonist can be solved efficiently using the SCMP algorithm proposed by Hegemann and Wolff [HW24]. Consequently, our task reduces to determining an effective total ordering of all interactions within each timestamp.

The Traveling Salesperson Problem (TSP) provides a natural model for this task. Each interaction corresponds to a city that must be visited exactly once, and the goal is to minimize the total traversal cost, which here reflects expected crossings between consecutive interactions. There are, however, two key restrictions:

1. All interactions within a timestamp must be visited before moving on to the interactions of the next timestamp, and

2. the TSP instance is *asymmetric*: timestamps must be processed in increasing order, and a reversal of the chosen route is prohibited.

6.1.1 The Asymmetric Model

To construct a valid TSP instance, we represent each interaction as an *interaction node*. Additionally, for every timestamp we introduce a single *helper node* that acts as a reset point, ensuring that the tour visits all interactions of a timestamp before moving to the next.

The edges between interaction nodes are weighted according to a chosen *weight metric*, which estimates the number of crossings that would occur if one interaction were placed before another within the same timestamp. We provide two such metrics:

- The *Crossing Distance* counts the unavoidable crossings that occur when moving from one interaction I_1 to the next I_2. It counts how many characters leave after I_1 and how many new characters join at I_2, and multiplies these two numbers. The product reflects the number of pairwise crossings that would necessarily occur when I_1 and I_2 are visualized sequentially.
- The *Hamming Distance* measures the dissimilarity between the sets of participating characters of two interactions. It counts how many characters change their participation status between I_1 and I_2, regardless of whether they join or leave between the interactions.

Table 6.1 shows two examples of interactions I_1, I_2 and their respective weights. Note that within a timestamp, the TSP is symmetric, as the crossings induced by moving from I_1 to I_2 match the crossings induced by moving from I_2 to I_1.

I_1 participants	I_2 participants	$\lvert L\rvert$	$\lvert R\rvert$	**CD**	**HD**
$\{A, B, C\}$	$\{A, C, D\}$	1	1	1	2
$\{A, B, C, D\}$	$\{E, F\}$	4	2	8	6

Table 6.1: Examples of interaction pairs (I_1, I_2) with their respective sets of participants. $|L|$ and $|R|$ denote the number of characters leaving and joining between I_1 and I_2, respectively. CD indicates the Crossing Distance ($|L| \cdot |R|$), and HD the Hamming Distance (number of participation changes). In the first example, CD $<$ HD, while in the second, HD $<$ CD.

Each interaction node is connected to all other interaction nodes within the same timestamp. Furthermore, each interaction node has a one-way (asymmetric) edge to the helper node of the corresponding timestamp at cost 0. That helper node then again has outgoing edges at cost 0 to all interaction nodes of the next timestamp. Helper nodes are never connected with other helper nodes. The helper node of the last timestamp connects to the interaction nodes of the first timestamp, ensuring that a cyclic route exists. Figure 6.1 shows an exemplary asymmetric TSP (ATSP) instance.

Last, we formally define the cost matrix C for the ATSP. Assigning costs for allowed moves by the procedure described above, we assign a large integer U to each invalid move.

Moves from a node to itself are assigned cost 0, though using such an edge is impossible in the current setting. The cost matrix for Figure 6.1 is shown in Table 6.2.

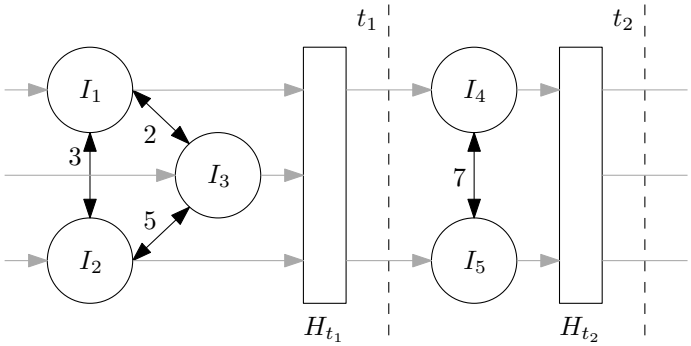


Figure 6.1: Each interaction is represented as a node. Within each timestamp t, directed black edges connect interactions and are weighted by the chosen metric. Gray edges indicate zero-cost transitions: interactions connect to their helper node H_t (exit), and each H_t connects to all interactions of the next timestamp $t{+}1$ (entry). Additional edges from $H_{t_{\max}}$ to the first timestamp t_1 close the tour.

	I_1	I_2	I_3	H_{t_1}	I_4	I_5	H_{t_2}
I_1	0	3	2	0	U	U	U
I_2	3	0	5	0	U	U	U
I_3	2	5	0	0	U	U	U
H_{t_1}	U	U	U	0	0	0	U
I_4	U	U	U	U	0	7	0
I_5	U	U	U	U	7	0	0
H_{t_2}	0	0	0	U	U	U	0

Table 6.2: ATSP cost matrix C for the example in Fig. 6.1. U marks forbidden transitions.

6.1.2 Symmetric Transformation

While the asymmetric model described above accurately represents the ordering constraints of our problem, it cannot be solved directly using a standard TSP solver such as Concorde [Tri08], which only accepts *symmetric* TSP instances. Therefore, we transform our asymmetric cost matrix into an equivalent symmetric one that preserves the optimal tour. We do so by applying a transformation approach presented by Jonker and Volgenant [JV83], which we briefly describe here.

Let n be the number of asymmetric nodes and let C denote the (nonnegative) ATSP cost matrix defined above. We derive $\bar{C}$ by introducing a large constant M, and assign

$$\bar{c}_{ij} = \begin{cases} c_{ij} + M & \text{for } i \neq j \\ 0 & \text{for } i = j \end{cases}$$

We now build a symmetric $2n \times 2n$ matrix S over the duplicated node set $\{1, \ldots, n, n+1, \ldots, 2n\}$ as follows:

$$S = \begin{pmatrix} U & \bar{C}^\top \\ \bar{C} & U \end{pmatrix}$$

The upper-left and lower-right blocks U forbid transitions within the same half, while the off-diagonal blocks $\bar{C}$ and $\bar{C}^\top$ encode the directed costs in symmetric form. The intuition behind this construction is that we force any optimal solution to perform alternate moves between $\bar{C}$ and $\bar{C}^\top$ (which are at cost 0 due to the definition of c_{ii}). Therefore, the solution takes on the form

$$i_1 \to (i_1 + n) \to \cdots \to i_n \to (i_n + n) \to i_1$$

or its reversed equivalent. Determining the correct solution of the initial ATSP is then done by simply ignoring the $(i + n)$ entries and performing a quick validation check if the final sequence is reversed, re-reversing if necessary.

The only thing left to do is to find appropriate values for U and M. To support large instances, we aim to keep these constants as small as possible while ensuring that no optimal solution can include an illegal move or violate the alternating principle described above.

We first define M. Since M is applied on every legal move that is not a jump from a node i to its corresponding duplicate $i + n$, we need to ensure that moving to a node without coming from the corresponding duplicate (or, respectively, without moving to the corresponding duplicate afterwards) is never a valid choice. We achieve this effect by defining

$$M = (c_{\max} - c_{\min}) \cdot 2n + 1,$$

where $c_{\min}$ and $c_{\max}$ correspond to the minimal (resp. maximal) edge cost in the original ATSP matrix. Since the total variation between the smallest and largest possible legal edge costs across the $2n$ transitions of a tour is bounded by $(c_{\max} - c_{\min}) \cdot 2n$ (that is, the worst-case difference between two legal tours), choosing M slightly above this range guarantees that no illegal shortcut can ever become competitive with a valid alternating sequence. In other words, the offset M ensures that any deviation from the intended pattern (such as skipping a duplicate node) would increase the total cost by more than any possible gain from lower edge weights, effectively enforcing the alternating structure of the symmetric instance.

Last, we choose U in a way that just using a single illegal edge assigned with this weight exceeds the largest possible cost of a valid tour. As such a tour needs to use n edges of maximum weight $M + c_{\max}$ and another n transitions to duplicated nodes at cost 0, we can safely upper-bound the maximum route cost by

$$U = (n + 1) \cdot (M + c_{\max}) + 1.$$

Having defined parameters M and U, we successfully transformed the original ATSP formulation into an equivalent symmetric TSP formulation using only nonnegative weights.

6.2 Simulated Annealing Framework and Neighborhoods

With the TSP model set up, we can now generate initial total orderings, and consequently initial solutions effectively. However, since the quality of the solver depends on the heuristic weight metrics used, it is worthwhile to explore strategies that improve the initial ordering. By modifying the total order, we aim to further reduce crossings efficiently. In this section, we first outline the generic Simulated Annealing (SA) framework and describe its adaptation to 1-SCM-TI-P. We then explore three different neighborhood definitions of increasing complexity.

6.2.1 Simulated Annealing Framework

In our context, a solution state is a total ordering of interactions within each timestamp. The energy of a state is the total number of crossings in the resulting drawing, as defined in Section 4.3. Building on the SA Framework introduced in Chapter 3, we make two specific adaptions:

1. we calibrate the initial temperature T_0 and the cooling schedule from the instance itself, and
2. we evaluate a batch of candidate neighbors per iteration and either pick the best candidate or a random one from the batch before applying the acceptance test.

Temperature Calibration. We estimate the scale of uphill moves directly from the initial solution received by the TSP. Let Δ^+ denote positive deltas (worse moves). We sample s random neighbors of the initial solution and compute

$$\overline{\Delta^+} = \frac{1}{|\{\Delta > 0\}|} \sum_{\Delta > 0} \Delta.$$

Note that this approach does not directly relate to the initial sample size s, as we only average on positive deltas. This is because for the initial SA phase, we want to encourage the acceptance of worsening moves, and want to avoid starting with a too restrictive temperature in case the initial TSP-based solution is already close to a local optimum.

We then choose two target acceptance probabilities p_0 and p_{end} and set

$$T_0 = -\frac{\overline{\Delta^+}}{\ln p_0}, \qquad T_{\text{end}} = -\frac{\overline{\Delta^+}}{\ln p_{\text{end}}}.$$

With a geometric schedule $T_{k+1} = \alpha T_k$ over I iterations, we obtain

$$\alpha = \left(\frac{T_{\text{end}}}{T_0}\right)^{1/I}.$$

Empirical values for s, p_0 and p_{end} will be determined experimentally and discussed in Chapter 7.

Neighborhood evaluation. In each iteration we ask the neighborhood generator for a batch of candidates. We compute their crossing counts and select either the best candidate or one at random from the batch. Importantly, even with picking the best candidate, we still apply the standard SA rule to decide whether to move:

$$\text{accept with probability } \begin{cases} 1, & \Delta < 0 \\ \exp(-\Delta/T), & \Delta \geq 0 \end{cases}$$

Here, Δ is the candidate's cost minus the current cost. With this modification, we purposely bias the candidate selection without turning SA into a pure greedy algorithm, creating a mixed approach.

6.2.2 Neighborhood Generation

Having the generic SA Framework set up, we now prepare several concrete approaches on how to modify the total interaction ordering. In metaheuristic optimization, the definition of a neighborhood function determines how the search space is explored and thus directly affects convergence and solution quality. In the following, we investigate three conceptually distinct neighborhood definitions: a naive *Random Swap*, a *Weight-based Swap* focusing on single, high-crossing interactions, and a *Group-based Swap* that jointly moves related interactions. While they are introduced in this section, their comparison in terms of runtime and result quality will be discussed in Chapter 7.

In our implementation, neighborhoods are generated by reordering interactions within their respective timestamps. As explained in Chapter 1, once a total order has been fixed, the corresponding optimal vertical placement of characters can be computed using the SCMP algorithm. Accordingly, for every neighborhood candidate we first apply the SCMP algorithm to determine the resulting layout and then compute the total number of crossings via an inversion-based counting procedure ($\Theta(n \log n)$ per interaction). Since the SCMP algorithm itself is treated as a black box in this thesis, we denote the combined cost of running SCMP and evaluating crossings by a single unit T_{eval}.

Random Swap

The most straightforward approach to generate neighbors is to randomly select two interactions within a randomly chosen timestamp and swap their positions. This is exactly the approach how the *Random Swap* generator operates. It performs no weighting or structural evaluation: first, a timestamp is selected uniformly at random, then two interactions within it are changed (regardless of their local fit), and the resulting configuration is returned to the SA framework as a new candidate.

Unlike other neighborhood generators, the Random Swap generator produces only a single neighbor per iteration rather than a full batch of candidates. This keeps the computational overhead negligible and results in an extremely fast search process. As only one neighbor is evaluated at a time, this variant effectively represents a *pure* form

of Simulated Annealing, relying entirely on probabilistic acceptance rather than the localized search bias presented in the prior section.

Assuming that the size of each timestamp is known, each iteration requires only one full evaluation of the resulting configuration. The overall time per iteration is therefore $O(T_{\text{eval}})$.

Weight-Based Swap

The *Weight-Based Swap* generator directs the search toward timestamps and interaction pairs that currently contribute most to the total number of crossings. The procedure operates as follows.

First, each timestamp is assigned a weight proportional to the total number of crossings between all adjacent interaction pairs within that timestamp. One timestamp is then selected by weighted random sampling, giving preference to timestamps with higher local complexity.

Within the chosen timestamp, all adjacent interaction pairs are evaluated, and one pair is again selected proportionally to its number of crossings. This defines a local region of interest around which new solutions are generated. If the timestamp is small (i.e., it contains no more than a given threshold of interactions), all possible swaps within the timestamp are enumerated. Otherwise, a fixed number of random swaps are created by moving one of the two highly crossing interactions to a new position within the timestamp, supplemented by a few purely random swaps. Each of these modified configurations forms a candidate neighbor for the SA framework.

This generator therefore balances focus (by prioritizing high-crossing regions) and diversity (through random selections). While the computational cost is significantly increased in comparison to the random swapping, the generated neighbors are typically more meaningful.

Let $|\mathcal{I}|$ denote the total number of interactions, $|\mathcal{T}|$ the number of timestamps, and B the batch size. Timestamp weighting requires a single pass over all adjacent column pairs, costing $O\big((|\mathcal{I}| - |\mathcal{T}|) \cdot n \log n\big)$ via the crossing count. Within the chosen timestamp of size ℓ, building pair weights adds $O\big((\ell - 1) \cdot n \log n\big)$. Candidate evaluation dominates:

$$\text{cost} = \begin{cases} \binom{\ell}{2} T_{\text{eval}}, & \ell \leq \text{smallTimestampThreshold}, \\ B \cdot T_{\text{eval}}, & \text{otherwise.} \end{cases}$$

Hence, one call costs

$$O\big((|\mathcal{I}| - |\mathcal{T}| + \ell) \cdot n \log n\big) + \begin{cases} \binom{\ell}{2} T_{\text{eval}}, \\ B \cdot T_{\text{eval}}. \end{cases}$$

Group-Based Swap

The *Group-Based Swap* generator moves not just a single interaction but a contiguous *block* of interactions within one timestamp. As before, timestamps are sampled with probability proportional to their current crossing contribution. Within the chosen timestamp, each pair of adjacent interactions is assigned a weight according to the chosen weight metric of the TSP solver (e.g., Crossing or Hamming Distance). Two boundaries between adjacent interactions are sampled according to their edge weights; the interactions enclosed between these two boundaries define the movable block.

This block is removed and reinserted at a new position within the same timestamp. The insertion boundary is chosen probabilistically, favoring positions that minimize the metric cost between the block and its new left and right neighbors. Repeating this process produces a batch of diverse candidate solutions.

By moving groups of closely related interactions together, this generator enables larger, more structured changes in the solution space while preserving local coherence within blocks. Compared to single-swap neighborhoods, it facilitates stronger diversification and helps escape local minima, at the expense of increased computational effort per iteration.

Let $|\mathcal{I}|$ denote the total number of interactions, $|\mathcal{T}|$ the number of timestamps, ℓ the size of the selected timestamp, and B the batch size. Timestamp selection again requires $O((|\mathcal{I}| - |\mathcal{T}|) \cdot n \log n)$ time for computing timestamp weights via the crossing count. Within the chosen timestamp, edge weights between adjacent interactions are now determined by the same distance metric used in the TSP construction (see Section 6.1.1). Each such metric evaluation compares the participating character sets of two interactions to estimate their dissimilarity, rather than computing (exact) crossings. This incurs $O((\ell - 1) \cdot n^2)$ time in the worst case, as every weight may compare two sets of size $O(n)$ (practically, interaction sets will be much smaller). Sampling the two cuts and reinsertion position is constant-time once these weights are known. Candidate evaluation again dominates with B generated solutions, resulting in a total complexity of

$$O((|\mathcal{I}| - |\mathcal{T}|) \cdot n \log n + (\ell - 1)\, n^2) + B \cdot T_{\text{eval}}.$$

In conclusion, Figure 6.2 visualizes the three different strategies of neighbor generation, which are increasing in both their complexity and their computational effort.

6.2.3 Postprocessing

The methods presented so far all involved heuristics and probabilistic analysis. We now extend these methods with an exact refinement step that eliminates redundant crossings while preserving all timestamp constraints. The procedure relies on a simple yet effective *Postprocessing Rule*, which we describe and justify below. It can be applied repeatedly until no further improvement is possible. It can also be used on the neighborhood generators presented above to optimize the batch candidates before evaluating them using the SA Framework.

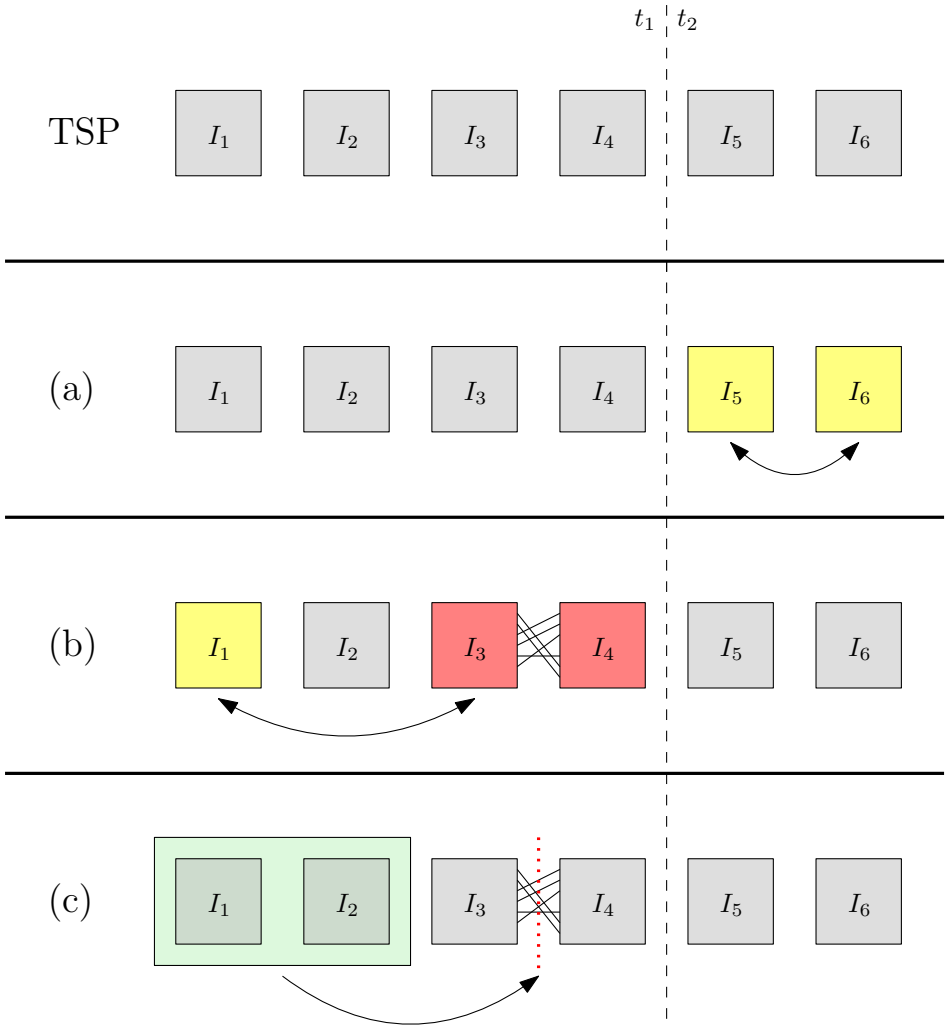


Figure 6.2: Illustration of the three neighborhood generators.
(a) **Random Swap**: two interactions are chosen uniformly at random and exchanged.
(b) **Weight-Based Swap**: a high-crossing adjacent pair is identified (red). One of the two interactions is moved to a 'good' new position (yellow) within the timestamp.
(c) **Group-Based Swap**: a contiguous block is selected (green) and reinserted at a different position while preserving internal order. The dotted line indicates the insertion boundary.

Postprocessing Rule. Fix a timestamp $t \in \mathcal{T}$ and consider a consecutive block of interactions

$$\ldots, P, S_1, S_2, \ldots, S_k, X, N, \ldots \quad (k \geq 1)$$

with all S_i and X belonging to timestamp t, and with P preceding X. Let $\pi(A)$ denote the total order of non-protagonists beneath the protagonist c_P in column A. Let $r := |C_X|$

and define

$$H := \{\text{the first } r \text{ characters beneath } c_P \text{ in } \pi(P)\} \subseteq \mathcal{C} \setminus \{c_P\}.$$

Assume $H = C_X$. Then construct a new interaction $X^\star$ by copying the parent order, $\pi(X^\star) := \pi(P)$, and move $X^\star$ immediately after P within the same timestamp:

$$\ldots, P, S_1, \ldots, S_k, X, N, \cdots \longrightarrow \ldots, P, X^\star, S_1, \ldots, S_k, N, \ldots$$

Since $C_X = H$ occupies the first r positions beneath c_P in $\pi(X^\star)$, this operation preserves validity.

Lemma 1. *Each application of the Postprocessing Rule preserves or improves the global crossing count.*

Proof. Let crossings between adjacent columns A, B be measured by the Kendall-tau distance

$$d(A, B) := K(\pi(A), \pi(B)), \quad \text{and let Cost} := \sum_{\text{adjacent } (A,B)} d(A, B).$$

Performing the described reordering changes the total crossing cost by

$$\Delta = d(P, X^\star) + d(X^\star, S_1) + d(S_k, N) - d(P, S_1) - d(S_k, X) - d(X, N).$$

Because $\pi(X^\star) = \pi(P)$, we have $d(P, X^\star) = 0$ and $d(X^\star, S_1) = d(P, S_1)$. By the triangle inequality of Kendall-tau distance on a fixed ground set,

$$d(S_k, N) \leq d(S_k, X) + d(X, N).$$

Combining these, we obtain $\Delta \leq 0$, showing that the total number of crossings cannot increase. Hence, each application of the Postprocessing Rule preserves or improves the global crossing count. □

Runtime. Let $|\mathcal{I}|$ be the number of interactions, $|\mathcal{T}|$ the number of timestamps, and ℓ_t the number of interactions in timestamp t ($\sum_t \ell_t = |\mathcal{I}|$). Each timestamp requires examining all ordered pairs (P, X) with P preceding X, i.e., $O(\ell_t^2)$ comparisons. For each pair we test $C_X \subseteq C_P$ and verify that C_X matches the prefix beneath c_P in $\pi(P)$, both $O(n)$ with $n = |\mathcal{C}|$. If the rule applies, rebuilding and repositioning X costs $O(n^2 + \ell_t)$ in the worst case, including the bubble-swap movement, where the relocated column is iteratively exchanged with its left neighbor until it reaches the target position. As relocations trigger re-evaluation of affected pairs, the total number of scans can increase proportionally to the number of relocations R. Writing $\ell_{\max} = \max_t \ell_t$, the overall cost until no further relocations occur is

$$O\left((R+1)\,(n \sum_{t \in \mathcal{T}} \ell_t^2) \;+\; R\,(n^2 + \ell_{\max})\right).$$

In practice, most pairs fail early and R is small, but the quadratic timestamp scan dominates on large timestamps.

This rule forms the final refinement stage of the heuristic pipeline. Each application performs a local, guaranteed non-worsening adjustment. However, as it relies on repeated pairwise comparisons of interactions, the procedure is comparatively time-consuming. Its practical efficiency will be evaluated in the next chapter.

CHAPTER 7

Experiment Framework and Evaluation

Having introduced several exact and heuristic approaches for 1-SCM-TI-P, we now evaluate their practical performance. This chapter first outlines the experimental setup, including datasets, parameters, and measurement criteria. We then examine the feasibility of the ILP formulation and compare its results against the algorithmic lower bound introduced in Chapter 5. Afterwards, we evaluate the heuristic methods from Chapter 6, beginning with the TSP-based initial solutions and ending with a detailed comparison of the three Simulated Annealing neighborhood generators in terms of solution quality and runtime.

7.1 Setup

Before presenting the experimental results, we briefly describe the hardware environment, the datasets used for evaluation, and the metrics applied to compare the different methods. All experiments were performed under reproducible conditions to ensure consistency across the exact and heuristic approaches.

7.1.1 Benchmark Environment

All experiments were executed on a desktop workstation equipped with an Intel Core i9-9900K processor (8 cores, 3.60 GHz) and 32 GB of RAM. Gurobi was allowed to utilize up to 16 threads, whereas the heuristic methods were executed single-threaded. Consequently, runtime comparisons should be interpreted qualitatively rather than quantitatively. The heuristic implementations were executed in a Java 21 environment under Windows 11 (64-bit), built and managed using Maven 4.0. The Concorde TSP Solver (v03.12.19) was

executed via the Windows Subsystem for Linux (Ubuntu 22.04), since the solver is natively compiled for Linux.

Integer Linear Programs were solved using Gurobi 12.0.1[1] with default presolve and cutting-plane settings. The TSP instances were solved using the Concorde TSP Solver[2] compiled with non-PIE flags and linked against QSopt[3] for linear programming. Concorde does not support explicit random seed inputs. For the Simulated Annealing optimization, each run was initialized with a distinct fixed random seed.

7.1.2 Testsets

All storyline instances used in this thesis are derived from the DBLP bibliographic database[4], which provides a structured record of computer science publications and their coauthor relations. Because we are more interested in dense publication years rather than a large range of different (sparse) years, we restrict all datasets to publications between 2010 and 2025. We also only consider records of type article (journal publications) or inproceedings (conference publications). From this base, we construct five thematic testsets representing different patterns of academic collaboration.

The first two sets model individual researcher profiles of varying scale. *Junior Researchers* (abbreviated *Juniors*) comprise protagonists with between 10 and 30 publications, representing small to medium-sized collaboration networks. *Senior Professors* (abbreviated *Seniors*) contain prolific authors with more than 100 publications, where only coauthors sharing at least three joint papers with the protagonist are included, in order to reduce the number of rarely contributing non-protagonists. The third category, *High-Frequency Collaborations*, focuses on protagonists with 20 to 50 publications but restricted to their five most frequent collaborators, resulting in densely connected interaction structures. Since the first three categories have a lot of eligible protagonists, we randomly order them and select exactly 50 protagonists from the results. The remaining two sets are community-based: one for the IEEE VIS conference series and one for the Graph Drawing (GD) community. For both, we selected all authors with at least 15 publications in the respective venue and limited each instance to the fifteen most frequent coauthors to preserve readability while maintaining representative interaction patterns. Due to the imposed criteria, all eligible protagonists are taken into consideration. Thus, the VIS conference series includes 46 protagonists, the GD community 33.

All protagonist lists were generated automatically using parameterized SPARQL queries executed against the public DBLP endpoint[5]. Each query defines a distinct filter configuration corresponding to one of the five testset types. The full SPARQL queries used for dataset generation are listed in the Appendix to ensure reproducibility. The

[1] https://www.gurobi.com
[2] http://www.math.uwaterloo.ca/tsp/concorde.html
[3] https://www.math.uwaterloo.ca/~bico/qsopt/
[4] https://dblp.org
[5] https://dblp.org/sparql/

resulting author lists served as input to a data generation tool developed in the context of a related thesis project. This tool accepts a protagonist identifier and a configuration profile, retrieves the relevant coauthor relations from DBLP and outputs a complete storyline instance together with auxiliary metadata such as publication counts and interaction timestamps. It supports batch processing and enforces all constraints defined by the corresponding testset specification (e.g., minimum coauthor frequency, top-k coauthor cap, or publication-type filters). Each generated instance is stored as a *master file* used as direct input for the algorithms evaluated in this work, while accompanying index files summarize key statistics such as the number of characters, interactions, and timestamps.

Table 7.1 summarizes the five testsets, indicating their characteristic ranges of publications and collaborators. Overall, the datasets cover a wide spectrum of structural complexity, from small individual networks to large and highly interconnected collaboration graphs, thereby providing a diverse benchmark for evaluating the proposed methods.

Testset	Pubs	Coauthor Rule	Size	Med. Pubs	Med. Chars
Junior Researchers	10–30	all coauthors	50	14.5	30
Senior Professors	≥100	≥3 joint papers	50	123	49.5
High-Frequency	20–50	top 5 coauthors	50	28.5	5
VIS Conf.	≥15	top 15 coauthors	46	17	15
GD Conf.	≥15	top 15 coauthors	33	22	15

Table 7.1: Overview of the five DBLP testsets. Columns show publication range, coauthor selection rule, testset size (number of instances), and median numbers of publications and characters per protagonist.

7.1.3 Parameter Configuration and Tuning

All algorithms were executed with standardized parameter settings to ensure comparability. For the ILP formulation, we imposed a time limit of 30 minutes per instance. For the heuristic evaluation, we always re-ran instances 5 times to reduce randomness. For the same reason, unless stated otherwise, we always use the median for comparability, emphasizing robustness and minimizing the influence of outlier runs.

With the exception of the TSP evaluation, all subsequent experiments use the Crossing Distance for both the TSP calibration and the Group Swap Neighborhood as presented in Chapter 6.

More calibration was necessary for the SA Framework. In order to find suitable parameters, a *Pilotset* has been created by randomly picking 2 instances from each testset. This subset provided representative variation in size and density while keeping calibration effort low.

Based on preliminary runs, the total number of iterations was fixed to 250, which allowed even the larger *Senior Professor* instances to approach a convergence plateau. The

acceptance probabilities were set to $p_0 = 80\%$ and $p_{\text{end}} = 1\%$ as standard values. For initial temperature calibration, 25 samples were used to estimate the acceptance range. Reducing this number to 10 produced high variance between runs, while increasing to 50 yielded no measurable improvement.

Three neighborhood types were evaluated: *Random*, *Weighted*, and *Group*.

- The *Random* neighborhood required no parametrization.
- For *Weighted*, the enumerated depth was limited to five timestamps and sampling to 40 candidates, balancing runtime and progress speed.
- For *Group*, a group size of 40 candidates proved to be an effective trade-off between local diversity and computational cost, additionally offering good comparability to the *Timestamp* setup.

Neighborhood sizes were deliberately constrained to preserve the stochastic nature of the search and allow non-optimal candidates to remain competitive. With the exception of the *Random* neighborhood, the best candidate in each neighborhood was selected, as described in Section 6.2.1. The postprocessing method was applied to each candidate in both *Weighted* and *Group* modes for local optimization.

7.2 Exact Approaches: ILP and Lower Bound Evaluation

Before turning to the heuristic methods, we briefly assess the practical feasibility of the exact Integer Linear Programming (ILP) formulation introduced in Chapter 5 and compare it with the algorithmic lower bound.

7.2.1 ILP Feasibility

The ILP formulation from Chapter 5 was evaluated on all testsets to establish reference optima. As shown in Figure 7.1, the solver performed well for most categories but revealed clear scalability limits on larger instances.

For *Juniors*, 37 of 50 instances (74%) were solved to optimality, while one instance was infeasible and the remaining unsolved cases formed a geometric-mean final gap of 59.6%. The *High-Frequency Collaborations* set was fully solved, confirming the formulation's ability to handle compact, dense structures. The two community-based sets performed similarly well: for the *VIS* instances, 43 of 46 (93%)

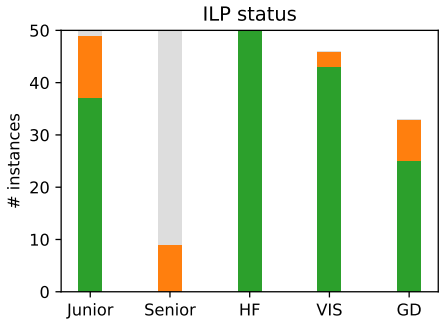


Figure 7.1: Fraction of instances solved to optimality within the 30-minute time limit per testset. The bars are colored green if an optimum was found, orange if at least any valid solution was found, and gray if no feasible solution has been found.

were solved optimally, with the few remaining instances showing a geometric mean gap of 22.6%. For the *GD* set, 25 of 33 (76%) reached optimality, and the remaining eight closed with a geometric-mean gap of 12.4%. Only the *Senior Professor* testset remained out of reach, with no instance solved to optimality and a geometric-mean reported gap (on the 18% where a solution was found) of 74.2%. These results indicate that the ILP formulation scales well for small and medium-sized networks but becomes intractable once the number of interactions exceeds a few hundred. Nevertheless, the obtained optima form a solid benchmark for assessing the heuristic methods in the following sections.

7.2.2 Algorithmic Lower Bound

The algorithmic lower bound introduced in Section 5.3 was evaluated only on instances for which an ILP optimum was available, as the comparison otherwise lacks reference. While the lower bound can be computed in negligible time, its quality proved consistently weak across all testsets. The geometric mean ratios relative to the ILP optima were 14.7% for *Juniors*, 45.9% for *High-Frequency Collaborations*, 22.5% for *VIS*, and 22.9% for *GD*. These values confirm that the current formulation provides little informative value as a tightness guarantee, despite its speed. Consequently, we rely on the ILP optima as reference values in all subsequent evaluations.

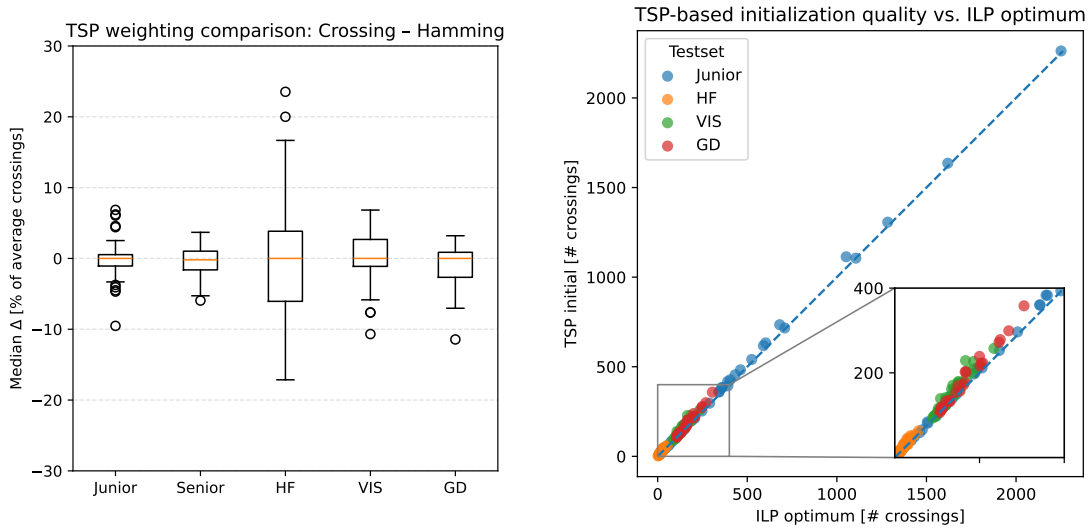
7.3 Heuristic Approaches: TSP-Based Initialization

We now investigate the heuristic methods introduced in Chapter 6. As a first step, we examine the quality of TSP-based initializations and analyze the influence of the underlying distance metric before comparing their results to exact ILP optima, where available.

7.3.1 Distance Metric Comparison

To evaluate the effect of the chosen distance metric on the TSP-based initialization, both the Hamming and the Crossing Distance formulations were applied to every instance. Each instance was solved five times per metric using Concorde, and the resulting orderings were evaluated with respect to their induced number of crossings. Figure 7.2a summarizes the relative difference between both variants, measured as the percentage deviation of the Crossing Distance from the Hamming Distance result for each testset.

Across all datasets, the median difference remains close to zero, indicating that both distance definitions lead to nearly identical (initial) orderings. Slight negative shifts in most testsets suggest a marginal but consistent advantage for the Crossing Distance, while the spread for the *High-Frequency* set reflects its higher sensitivity to small changes in local interaction patterns. Since the Crossing Distance directly models the objective



(a) Relative median differences between Crossing and Hamming Distance formulations across all testsets.

(b) Relation between TSP-based initialization (Crossing Distance) and ILP optima.

Figure 7.2: Evaluation of the TSP-based initialization strategies. (a) Comparison of Hamming and Crossing Distance formulations, showing only marginal differences across testsets. (b) Quality of the Crossing Distance initialization relative to ILP optima, including a zoomed-in region for smaller instances.

function of the storyline problem, we adopt it as the default metric for all subsequent heuristic evaluations.

7.3.2 Comparison to ILP Optima

After establishing the Crossing Distance as the default metric, we now assess the quality of the corresponding TSP-based initializations by comparing them against the ILP optima. Figure 7.2b visualizes this relation across all testsets for which optimal ILP solutions were available. Each point represents a single instance, plotting the number of crossings of the TSP initialization against its optimal counterpart. To improve visibility in the low-crossing region, a zoomed-in view of all instances with fewer than 400 crossings is included.

Across all datasets, the TSP initialization correlates closely with the optimal results. Despite their larger instance sizes, the *Junior* cases show the smallest median deviation from the optimum (below 4%), while the *VIS* and *GD* sets deviate slightly more (around 9% and 7%, respectively). In both *VIS* and *GD*, the deviation increases moderately with instance size, whereas no such trend is observed for *Junior*, indicating structural differences between the datasets. For *High-Frequency*, absolute differences remain small (median $\Delta < 2$ crossings) but appear large in percentage terms due to very low optima.

Testset	Neighborhood	GM (SA/ILP)	Win %	Median Cross.	Median RT [ms]	N
GD	Random	1.006	72.0	154	83	25
	Weighted	**1.001**	92.0	154	205	25
	Group	1.029	24.0	157	985	25
HF	Random	1.055	68.0	20	99	50
	Weighted	**1.012**	92.0	20	282	50
	Group	1.034	68.0	20	694	50
Junior	Random	**1.000**	100.0	342	70	37
	Weighted	**1.000**	100.0	342	170	37
	Group	1.010	51.4	342	826	37
VIS	Random	1.003	81.4	131	71	43
	Weighted	**1.001**	93.0	131	138	43
	Group	1.002	72.1	132	625	43

Table 7.2: Overall Simulated Annealing quality on instances with optimal ILP references (Senior excluded). All values are based on the median run (by crossing) per instance. *GM (SA/ILP)* is the geometric mean of the ratio between the SA and ILP crossing counts (values ≈ 1 indicate equivalence with the ILP optimum). *Win %* is the percentage of instances where the median SA solution equals the ILP optimum. *Med. Cross.* and *Med. RT* denote the median crossing number and median runtime in milliseconds. *N* gives the number of accounted instances per testset.

Results should therefore be interpreted in absolute terms. Not taking this testset into account, across ILP-solvable instances, the TSP initialization remains within 10% above the ILP optimum. In addition to its strong solution quality, the TSP initialization also proved extremely fast. Across all testsets except the *Seniors*, Concorde required between 33 and 134 ms per instance, with median runtimes below 75 ms. Even for the largest *Senior* instances, the solver completed on average after only 453 ms, confirming that the initialization step introduces virtually no computational bottleneck for the overall heuristic pipeline.

7.4 Heuristic Approaches: Simulated Annealing Evaluation

Building upon the TSP-based initializations, we now turn to the Simulated Annealing (SA) framework presented in Chapter 6. This section evaluates the three neighborhood generators (*Random*, *Weighted* and *Group*) in terms of convergence behavior, runtime, and final solution quality. Where available, results are compared against the ILP optima. We first start with another ILP comparison, consequently excluding the *Senior* testset for now.

Neighborhood	Median TSP	Median SA	Δ	Δ%	Median RT [ms]	N
Random	2744	2740	-4	-0.2	782	50
Weighted	2821	2390	-472	-13.7	21478	50
Group	2747	2368	-437	**-14.1**	20311	50

Table 7.3: Senior testset: Simulated Annealing (SA) compared to the TSP initialization (Crossing Distance metric). All values are medians over instances' median runs. Δ = SA − Init (negative values indicate improvement).

7.4.1 Overall SA Quality

Table 7.2 shows that across all ILP-solvable sets, SA reliably reaches near-optimal quality. *Weighted* dominates overall: it achieves the best geometric-mean ratio in every testset (GD 1.001, HF 1.012, Junior 1.000, VIS 1.001) while keeping runtime modest (median 138-282 ms), and it wins on 92-100% of instances. *Random* is surprisingly competitive on small and regular instances, reflecting the strength of the TSP-based initialization. It leads in the runtime comparison by far, being about 2-3 times faster than the weighted approach, and 7-12 times faster than the group approach. *Group* is consistently slower (median 625-985 ms) and does not translate its higher per-iteration effort into better medians on these mid-sized sets. Taken together, *Weighted* offers the most attractive quality-runtime trade-off for the instance scales where ILP is available.

Having verified that SA achieves near-optimal results on ILP-solvable instances, we now turn to the larger and more complex *Senior* testset, where no exact reference is available. In this case, the comparison focuses on relative quality among neighborhoods and on runtime behavior, using the TSP-based initialization as the baseline.

Across the large *Senior* instances, both the *Weighted* and *Group* neighborhoods achieve substantial improvements over the TSP initialization, reducing the median number of crossings by roughly 14%. Interestingly, the runtime gap observed on smaller instances disappears: *Group* achieves comparable improvements at only slightly lower computational cost (median 20 s vs. 21 s for *Weighted*). In contrast, the *Random* neighborhood stagnates near the initialization quality, indicating that random local perturbations alone are insufficient to escape the large plateaus typical for dense collaboration structures.

Overall, these results demonstrate that the more structured neighborhoods are effective even on large-scale, high-density instances, confirming scalability of the more sophisticated neighborhood generators.

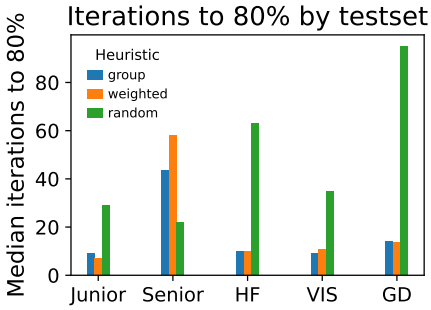
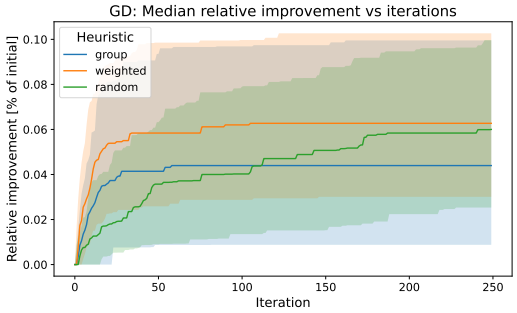
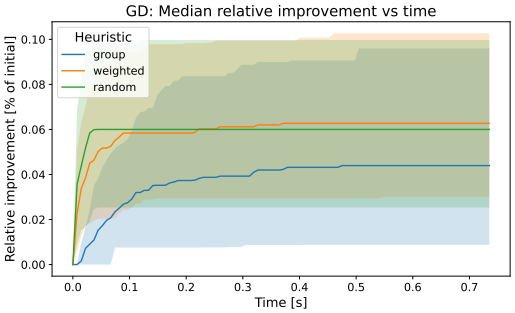


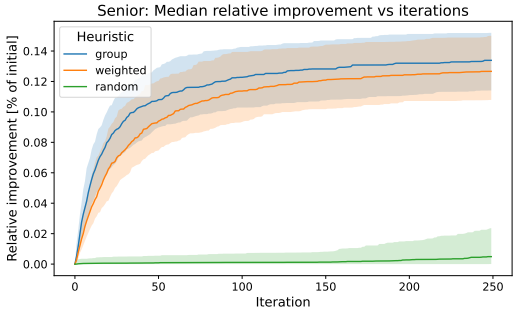
Figure 7.3: Median iteration index at which each neighborhood reaches 80% of its total improvement in crossing reduction. On the *Senior* testset, *Random* appears faster only due to its minimal total improvement (see Table 7.3).



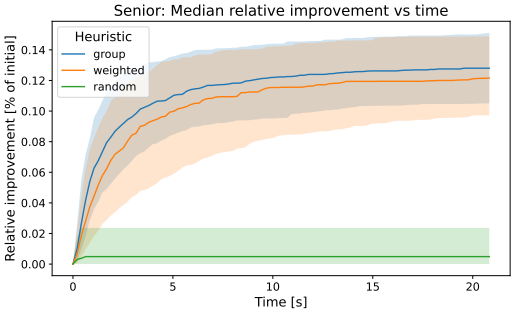
(a) GD - crossing count over iterations.



(b) GD - crossing count over runtime.



(c) Senior - crossing count over iterations.



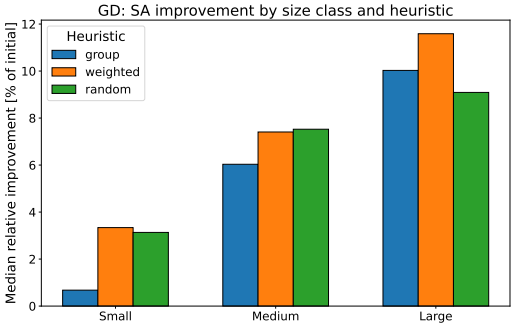
(d) Senior - crossing count over runtime.

Figure 7.4: Convergence trajectories of Simulated Annealing across neighborhoods. The shaded regions represent the interquartile range (25th - 75th percentile) of relative improvements across all runs for each heuristic. Top: mid-sized GD instances. Bottom: large-scale Senior instances. The structured neighborhoods (*Weighted* and *Group*) achieve steep early improvements and maintain steady improvement. *Random* runs through all 250 iterations quickly, but fails to show significant improvements in large instances. Note that for the runtime plot, the x-axis is truncated at the 75th percentile of runtime for readability.

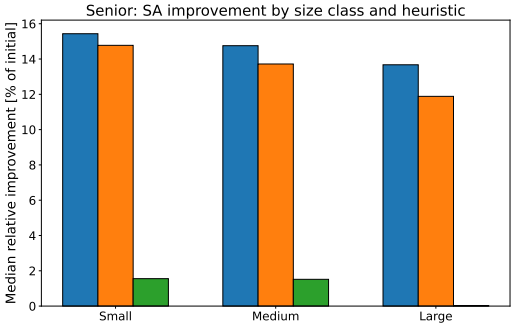
However, on small instances, the simplest and quickest neighborhood generator offers solutions without significant quality loss. To further understand these quality differences, we next investigate the convergence behavior and runtime progression of the three neighborhood strategies.

7.4.2 Convergence and Runtime Behavior

Looking beyond the final solution quality, it is of interest to examine the convergence dynamics of the three Simulated Annealing variants. This analysis (shown in Figure 7.3) provides insight into how quickly each neighborhood generator progresses toward a stable configuration and how efficiently it utilizes its computational budget. To that end, we



(a) GD testset ($n = 33$): median relative improvement by size quantile.



(b) Senior testset ($n = 50$): median relative improvement by size quantile.

Figure 7.5: Median relative improvement of each neighborhood over the TSP initialization, grouped by instance-size percentiles (33rd and 66th percentile cutoffs). Improvement decreases consistently with increasing instance size, especially for the *Random* neighborhood, which fails to provide any meaningful progress on the largest *Senior* instances. In contrast, the *Group* neighborhood performs poorly on small instances but gains relative strength as instance complexity increases.

track both the crossing-count trajectories over the 250 iterations and their corresponding runtime progress.

Across all datasets, the *Weighted* and *Group* neighborhoods exhibit steep early improvements, typically reaching 80% of their final quality within the first 10-15 iterations. Before investigating *Random*, we recall that this neighborhood does not pick the best candidate from a batch, but instead generates a singleton candidate per iteration. In contrast to the other generators, *Random* requires substantially more iterations (30-90) to reach the same relative threshold. The only exception is the *Senior* testset, where *Random* reaches the 80% mark after just 22 iterations. However, this is a consequence of its very small overall improvement. By contrast, *Weighted* and *Group* continue to refine the layout for much longer and finish at markedly lower crossing counts (compare Table 7.3).

We now investigate two testsets in more detail. Given their higher complexity and density, we focus on the *GD* and *Senior* testsets. In both cases, the iteration trajectories in Figure 7.4 reveal the differing convergence behavior of the three neighborhoods. The *Weighted* heuristic consistently exhibits steep early improvements, achieving a balance between runtime and quality. By contrast, the *Random* generator runs through all 250 iterations quickly, yet fails to reach a stable plateau, indicating a limited ability to refine layouts in complex instances. Finally, in the large *Senior* instances, the *Group* neighborhood continues to improve even in later iterations, surpassing the *Weighted* generator and demonstrating its advantage in navigating dense collaboration structures. The runtime trajectories mirror these findings, with *Random* completing all iterations early and the structured neighborhoods maintaining gradual but consistent refinement.

While the convergence trajectories illustrate temporal dynamics, it is also helpful to analyze how the achievable improvement varies with instance size. To illustrate this, each testset was partitioned into three equally sized instance-size groups using percentile cutoffs at the 33rd and 66th percentiles, and the median relative improvement of each neighborhood was computed within these groups. Figure 7.5 shows the results for the *GD* and *Senior* testsets.

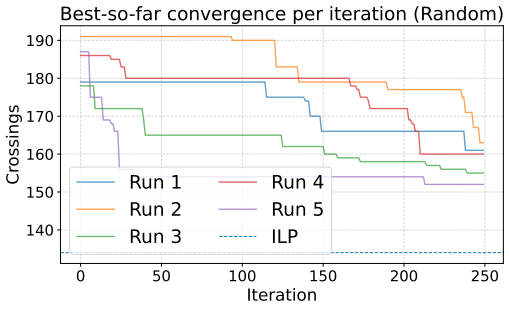
The figure confirms that, for the *Senior* testset, the obtainable relative improvement decreases with instance size, reflecting the increasing difficulty of escaping local minima in dense collaboration structures. The *Random* neighborhood's performance deteriorates sharply for the largest instances, where it often terminates with no significant improvement at all. Conversely, *Group* shows the opposite pattern. It is less effective on small, simple cases but increasingly competitive as instance complexity rises. These observations highlight that structured neighborhood operations scale more robustly with instance size, whereas simple random perturbations fail to adapt to growing combinatorial complexity.

Overall, the convergence analysis confirms the earlier quantitative trends: structured neighborhoods achieve rapid and sustained improvement, whereas purely random moves yield quick but shallow gains once instances get large. These results further emphasize that the additional computational cost of *Weighted* and *Group* directly translates into better solution quality, particularly for large, high-density instances. While both structured neighborhoods scale well, *Weighted* performs slightly better on small and medium instances, whereas *Group* gains a relative edge once interaction density becomes very high.

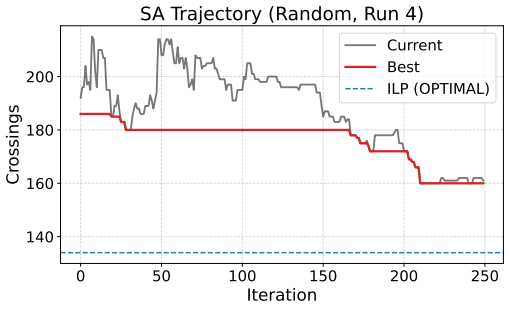
Last, we briefly examine the runtime composition of these methods, highlighting the share of time consumed by postprocessing. To clarify where the Simulated Annealing budget is spent, we decomposed total runtime into search time, neighborhood generation, and (within neighborhood generation) postprocessing. For *Weighted* and *Group*, the search phase accounts for the vast majority of total time (typically $85 - 95\%$), and within search, neighborhood generation contributes about 90%. Postprocessing then constitutes roughly $40 - 55\%$ of neighborhood time on small and mid-sized sets (Junior, VIS, GD) and rises to nearly 56% on the largest *Senior* instances, corresponding to about $31 - 49\%$ of the overall runtime depending on dataset scale. By contrast, *Random* performs no postprocessing and spends a smaller share in search overall, consistent with its single-candidate iterations. These measurements confirm that the local postprocessing step (introduced in Section 6.2.3) forms the principal cost driver for the structured neighborhoods.

7.5 Case Study: A Signature Professor

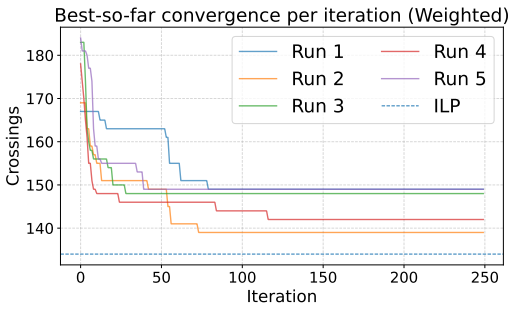
To complement the analysis presented so far, we now take a closer look at a single, representative instance. This "signature" example illustrates how the proposed methods behave on a concrete storyline and provides an intuitive understanding of the optimization dynamics beyond statistical averages. For this purpose, we select **Stephen G. Kobourov**,



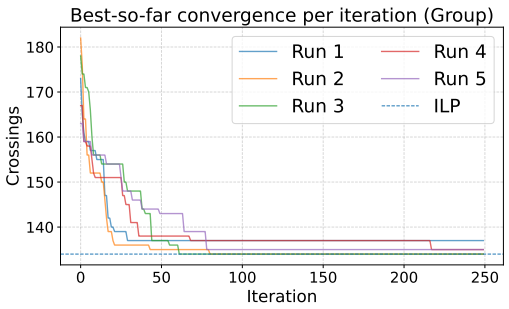
(a) *Random*: best-so-far trajectories across five runs.



(b) *Random*: current vs. best in run 4.



(c) *Weighted*: best-so-far trajectories across five runs.



(d) *Group*: best-so-far trajectories across five runs.

Figure 7.6: Convergence curves across five Simulated Annealing runs for each neighborhood on the signature instance (Stephen G. Kobourov). The top-left plot shows the *Random* neighborhood's best-so-far trajectories, while the top-right panel visualizes a single run's current vs. best trace, highlighting classical SA behavior. The bottom row compares the structured *Weighted* and *Group* neighborhoods, which converge more tightly around the ILP optimum (134).

a long-standing contributor to the VIS and GD communities, thus fitting perfectly into our presented evaluation structure.

Restricting the data to the years 2020 to 2025 and including only coauthors with at least five joint publications yields an instance of 39 publications and 10 recurring collaborators, with an average set-strength of 7.95 shared papers per coauthor. This configuration represents a quite dense structure and serves as a challenging test case for the Simulated Annealing heuristics.

An ILP result is available for this instance, though its computation took a total of 179 minutes. As a remarkable side-note, the ILP found the optimal heuristic solution after just 9.25 minutes; the remaining time was spent on matching the lower bound with this (optimal) solution.

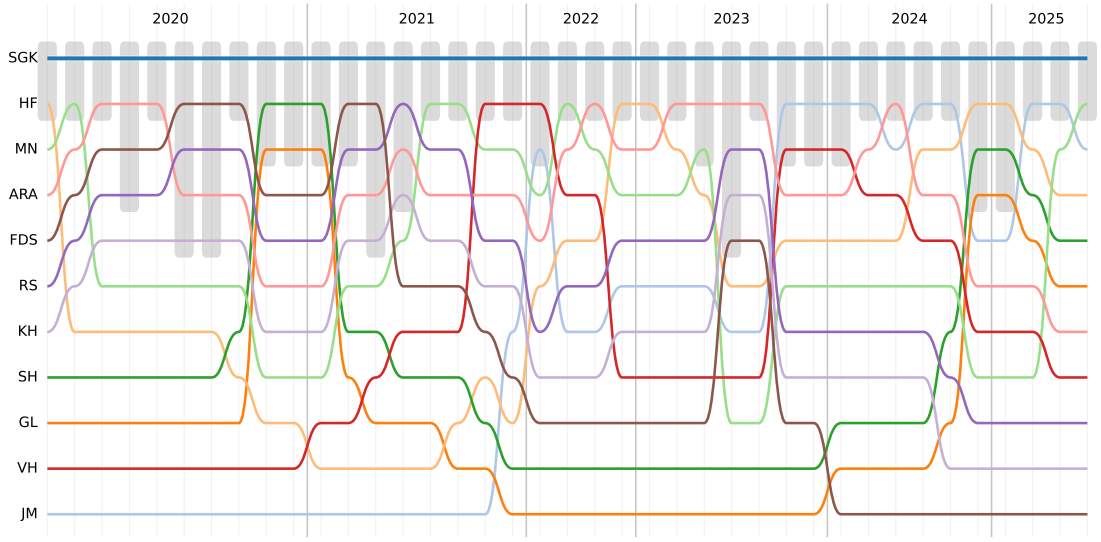
The ILP optimum of 134 crossings serves as a clear performance reference for all heuristics. Across the five Simulated Annealing runs per neighborhood, the *Group* and *Weighted* variants consistently approach this value within one to two crossings, while *Random* remains roughly 20% above optimum in the median case. The convergence curves in Figure 7.6 demonstrate this visually: all structured variants show steep early progress and quickly stabilize around the optimum, whereas the unstructured *Random* variant oscillates for longer and plateaus prematurely.

Together, these results reinforce the findings from the broader benchmark: the TSP-based initialization already provides a strong baseline, but meaningful additional improvements require structured neighborhoods that exploit the problem's temporal and interaction-based structure. Even on a dense, real-world instance such as the Kobourov dataset, the *Weighted* and *Group* heuristics achieve results indistinguishable from the ILP optimum at a fraction of the computational cost. The resulting layouts are visualized in Figures 7.7 and 7.8, illustrating the progression from the TSP-based initialization to the final heuristic solutions.

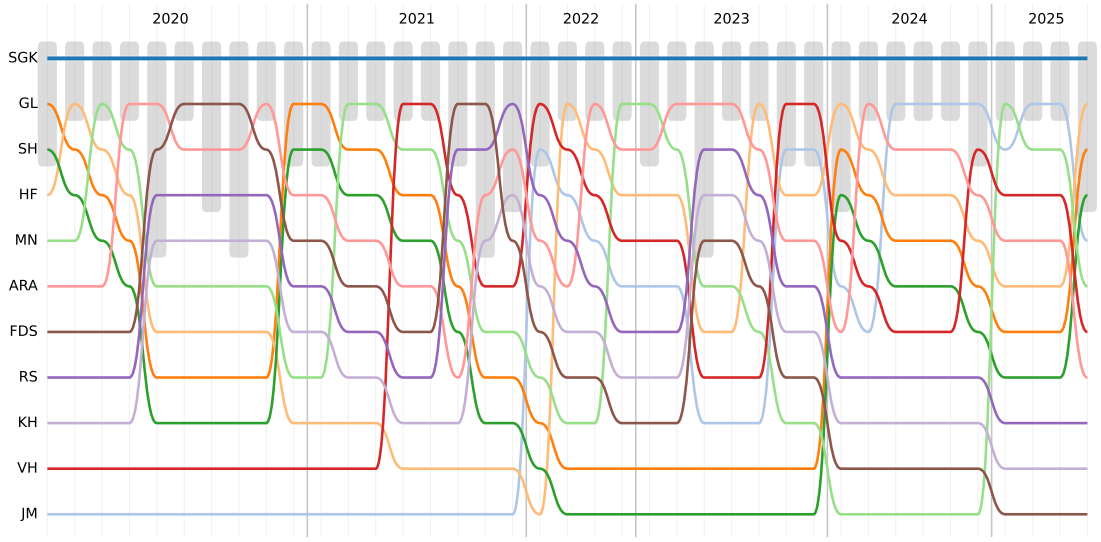
This single-case analysis thus complements the statistical evaluation by offering an intuitive confirmation of stability and solution behavior on a dense real-world instance. It also highlights how the heuristic framework balances speed and quality, reaching near-optimal layouts within seconds, while the ILP demands several hours. The observed convergence patterns conclude the experimental evaluation, which has shown that the developed heuristic framework produces solutions of consistently high quality across a broad spectrum of instance scales.

7.6 Summary of Findings

The experimental evaluation has demonstrated that the proposed framework reliably delivers high-quality solutions across all instance scales. The ILP formulation establishes solid reference optima for small and medium-sized storylines, while the heuristic pipeline (rooted in the TSP-based initialization and refined through Simulated Annealing) extends this performance to much larger and denser instances. Among the evaluated neighborhoods, *Weighted* consistently offers the best balance between runtime and quality, with *Group* providing additional gains on large and more interconnected structures. The detailed case study confirms that these heuristics closely match ILP-level quality at a fraction of the computational cost. Overall, the results underline both the scalability and robustness of the developed heuristic approach, forming a strong empirical foundation for the upcoming discussion.

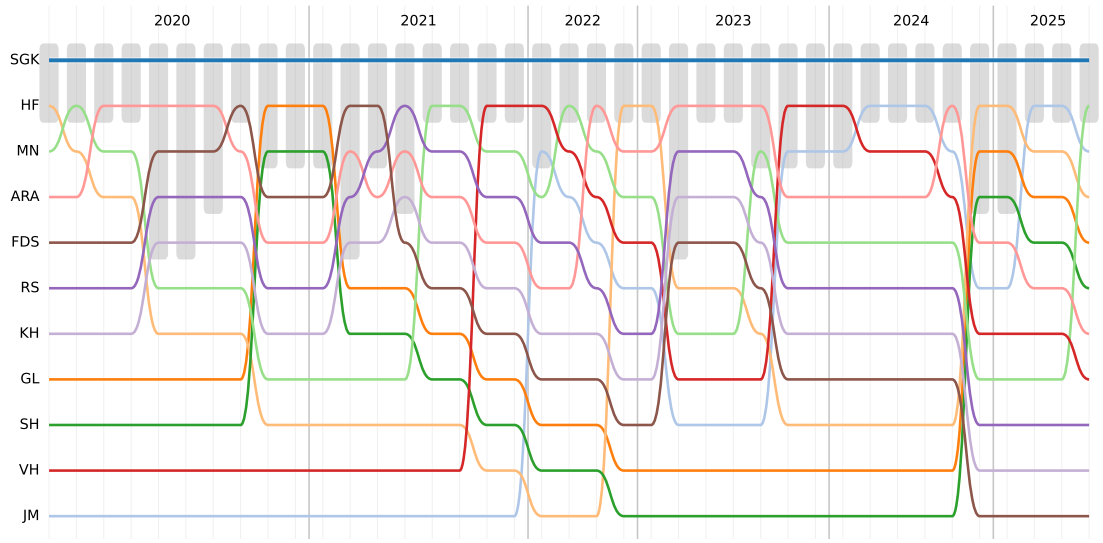


(a) ILP optimum: 134 crossings.

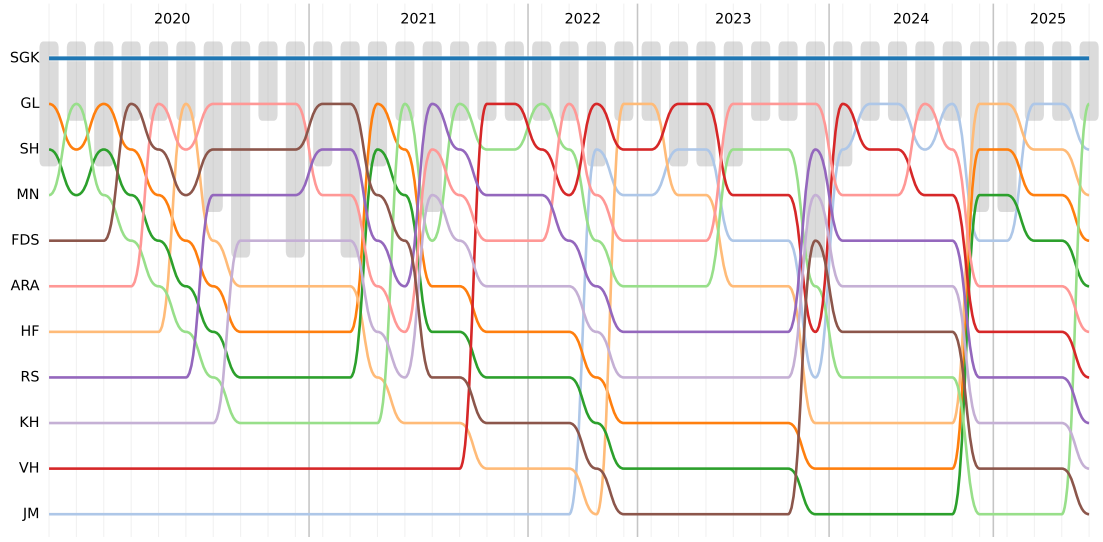


(b) TSP-based initial layout: 184 crossings.

Figure 7.7: Optimal and initial layouts for the signature instance (Stephen G. Kobourov). The ILP optimum represents the best achievable configuration with 134 crossings, while the TSP-based initialization serves as a strong but visibly less organized starting point with 184 crossings.



(a) *Group* heuristic (median run): 134 crossings.



(b) *Random* heuristic (median run): 160 crossings.

Figure 7.8: Heuristic layouts (median runs) for the same instance. The *Group* neighborhood reaches the ILP optimum with 134 crossings, though its layout differs visually from the ILP configuration. The unstructured *Random* variant remains at 160 crossings, resulting in noticeably more visual clutter. The omitted *Weighted* neighborhood performed comparably to *Group* (142 crossings).

CHAPTER 8

Discussion

In this chapter, the experimental findings are interpreted to explain why the methods behave as they do and when each approach is preferable. The discussion contrasts exact and heuristic strategies, relates their performance to structural properties of the datasets, and reflects on scalability, limitations, and generalization. The aim is to connect the empirical evidence from Chapter 7 with the problem structure defined in Chapter 4 and the algorithmic designs presented in Chapters 5 and 6.

8.1 Trade-offs Between Exact and Heuristic Methods

TSP and ILP. The previous results show that even without the Simulated Annealing framework, the TSP-based initialization alone yields solutions close to the optimal layouts, particularly in less complex instances. As the primary goal of storyline visualizations is to produce clear and readable narratives rather than exact optima, a small deviation in crossing count is an acceptable trade-off for the drastic runtime reduction. Consequently, the TSP initialization alone represents a valid option when rapid, visually coherent results are desired.

The ILP formulation achieved satisfactory results even on medium-complex instances within reasonable time. Interestingly, its performance did not correlate directly with instance size: large but sparsely connected networks were often solved efficiently, while smaller yet denser instances caused the solver to stagnate. This suggests that the practical difficulty of the ILP is driven less by the number of characters or interactions and more by structural density and interconnection, which increase the number of conflicting constraints and hinder convergence.

Simulated Annealing. For small and moderately sized instances without strong structural density, the *Random* neighborhood frequently reaches the ILP optimum. In

such cases, the simplicity and minimal runtime of this generator make it the most efficient choice for quick visualizations where further refinement would yield only marginal gains.

Once instance complexity increases, a more structured search strategy becomes necessary. Both the *Weighted* and *Group* neighborhoods outperform the *Random* generator under such conditions. Conversely, for simple instances, the *Group* generator often fails to reach the optimum, as it primarily targets timestamps with multiple interactions. Similarly, the *Weighted* approach may overlook small timestamps with few crossings, since its weighted selection favors timestamps with higher crossing counts. These patterns highlight that both structured generators are tuned for dense, complex configurations rather than for simple or sparse ones.

Given the short runtime of the *Random* generator, the described problems could be addressed using a combined approach: if the given instance is large, it first is pre-solved using the *Weighted* or *Group* generator. Afterwards, the random generator tries to enhance an already almost perfect layout, though when doing so, temperatures for Simulated Annealing must be chosen carefully so as not to irreversibly escape the local optima found by the first generator run. Another idea for easily gaining broader diversity is to populate the neighborhood of the more sophisticated generators with some purely random solutions in each generation, making use of the greedy behavior of the neighborhood. This concept is already partially implemented in the well-performing *Weighted* approach, which mixes random and weighted swaps within its selected timestamp. Extending this approach to include other timestamps as well might already allow solutions to constantly be on par with the *Random* generator.

Last, considering the large runtime of the structured neighborhood generators, the exhaustive usage of the postprocessing procedure may need to be revisited. As it consumes the large majority of the neighborhood runtime ($30 - 50\%$, depending on the testset scale), an easy runtime decrease is gained by not individually performing the procedure on all neighborhood candidates before evaluation, but instead first choosing an evaluation candidate, then postprocessing only that winning candidate alone. The so-gained increase in randomness and potential slight worsening of candidate solutions can be countered by adjusting the temperature of the SA accordingly.

8.2 Structure and Scalability

As implied by the ILP results, the main driver of complexity is not the overall number of interactions. Even without cross-timestamp interactions, optimizing within a single timestamp already imposes its own challenge. On the other hand, the stricter the partial order becomes, the less work remains for our algorithms, especially since once the total order is fixed, the problem is essentially solved. For the ILP, this effect can be seen directly in the constraints: for each interaction, we need as many assignment variables as there are other interactions, quickly and drastically increasing the total number of required variables. Sparsity, on the other hand, allows nearly independent local orders. When protagonists rarely collaborate with the same co-authors or attend overlapping

venues, optimization becomes much easier, as the independence between timestamps increases.

This pattern appears both in the signature instance and across the large testsets. Once collaborations between the protagonist and other characters increase, the *Group* strategy becomes particularly effective in moving sets of bundled interactions without destroying their internal structure. The *Weighted* strategy, in contrast, excels when crossings are concentrated in multiple, separate timestamps. Given these observations, structural awareness and informed decisions on which heuristic to choose may provide a stronger improvement than simply increasing the number of iterations.

Finally, we have seen that even large instances can be solved within seconds by our heuristic approaches. However, while the true optima of these large instances remain unknown, the TSP becomes significantly less sufficient as instance complexity grows. From the initial TSP solution, our heuristics achieved further improvements of about 10–15%, showing that the initialization alone no longer captures the structural depth of complex cases. Still, layouts with several thousand crossings can be optimized by the proposed methods without major adaptions, making the framework suitable for most applications where the result is intended for visual presentation. Nevertheless, the hidden structural properties of the observed patterns limit the generalization of such a statement.

8.3 Limitations

With all the insights given so far, it is important to clarify the limits of their interpretation. First, the presented parameter settings were derived from a pilot set through manual calibration rather than formal tuning. While they proved stable across datasets, this informal process leaves open the question of optimality for other instance families. Additionally, some parameters could be analyzed even further. This is especially true for the postprocessing procedure, whose practical effect was not clearly separated from the performance of the pure heuristics.

Second, the *Lower Bound Algorithm* presented in Section 5.3 provides little practical value for assessing absolute solution quality, even for instances with few total characters. Quality loss further increases with more characters involved, making the presented algorithm insufficient for its primary goal, which was to provide an estimate when the ILP becomes intractable. The main source of this limitation is that the algorithm neglects necessary relations between characters, resulting in a vast underestimation for practical applications. Stronger theoretical bounds or problem relaxations could lead to more usable lower bounds for the investigated problem.

Another limitation lies in the missing formal structural characterization of instances. The current framework does not yet identify the metrics that best describe interaction density or temporal overlap, making the choice of heuristic largely empirical. Establishing such indicators would allow future work to predict suitable strategies without extensive trial runs and could also allow further claims across different dataset families.

This thesis treated the adapted SCMP algorithm by Hegemann et al. [HW24] as a black box and did not investigate potential efficiency gains from reusing intermediate results between successive Simulated Annealing generations. Such reuse could reduce redundant computations, especially in the structured neighborhoods, and might further improve runtime without affecting solution quality.

Finally, while the observed behavior suggests combinatorial complexity, the theoretical status of the 1-SCM-TI-P remains unresolved, as it is currently unknown whether the problem is NP-hard.

Despite these limitations, the experiments provide a robust empirical foundation, showing that the 1-SCM-TI-P admits efficient and high-quality solutions even for large problem instances.

CHAPTER 9

Conclusion and Outlook

This thesis introduced and evaluated a complete optimization framework for 1-SCM-TI-P. We started by formulating the problem mathematically, constructing an ILP baseline for exact comparisons. We also proposed a computationally lightweight but loose lower bound algorithm, which systematically underestimates crossings that must occur in any feasible layout.

We then investigated heuristic approaches, starting with an already very effective TSP-solution that within short time produces satisfying solutions for small- to medium-complex instances. To iteratively improve upon these initial solutions, we introduced three neighborhood generators in combination with a Simulated Annealing framework, which greatly differed in complexity and runtime. While all three neighborhoods performed well on instances up to medium complexity, reaching near-optimum solutions in milliseconds, on large instances the more structured generators had the advantage against a naive, pure Simulated Annealing approach.

During evaluation, we showed that the problem structure cannot simply be measured by the total number of interactions. Evidence suggests that a combination of interactions per timestamp and the recurrence of collaborations drive instance complexity, though more formal analysis is required to confirm this.

Future work should thus focus on predictive heuristic selection and on extending the framework to more general storyline variants. In addition, formal parameter calibration may reveal unused potential in the current setup.

Last, it is still an open problem whether 1-SCM-TI-P is NP-hard. While it is known that for a fixed total ordering, efficient solutions are available, the question if the current relaxation already makes the problem intractable remains.

Appendix: SPARQL Queries for Dataset Generation

Each query below defines one of the five dataset testsets described in Chapter 7. The queries can be run on the DBLP SPARQL webpage at `https://sparql.dblp.org`.

Junior Researchers

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dblp: <https://dblp.org/rdf/schema#>

SELECT (STRAFTER(str(?author), "/pid/") AS ?pid)
WHERE {
  VALUES ?type { dblp:Article dblp:Inproceedings }
  ?pub a ?type ;
       dblp:authoredBy ?author ;
       dblp:yearOfPublication ?y .
  ?author a dblp:Person .

  FILTER (?y >= "2010"^^xsd:gYear && ?y <= "2025"^^xsd:gYear)
  FILTER NOT EXISTS { ?pub dblp:isVersionOf ?vr . }

  FILTER EXISTS {
    ?pub dblp:authoredBy ?co .
    FILTER (?co != ?author)
  }
}
GROUP BY ?author
HAVING (COUNT(DISTINCT ?pub) >= 10 && COUNT(DISTINCT ?pub) <= 30)
ORDER BY RAND()
LIMIT 50
```

Senior Researchers

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dblp: <https://dblp.org/rdf/schema#>

SELECT (STRAFTER(str(?author), "/pid/") AS ?pid)
WHERE {
  VALUES ?type { dblp:Article dblp:Inproceedings }

  { SELECT ?pub WHERE { ?pub dblp:authoredBy ?a }
    GROUP BY ?pub HAVING (COUNT(DISTINCT ?a) >= 2) }

  ?pub a ?type ;
       dblp:authoredBy ?author ;
       dblp:yearOfPublication ?y .
  ?author a dblp:Person .

  FILTER (?y >= "2010"^^xsd:gYear && ?y <= "2025"^^xsd:gYear)
  FILTER NOT EXISTS { ?pub dblp:isVersionOf ?vr }
}
GROUP BY ?author
HAVING (COUNT(DISTINCT ?pub) >= 100)
ORDER BY RAND()
LIMIT 50
```

High Frequency Collaborators

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dblp: <https://dblp.org/rdf/schema#>

SELECT (STRAFTER(str(?author), "/pid/") AS ?pid)
WHERE {
  VALUES ?type { dblp:Article dblp:Inproceedings }

  { SELECT ?pub WHERE { ?pub dblp:authoredBy ?a }
    GROUP BY ?pub HAVING (COUNT(DISTINCT ?a) >= 2) }

  ?pub a ?type ;
       dblp:authoredBy ?author ;
       dblp:yearOfPublication ?y .
  ?author a dblp:Person .

  FILTER (?y >= "2010"^^xsd:gYear && ?y <= "2025"^^xsd:gYear)
  FILTER NOT EXISTS { ?pub dblp:isVersionOf ?vr }
}
GROUP BY ?author
HAVING (COUNT(DISTINCT ?pub) >= 20 && COUNT(DISTINCT ?pub) <= 50)
ORDER BY RAND()
LIMIT 50
```

VIS Community

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dblp: <https://dblp.org/rdf/schema#>

SELECT (STRAFTER(str(?author), "/pid/") AS ?pid)
WHERE {
  VALUES ?stream { <https://dblp.org/streams/conf/visualization> }

  { SELECT ?pub WHERE { ?pub dblp:authoredBy ?a }
    GROUP BY ?pub HAVING (COUNT(DISTINCT ?a) >= 2) }

  ?pub dblp:publishedInStream ?stream ;
       dblp:authoredBy ?author ;
       dblp:yearOfPublication ?y .
  ?author a dblp:Person .

  FILTER (?y >= "2010"^^xsd:gYear && ?y <= "2025"^^xsd:gYear)
  FILTER NOT EXISTS { ?pub dblp:isVersionOf ?vr }
}
GROUP BY ?author
HAVING (COUNT(DISTINCT ?pub) >= 15)
ORDER BY RAND()
LIMIT 50
```

Graph Drawing Community

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dblp: <https://dblp.org/rdf/schema#>

SELECT (STRAFTER(str(?author), "/pid/") AS ?pid)
WHERE {
  VALUES ?gdStream { <https://dblp.org/streams/conf/gd> }

  { SELECT ?pub WHERE { ?pub dblp:authoredBy ?a }
    GROUP BY ?pub HAVING (COUNT(DISTINCT ?a) >= 2) }

  ?pub dblp:publishedInStream ?gdStream ;
       dblp:authoredBy ?author ;
       dblp:yearOfPublication ?y .
  ?author a dblp:Person .

  FILTER (?y >= "2010"^^xsd:gYear && ?y <= "2025"^^xsd:gYear)
  FILTER NOT EXISTS { ?pub dblp:isVersionOf ?vr }
}
GROUP BY ?author
HAVING (COUNT(DISTINCT ?pub) >= 15)
ORDER BY RAND()
LIMIT 50
```

Overview of Generative AI Tools Used

GitHub Copilot was used during software development for code completion suggestions only. Additionally, a language model was consulted to support plotting tasks (e.g., axis calibration and visual adjustments) and to provide non-textual feedback on clarity and consistency during proofreading. All conceptual work, implementation decisions, and the final written content are my own.

List of Figures

List of Tables

Bibliography

[AB14] Dustin Lockhart Arendt and Leslie M. Blaha. "SVEN: Informative Visual Representation of Complex Dynamic Structure". In: *CoRR* abs/1412.6706 (2014). DOI: 10.48550/arXiv.1412.6706.

[Dij+16] Thomas C. van Dijk et al. "Block Crossings in Storyline Visualizations". In: *Proc. 24th International Symposium on Graph Drawing and Network Visualization (GD)*. Ed. by Yifan Hu and Martin Nöllenburg. Vol. 9801. LNCS. Springer, 2016, pp. 382–398. DOI: 10.1007/978-3-319-50106-2.

[Dob+23] Alexander Dobler et al. "Crossing Minimization in Time Interval Storylines". In: *CoRR* abs/2302.14213 (2023). DOI: 10.48550/ARXIV.2302.14213.

[Gro+16] Martin Gronemann et al. "Crossing Minimization in Storyline Visualization". In: *Proc. 24th International Symposium on Graph Drawing and Network Visualization (GD)*. Ed. by Yifan Hu and Martin Nöllenburg. Vol. 9801. LNCS. Springer, 2016, pp. 367–381. DOI: 10.1007/978-3-319-50106-2.

[HW24] Tim Hegemann and Alexander Wolff. "Storylines with a Protagonist". In: *Proc. 32nd International Symposium on Graph Drawing and Network Visualization (GD)*. Ed. by Stefan Felsner and Karsten Klein. Vol. 320. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 26:1–26:22. DOI: 10.4230/LIPICS.GD.2024.26.

[JV83] Roy Jonker and Ton Volgenant. "Transforming asymmetric into symmetric traveling salesman problems". In: *Operations Research Letters* 2.4 (1983), pp. 161–163. ISSN: 0167-6377. DOI: 10.1016/0167-6377(83)90048-2.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by Simulated Annealing". In: *Science* 220.4598 (1983), pp. 671–680. DOI: 10.1126/science.220.4598.671.

[Kos+15] Irina Kostitsyna et al. "On Minimizing Crossings in Storyline Visualizations". In: *Proc. 23rd International Symposium on Graph Drawing and Network Visualization (GD)*. Ed. by Emilio Di Giacomo and Anna Lubiw. Vol. 9411. LNCS. Springer, 2015, pp. 192–198. DOI: 10.1007/978-3-319-27261-0.

[KLM25] Yun-Hsin Kuo, Dongyu Liu, and Kwan-Liu Ma. "SpreadLine: Visualizing Egocentric Dynamic Influence". In: *IEEE Trans. Vis. Comput. Graph.* 31.1 (2025), pp. 1050–1060. DOI: 10.1109/TVCG.2024.3456373.

[Law95] Eugene L Lawler. *The traveling salesman problem : a guided tour of combinatorial optimization*. eng. Reprint. Wiley-Interscience series in discrete mathematics and optimization. Chichester [u.a.]: Wiley, 1995. ISBN: 0471904139.

[Mue+13] Chris Muelder et al. "Egocentric storylines for visual analysis of large dynamic graphs". In: *Proc. IEEE International Conference on Big Data (BigData 2013)*. Ed. by Xiaohua Hu et al. IEEE Computer Society, 2013, pp. 56–62. DOI: 10.1109/BIGDATA.2013.6691715.

[Mun09] Randall Munroe. *Movie Narrative Charts*. https://xkcd.com/657/. Accessed: 2025-10-28. 2009.

[NW88] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley interscience series in discrete mathematics and optimization. Wiley, 1988. ISBN: 978-0-471-82819-8. DOI: 10.1002/9781118627372.

[OM10] Michael Ogawa and Kwan-Liu Ma. "Software evolution storylines". In: *Proc. ACM Symposium on Software Visualization (SOFTVIS 2010)*. Ed. by Alexandru C. Telea, Carsten Görg, and Steven P. Reiss. ACM, 2010, pp. 35–42. DOI: 10.1145/1879211.1879219.

[TM12] Yuzuru Tanahashi and Kwan-Liu Ma. "Design Considerations for Optimizing Storyline Visualizations". In: *IEEE Trans. Vis. Comput. Graph.* 18.12 (2012), pp. 2679–2688. DOI: 10.1109/TVCG.2012.212.

[Tri08] Michael A. Trick. "David L. Applegate, Robert E. Bixby, Vasek Chvátal , William J. Cook. The Traveling Salesman Problem: A Computational Study, Princeton University Press, Princeton, 2007, ISBN-13: 978-0-691-12993-8, 606 pp". In: *Oper. Res. Lett.* 36.2 (2008), pp. 276–277. DOI: 10.1016/J.ORL.2007.06.002.