Revisiting ILP Models for Exact Crossing Minimization in Storyline Drawings

Alexander Dobler
TU Wien, Austria

Michael Jünger
University of Cologne, Germany

Paul J. Jünger
University of Bonn, Germany

Julian Meffert
University of Bonn, Germany

Petra Mutzel
University of Bonn, Germany

Martin Nöllenburg
TU Wien, Austria

Abstract

Storyline drawings are a popular visualization of interactions of a set of characters over time, e.g., to show participants of scenes in a book or movie. Characters are represented as x-monotone curves that converge vertically for interactions and diverge otherwise. Combinatorially, the task of computing storyline drawings reduces to finding a sequence of permutations of the character curves for the different time points, with the primary objective being crossing minimization of the induced character trajectories. In this paper, we revisit exact integer linear programming (ILP) approaches for this NP-hard problem. By enriching previous formulations with additional problem-specific insights and new heuristics, we obtain exact solutions for an extended new benchmark set of larger and more complex instances than had been used before. Our experiments show that our enriched formulations lead to better performing algorithms when compared to state-of-the–art modelling techniques. In particular, our best algorithms are on average 2.6–3.2 times faster than the state-of-the-art and succeed in solving complex instances that could not be solved before within the given time limit. Further, we show in an ablation study that our enrichment components contribute considerably to the performance of the new ILP formulation.

2012 ACM Subject Classification Human-centered computing → Graph drawings; Mathematics of computing → Integer programming; Mathematics of computing → Permutations and combinations

Keywords and phrases Storyline drawing, crossing minimization, integer linear programming, algorithm engineering, computational experiments

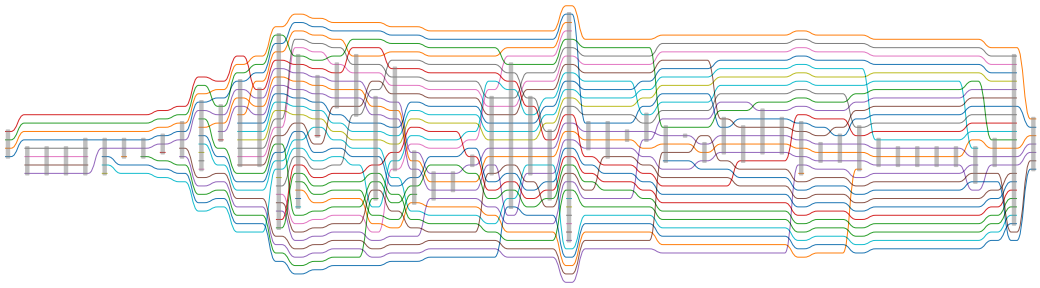
Digital Object Identifier 10.4230/LIPIcs.GD.2024.31

Related Version *Full Version*: `https://arxiv.org/abs/2409.02858` [6]

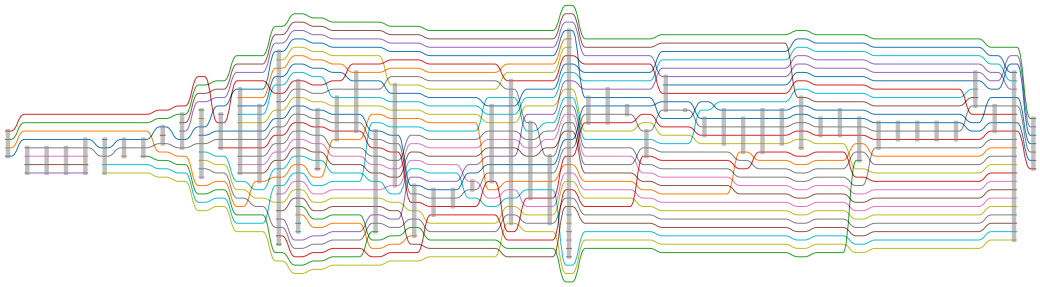
Supplementary Material *Software*: `https://doi.org/10.17605/OSF.IO/3BUA2` [5]

Funding *Alexander Dobler*: Vienna Science and Technology Fund (WWTF) grant [10.47379/ICT19035].
Julian Meffert: Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 459420781.
Petra Mutzel: partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 459420781.
Martin Nöllenburg: Vienna Science and Technology Fund (WWTF) grant [10.47379/ICT19035].

© Alexander Dobler, Michael Jünger, Paul J. Jünger, Julian Meffert, Petra Mutzel, and Martin Nöllenburg;
licensed under Creative Commons License CC-BY 4.0



(a) Solution with 374 crossings computed in 0.15 s by a greedy heuristic.



(b) Drawing with the minimum number of 236 crossings computed in 401.23 s by our new ILP formulation.

Figure 1 Storylines for the first Harry Potter movie. Interactions are shown as vertical gray bars.

1 Introduction

Storyline drawings are a well-studied visualization style for complex event-based temporal interaction data and have been popularized by the xkcd comic "Movie Narrative Charts" in 2009 [19]. They show a set of characters, e.g., from the plot of a movie or book, and how they interact or co-occur in a sequence of events over time, e.g., by participating in the same scene or conversation of the evolving story. A storyline drawing is a two-dimensional diagram, where the x-axis represents time and the y-dimension is used for the vertical grouping of characters according to their interaction sequence. The exact temporal distance of interactions is usually not depicted, only their order. Each character is represented as an x-monotone curve, and interactions are represented by vertically grouping the curves of the participating characters at the x-coordinate corresponding to the interaction time. Characters that are not participating in an interaction at any specific point in time are vertically separated from each other. Figure 1 shows an example of a storyline drawing. Due to their popularity and the intuitive data encoding, they are well suited for visual storytelling and have since been used as visual metaphors for representing a variety of different event-based data sets beyond the original book and movie plots [19, 28], e.g., for software projects [20, 27], newspaper articles [1], political debates on social media [18], visual summaries of meeting contents [23], scientific collaborations [14], sports commentary [34], and gameplay data [33].

There is one main degree of freedom when computing and optimizing storyline drawings: the (vertical) linear order and positioning of the characters at each discrete time steps. The only hard constraint is that all characters participating in the same interaction must be consecutive as a group. This degree of freedom can thus be used to minimize the number of crossings between character curves, their wiggles (i.e. the amount of vertical movement of

character lines in the visualization), and any excessive white space in the diagram, which are the three major objectives that have been identified for computing storyline drawings [27, 28]. While the number of crossings is determined purely combinatorially by the sequence of character permutations, wiggles and white space depend on the actual y-coordinates assigned to each character curve at each point in time.

In this paper, our interest is the combinatorial crossing minimization problem. It is the primary objective in practical storyline optimization pipelines [18], where it forms the input to the subsequent steps of reducing wiggles and white space while maintaining the character order. Additionally, crossing minimization is one of the most fundamental graph drawing problems [2, 22] and it is well known that graph drawings with fewer crossings increase readability [21]. Unfortunately, crossing minimization in storyline drawings is an NP-hard problem [9, 17] and hence practical approaches for storyline visualization usually resort to heuristics, even though they cannot guarantee optimal solutions.

We consider the crossing minimization problem from the opposite side and revisit exact integer linear programming (ILP) approaches [11] for computing provably optimal solutions. Such approaches often lead to practical exact algorithms. Our goal is to improve on the runtime performance of such exact methods by enriching the models with both new problem-specific insights and better heuristics. Faster exact algorithms for crossing minimization in storyline drawings are practically relevant for two reasons: firstly, solving moderately sized instances to optimality within a few seconds provides a strictly better alternative to commonly used suboptimal heuristics, and secondly, knowing optimal solutions for a large set of representative benchmark instances (even if their computations take several minutes or up to a few hours) is a prerequisite for any thorough experimental study on the performance of non-exact crossing minimization heuristics and for generating crossing-minimum stimuli in user experiments.

Related Work. Tanahashi and Ma [28] introduced storyline drawings as an information visualization problem, provided the first visual encoding model, and defined the above-mentioned optimization criteria (crossings, wiggles, white space). They suggested a genetic algorithm to compute storyline drawings. Ogawa and Ma [20] used a greedy algorithm to compute storylines to depict software evolution. Due to slow computation times of previous methods, Liu et al. [18] split the layout process into a pipeline of several subproblems ordered by importance, the first one being crossing minimization. They solved the character line ordering by an iterated application of a constrained barycenter algorithm, a classic heuristic for multi-layer crossing minimization [25]. Their results were obtained in less than a second and had fewer crossings than those computed by the genetic algorithm [28], which took more than a day to compute on some of the same instances. Tanahashi et al. [27] enhanced previous methods to take into account streaming data and apply a simple sequential left-to-right sorting heuristic. Recent practical works on storyline drawings focus on other aspects, such as an interactive editor [30] or a mixed-initiative system including a reinforcement learning AI component [29]; both these systems apply a two-layer crossing minimization heuristic [8].

Several authors focused on the combinatorial crossing minimization problem and its complexity. Kostitsyna et al. [17] observed that the NP-hardness of the problem follows from a similar bipartite crossing minimization problem [9] and proved fixed-parameter tractability when the number of characters is bounded by a parameter k. Gronemann et al. [11] were the first to model the problem as a special type of tree-constrained multi-layer crossing minimization problem. They implemented an exact branch-and-cut approach that exploits the equivalence of the quadratic unconstrained 0/1-optimization problem with the maximum

cut problem in a graph. They managed to solve many instances with up to 20 characters and 50 interactions optimally within a few seconds. Van Dijk et al. [31,32] proposed block-crossing minimization in storyline drawings, which counts grid-like blocks of crossings rather than individual crossings. They showed NP-hardness and proposed greedy heuristics, a fixed-parameter tractable algorithm, and an approximation algorithm. In a follow-up work, van Dijk et al. [32] implemented and experimentally evaluated an exact approach for the block crossing minimization problem using SAT solving. A different variation of storylines was studied by Di Giacomo et al. [10], who considered ubiquitous characters as x-monotone trees with multiple branches, enabling characters to participate in multiple simultaneous interactions; they solved the crossing minimization aspect using an adaptation of the previous SAT model [32]. Dobler et al. [7] consider time interval storylines, where additionally to the order of characters, the order of time steps in so-called time-intervals can be permuted.

The problem is also similar to crossing minimization in layered graph drawing, which was introduced by Sugiyama et al. [25]. The problem is to draw a graph with its vertices on multiple parallel lines while minimizing crossings. A notable difference to storyline crossing minimization is that vertices can have arbitrary degree and that edges can span more than one layer. For a survey of algorithms and techniques in layered graph drawing, we refer to Healy and Nikolov [13].

Contributions. The contributions of this paper are the following:

- We identify structural properties of storyline drawings and prove that there exist crossing-minimum drawings satisfying them, reducing the search space of feasible solutions.
- We propose a new ILP formulation exploiting these structural insights in order to (i) significantly reduce the number of required constraints and (ii) apply symmetry breaking constraints to strengthen the ILP model.
- We introduce several new heuristics that support the exact solver, either as initial heuristics to improve branch-and-bound pruning or for deriving integral solutions from fractional ones during the incremental ILP solving process.
- We have compiled a new benchmark set of storyline instances, including those of earlier studies, as well as several challenging new ones.
- We have conducted a detailed experimental evaluation of our new ILP model using the above benchmark set. We compare its ability to solve instances with state-of-the-art ILP models. Moreover, in an ablation study, we show that our further enhancements (e.g., adding symmetry breaking constraints and novel heuristics) contributes considerably to the performance of both the new and several state-of-the-art ILP formulations.
- We show that our ILP models are able to solve previously unsolved instances from the literature and obtain a speedup of 2.6–3.2 compared to the state of the art.

Data sets, source code, evaluation, and a visualization software are available on `https://osf.io/3bua2/`.

Due to space constraints, statements marked with ($\star$) *are proved in a full version of the paper [6].*

2 Preliminaries

Permutations. Given a set $X = \{x_1, \ldots, x_n\}$, a permutation π is a linear order of its elements, or equivalently, a bijective mapping from $\{1, 2, \ldots, |X|\}$ to X. For $x, x' \in X$ we write $x \prec_\pi x'$ if x comes before x' in π. For $Y \subseteq X$, $\pi[Y]$ is the permutation π restricted to Y, formally, for $y, y' \in Y$, $y \prec_{\pi[Y]} y'$ if and only if $y \prec_\pi y'$. For two permutations π, ϕ

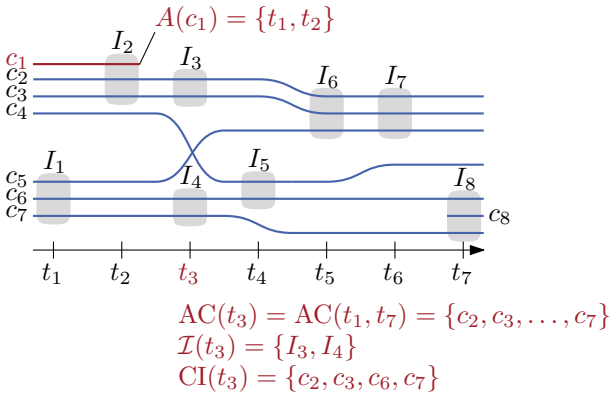


Figure 2 Illustration of important notation throughout this paper with the aid of a storyline drawing depicting interactions I_1–I_8 of the characters c_1–c_8 over the time steps t_1–t_7.

of two sets X and Y with $X \cap Y = \emptyset$, we denote by $\pi \star \phi$ their concatenation. Given two permutations π, π' of the same set X, the *inversions* between π and π' is the number of pairs $x, x' \in X$ such that $\pi^{-1}(x) < \pi^{-1}(x')$ and $\pi'^{-1}(x) > \pi'^{-1}(x')$.

Problem input. A *storyline instance* consists of a 4-tuple $(T, \mathcal{C}, \mathcal{I}, A)$ where $T = \{t_1, t_2, \ldots, t_\ell\}$ is a set of totally ordered *time steps* (or *layers*), $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$ is a set of *characters*, and $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ is a set of interactions. Each interaction $I \in \mathcal{I}$ has a corresponding time step $\mathrm{time}(I) \in T$ and consists of a set of characters $\mathrm{char}(I) \subseteq \mathcal{C}$. Further, A maps each character $c \in \mathcal{C}$ to a consecutive set of time steps, i.e., $A(c) = \{t_i, t_{i+1}, \ldots, t_j\}$ for $1 \leq i \leq j \leq \ell$. We say that character c is *active* at the time steps in $A(c)$, it *starts* at t_i and *ends* at t_j. We define $\mathrm{AC}(t)$ for $t \in T$ as the set of all characters $c \in \mathcal{C}$ active at time t, i.e., $\mathrm{AC}(t) = \{c \in \mathcal{C} \mid t \in A(c)\}$. Clearly, for each interaction $I \in \mathcal{I}$, $\mathrm{char}(I) \subseteq \mathrm{AC}(\mathrm{time}(I))$ must hold. Next, we define the set of all characters active in the time interval $[t_i, t_j]$ $(1 \leq i \leq j \leq \ell)$ as $\mathrm{AC}(t_i, t_j) = \mathrm{AC}(t_i) \cap \mathrm{AC}(t_{i+1}) \cap \cdots \cap \mathrm{AC}(t_j)$. For a time step $t \in T$ we define the set of interactions at t as $\mathcal{I}(t) = \{I \in \mathcal{I} \mid \mathrm{time}(I) = t\}$ and its corresponding set of characters as $\mathrm{CI}(t) = \bigcup_{I \in \mathcal{I}(t)} \mathrm{char}(I)$. Without loss of generality, for the interactions at time step t we assume that $|\mathcal{I}(t)| \neq 0$ and that the sets of characters of the interactions $\mathcal{I}(t)$ are pairwise disjoint. This is a reasonable assumption as characters usually participate in at most one interaction at any given time, e.g. in movies. Important notation is also illustrated in Figure 2.

Problem output. Solutions to storyline instances $(T, \mathcal{C}, \mathcal{I}, A)$ consist of a sequence of ℓ permutations $S = (\pi_1, \pi_2, \ldots, \pi_\ell)$ such that π_i is a permutation of $\mathrm{AC}(t_i)$ for all $i = 1, \ldots, \ell$ satisfying the condition that the set of characters of each interaction $I \in \mathcal{I}(t_i)$ appears consecutively. We call S a *storyline solution* or *drawing*.

The number of crossings $\mathrm{cr}(\pi_i, \pi_{i+1})$ between two consecutive permutations π_i and π_{i+1} is defined as the number of inversions of the two permutations $\pi_i[C]$ and $\pi_{i+1}[C]$, where $C = \mathrm{AC}(t_i) \cap \mathrm{AC}(t_{i+1})$. The number of crossings in a storyline solution is $\sum_{i=1}^{\ell-1} \mathrm{cr}(\pi_i, \pi_{i+1})$. The problem addressed in this paper is the following.

▶ **Problem 1** (Storyline Problem). *Given a storyline instance $(T, \mathcal{C}, \mathcal{I}, A)$, find a storyline drawing S with the minimum number of crossings.*

3 Standard Models for the Storyline Problem

The most natural ILP formulation to solve Problem 1 has a quadratic objective function and is based on the linear ordering model, which uses binary variables in order to encode the linear ordering at each time step. The number of crossings between two subsequent time steps is then given by the number of inversions of the two permutations.

From now on, we assume that characters c_u, c_v, c_w are pairwise different, even if we write for example $c_u, c_v \in C$ for some set C of characters.

Quadratic Model (QDR)

For each time step $t_i, i = 1, 2, \ldots, \ell$ and each tuple of characters $c_u, c_v \in \mathrm{AC}(t_i)$ we introduce a binary *ordering variable* $x_{i,u,v}$ which is equal to 1 if and only if $c_u \prec_{\pi_i} c_v$. The quadratic model QDR is given as follows:

$$\min \sum_{i=1}^{\ell-1} \sum_{c_u, c_v \in \mathrm{AC}(t_i, t_{i+1})} x_{i,u,v} x_{i+1,v,u} \tag{QDR}$$

$$x_{i,u,v} = 1 - x_{i,v,u} \qquad \text{for all } i = 1, \ldots, \ell; c_u, c_v \in \mathrm{AC}(t_i) \text{ with } u < v \tag{EQ}$$

$$x_{i,u,v} + x_{i,v,w} + x_{i,w,u} \leq 2 \qquad \text{for all } i = 1, \ldots, \ell; c_u, c_v, c_w \in \mathrm{AC}(t_i) \tag{LOP}$$

$$x_{i,u,w} = x_{i,v,w} \qquad \text{for all } i = 1, \ldots, \ell; I \in \mathcal{I}(t_i); \tag{TREE}$$
$$c_u, c_v \in \mathrm{char}(I), u < v; c_w \in \mathrm{AC}(t_i) \setminus \mathrm{char}(I)$$

$$x_{i,u,v} \in \{0, 1\} \qquad \text{for all } i = 1, \ldots, \ell; c_u, c_v \in \mathrm{AC}(t_i), \tag{BIN}$$

The character curves for c_u and c_v cross between the two layers t_i and t_{i+1} if and only if one of the terms $x_{i,u,v}x_{i+1,v,u}$ and $x_{i,v,u}x_{i+1,u,v}$ equals 1. The (LOP) and (EQ) constraints ensure transitivity of the set of characters $\mathrm{AC}(t_i)$ present at time step t_i and guarantee that they define a total order. For all interactions $I \in \mathcal{I}(t_i)$ the (TREE) constraints ensure that characters from I appear consecutively at the respective time step t_i.

Linearized Model (LIN)

The standard linearisation of quadratic integer programs introduces additional variables $y_{i,u,v}$ that substitute the quadratic terms $x_{i,u,v}x_{i+1,v,u}$ for all t_i, $i = 1, 2, \ldots, \ell - 1$ and each tuple of characters $c_u, c_v \in \mathrm{AC}(t_i, t_{i+1})$ in the objective function. In order to link the new variables with the ordering variables, we introduce the following constraints:

$$y_{i,u,v} \geq x_{i,u,v} - x_{i+1,u,v} \quad \text{for all } i = 1, \ldots, \ell; c_u, c_v \in \mathrm{AC}(t_i, t_{i+1}) \tag{CR}$$

Obviously, the variable $y_{i,u,v}$ is forced to 1, if the character c_u is before c_v at time step t_i in the solution represented by the y-variables, and the order of both characters is reversed at time step t_{i+1}. The linearised model (LIN) is given as follows.

$$\min \sum_{i=1}^{\ell-1} \sum_{c_u, c_v \in \mathrm{AC}(t_i, t_{i+1})} y_{i,u,v} \tag{LIN}$$

$$y_{i,u,v}, x_{i,u,v} \text{ satisfy (BIN), (EQ), (LOP), (TREE), and (CR)}$$

Max-Cut Model (CUT)

Gronemann et al. [11] have suggested a formulation based on the transformation of the problem into a quadratic unconstrained binary program with additional (TREE) constraints, which is then solved using a maximum cut approach. Here, we omit the detour via the quadratic binary program and directly provide the corresponding maximum cut formulation. Starting with a feasible storyline drawing $\hat{S} = (\hat{\pi}_1, \ldots, \hat{\pi}_\ell)$, we define the graph $G_M = (V_M, E_M)$: The vertex set V_M is given by a vertex v^* and the union of the sets V_i $(i = 1, \ldots, \ell)$, where V_i has a vertex c^i_{uv} for each pair $c_u, c_v \in \mathrm{AC}(t_i)$ with $c_u \prec_{\hat{\pi}_i} c_v$.

We introduce an edge between the vertices c^i_{uv} and c^{i+1}_{pq} if the corresponding characters coincide. In the case that $c_u = c_p$ and $c_v = c_q$, the (type-1) edge $e = e^i_{uv}$ gets a weight of $w_e = -1$, and in the case that $c_u = c_q$ and $c_v = c_p$, the (type-2) edge $e = e^i_{uv}$ gets a weight of $w_e = 1$. We define the constant K as the number of edges of type (2). The intention is the following: An edge of type (1) results in a crossing if and only if it is in the cut, and an edge of type (2) results in a crossing if and only if it is not in the cut. This construction allows for associating the maximum cut objective function values W to corresponding crossing numbers $K - W$. In particular, $W = 0$ for the empty cut corresponds to the number of crossings K in $\hat{S}$. In order to guarantee that the characters of an interaction appear consecutively, we introduce type-3 edges with weight 0 from the additional vertex v^* to every vertex in V_i for all $i = 1, \ldots, \ell$, and add the additional constraints (TRC). We introduce a binary variable z_e for every edge $e \in E_M$ in the graph, which is 1 if and only if the edge is contained in the computed cut.

The following model guarantees that every optimal solution corresponds to a constrained maximum cut in the graph G_M that provides the optimal solution to the storyline problem. The constraints (CYC) capture the fact that any intersection of a cut and a cycle in a graph has even cardinality. The correctness is provided in [11], see also [3, 24].

$$\max \quad \sum_{e \in E_M} w_e z_e \tag{CUT}$$

$$\sum_{e \in F} z_e - \sum_{e \in C \setminus F} z_e \leq |F| - 1 \quad \text{for all cycles } C \subseteq E_M, F \subseteq C, |F| \text{ odd} \tag{CYC}$$

$$0 \leq z_{(v^*, c^i_{uv})} + z_{(v^*, c^i_{vw})} - z_{(v^*, c^i_{uw})} \leq 1 \quad \text{for all } i = 1, \ldots, \ell; c^i_{uv}, c^i_{vw}, c^i_{uw} \in V_i \text{ with } c_u \prec_{\hat{\pi}_i} c_v \prec_{\hat{\pi}_i} c_w \tag{LOPC}$$

$$\left.\begin{aligned} z_{(v^*, c^i_{uw})} = z_{(v^*, c^i_{vw})} \text{ if } c_u, c_v \prec_{\hat{\pi}_i} c_w \\ z_{(v^*, c^i_{uw})} = z_{(v^*, c^i_{wv})} \text{ if } c_u, c_v \succ_{\hat{\pi}_i} c_w \end{aligned}\right\} \quad \begin{aligned} &\text{for all } i = 1, \ldots, \ell; I \in \mathcal{I}(t_i); \\ &c_u, c_v \in \mathrm{char}(I); c_w \in \mathrm{AC}(t_i) \setminus \mathrm{char}(I) \end{aligned} \tag{TRC}$$

$$z_e \in \{0, 1\} \quad \text{for all } e \in E_M \tag{BIC}$$

4 Structural Properties of Storyline Solutions

In this section, we identify structural properties of storyline solutions that will help us to optimize the models proposed in Section 5, and that guide the exact optimization process. First, we define two properties of storyline drawings. Definition 2 captures that the relative order of characters in an interaction can be propagated backwards.

▶ **Definition 2** (Type-1 consistency). *Let $S = (\pi_1, \pi_2, \ldots, \pi_\ell)$ be a solution to a storyline instance $(T, \mathcal{C}, \mathcal{I}, A)$. Let $I \in \mathcal{I}$, $t_i = \mathrm{time}(I)$ and $C = \mathrm{char}(I)$. Let $1 < j(I) \leq i$ be the index of the earliest time step $t_{j(j)}$ such that $C \subseteq \mathrm{AC}(t_{j(I)}, t_i)$ and*

$$\forall k \in \{j(I) + 1, \ldots, i\} : \mathrm{CI}(t_k) \cap C = \emptyset \vee \exists I \in \mathcal{I}(t_k) : C \subseteq \mathrm{char}(I).$$

We say that S is I-consistent if

$$\forall k \in \{j(I), j(I)+1, \ldots, i\} : \pi_k[C] = \pi_i[C].$$

Further, we say that S is type-1-consistent *if it is I-consistent for all $I \in \mathcal{I}$.*

Definition 3 defines the property that between suitable pairs of interactions with the same set of characters, these characters are kept together between the two time steps. Note that this is not implied by type-1 consistency.

▶ **Definition 3** (Type-2 consistency). *Let $S = (\pi_1, \pi_2, \ldots, \pi_\ell)$ be a solution to a storyline instance $(T, \mathcal{C}, \mathcal{I}, A)$. Consider two interactions $I_1, I_2 \in \mathcal{I}$ such that*

- $\mathrm{char}(I_1) = \mathrm{char}(I_2) = C$,
- $i = \mathrm{time}(I_1) < \mathrm{time}(I_2) = j$, *and*
- $\forall k \in \mathbb{N} : i < k < j \Rightarrow [\mathrm{CI}(t_k) \cap C = \emptyset \vee \exists I_3 \in \mathcal{I}(t_k) : C \subseteq \mathrm{char}(I_3)]$.

We say that S is (I_1, I_2)-consistent if

$$\forall i < k < j : \exists \pi^a, \pi^b : \pi_k = \pi^a \star \pi_i[C] \star \pi^b.$$

Further, we say that S is type-2-consistent if it is (I_1, I_2)-consistent for all such pairs (I_1, I_2).

The following lemma shows that we can achieve type-1 consistency for storyline drawings without increasing the number of crossings. Essentially, if a storyline solution is not type-1 consistent for an interaction I, we can propagate the relative order of characters in that interaction forward from the time step $t_{j(I)}$ from Definition 2.

▶ **Lemma 4** ($\star$). *Let $(T, \mathcal{C}, \mathcal{I}, A)$ be an instance with a solution S. We can construct from S a type-1-consistent solution S' such that $\mathrm{cr}(S') \leq \mathrm{cr}(S)$. If S is type-2-consistent, so is S'.*

A similar result with a related proof argument holds for type-2 consistency.

▶ **Lemma 5** ($\star$). *Let $(T, \mathcal{C}, \mathcal{I}, A)$ be an instance with a solution S. We can construct from S a type-2-consistent solution S' such that $\mathrm{cr}(S') \leq \mathrm{cr}(S)$. If S is type-1-consistent, so is S'.*

The following is a direct consequence.

▶ **Corollary 6.** *For each storyline instance $(T, \mathcal{C}, \mathcal{I}, A)$ there exists a crossing-minimum solution S that is type-1-consistent and type-2-consistent.*

Theorem 7 is the main ingredient for a new ILP formulation given in Section 5. It shows that we can in specific cases assume the order of characters C_a above and C_b below an interaction at time step t_i to be equal to the relative order at t_{i-1}. This is similar to type-1-consistency, where the relative order of characters in an interaction sometimes can be kept.

▶ **Theorem 7** ($\star$). *Let $(T, \mathcal{C}, \mathcal{I}, A)$ be a storyline instance. There exists a crossing-minimum solution $S = (\pi_1, \pi_2, \ldots, \pi_\ell)$ with the following property. For all $t_i \in \{t_2, t_3, \ldots, t_\ell\}$ with $|\mathcal{I}(t_i)| = 1$, where $\mathcal{I}(t_i) = \{I\}$, the following holds.*

(1) $\exists C_a, C_b : \pi_i = \pi_i[C_a] \star \pi_i[\mathrm{char}(I)] \star \pi_i[C_b]$,
(2) *if* $C_a \subseteq \mathrm{AC}(t_{i-1}, t_i)$, *then* $\pi_i[C_a] = \pi_{i-1}[C_a]$,
(3) *if* $\mathrm{char}(I) \subseteq \mathrm{AC}(t_{i-1}, t_i)$, *then* $\pi_i[\mathrm{char}(I)] = \pi_{i-1}[\mathrm{char}(I)]$, *and*
(4) *if* $C_b \subseteq \mathrm{AC}(t_{i-1}, t_i)$, *then* $\pi_i[C_b] = \pi_{i-1}[C_b]$.

Proof sketch. For time steps t with $|\mathcal{I}(t)| = 1$ let C_a be the characters above the interaction in an optimal solution. Similarly, let C_b be the characters below. By imagining that C_a and C_b form two respective interactions, the result follows from Lemma 4. ◀

5 Refining the ILP models

We apply our structural insights from Section 4 to the models (besides the (CUT)-model) to obtain a new ILP formulation, including a reduction of the number of (LOP) constraints in Section 5.1 via Theorem 7 and the inclusion of additional symmetry breaking constraints in Section 5.2 via Corollary 6.

5.1 The Propagated Linear Ordering Model (PLO)

For our new formulation, we take the linearized model (LIN) as basis, but remove some of the (LOP)-constraints for time step t_i as we can make use of propagating the ordering at t_{i-1} by Theorem 7 as follows. If $\mathcal{I}(t_i)$ for $i > 1$ contains only one interaction I, and no characters outside the interaction start at t_i (i.e., $\mathrm{AC}(t_i) \setminus \mathrm{AC}(t_{i-1}) \subseteq \mathrm{CI}(t_i)$), we only include a part of the (LOP)-constraints for time step t_i using a representative character $c_w \in \mathrm{char}(I)$:

From the set of (LOP)-constraints containing at least one character in $\mathrm{AC}(t_i) \setminus \mathrm{char}(I)$, we keep only those that contain exactly two characters in $\mathrm{AC}(t_i) \setminus \mathrm{char}(I)$ and the representative character $c_w \in \mathrm{char}(I)$. This is sufficient, because we can define the order of the active characters in t_i relative to the order of the characters in the interaction I based on Theorem 7. Hence, let c_w be a representative character from the set $\mathrm{char}(I)$, and consider a pair of characters $c_u, c_v \in \mathrm{AC}(t_i) \setminus \mathrm{char}(I)$. By Theorem 7, if both c_u and c_v are above or below c_w, then their relative order can be fixed by their relative order at t_{i-1}. Otherwise, their relative order is already given by their relative order to c_w. That is, if, e.g., c_u is above c_w and c_v is below c_w, then we know that c_u is above c_v. To ensure the above, we add the following constraints in addition to the (LOP)-constraints for c_u, c_v, and c_w at time step t_i.

$$x_{i,u,v} \geq x_{i-1,u,v} + x_{i,u,w} + x_{i,v,w} - 2 \tag{PROP-R1}$$

$$x_{i,u,v} \geq x_{i-1,u,v} + x_{i,w,u} + x_{i,w,v} - 2 \tag{PROP-R2}$$

The two constraints ensure that c_u is above c_v if the requirements are met. By switching c_u and c_v, these constraints also ensure the case that c_u is below c_v.

If additionally $\mathrm{char}(I) \subseteq \mathrm{AC}(t_{i-1})$ we can apply Theorem 7 (3) to further reduce the number of those (LOP)-constraints, whose triples are taken from the set $\mathrm{char}(I)$: In this case, we do not add any of the (LOP)-constraints for the characters in I, but instead for each pair $c_u, c_v \in \mathrm{char}(I)$, we add the following constraint ensuring that the relative order of c_u and c_v is the same for t_i and t_{i-1}.

$$x_{i,u,v} = x_{i-1,u,v} \tag{PROP-I}$$

If both reductions for (LOP) apply, we get a quadratic rather than cubic number of constraints for t_i. We call this formulation *propagated linear order* (PLO). Note that this idea of reducing the number of (LOP)-constraints also works for any of the other standard ILP models.

▶ **Theorem 8** (⋆). *Every optimal solution to the formulation (PLO) corresponds to a crossing minimum storyline drawing.*

Proof sketch. Since we have only reduced some of the (LOP)-constraints, the correctness follows by induction on the time steps w.r.t. transitivity of the computed character order. ◀

5.2 Symmetry Breaking Constraints

We introduce the set (SBC) of *symmetry breaking constraints* that are based on Corollary 6 and might improve the solving process of the models, as they constitute equalities:

- We can assume that a crossing-minimum solution is type-1 consistent. Thus let $I \in \mathcal{I}$ with $t_i = \text{time}(I)$ and let $j(I)$ be defined as in Definition 2. For all pairs $c_u, c_v \in \text{char}(I)$ and all $j(I) \leq k < i$ we can add the following constraint enforcing type-1 consistency:

$$x_{k,u,v} = x_{i,u,v} \tag{SBC-1}$$

- We can assume that a crossing-minimum solution is type-2 consistent. Thus, let $I_1, I_2 \in \mathcal{I}$ be two distinct interactions satisfying the properties of Definition 3. Let $i = \text{time}(I_1)$ and $j = \text{time}(I_2)$. For all $i < k < j$, all pairs $c_u, c_v \in \text{char}(I_1)$, and all $c_w \in \text{AC}(t_k) \setminus \text{char}(I_1)$ we add the following constraint, enforcing type-2 consistency:

$$x_{k,u,w} = x_{k,v,w} \tag{SBC-2}$$

6 Implementation

In this section, we discuss relevant implementation details and new heuristic-based approaches to improve our algorithms.

6.1 Initial Heuristic

We propose a heuristic that is provided to the ILP solver as starting solution: Roughly, we solve intervals of the instance consisting of $\hat{\ell} < \ell$ consecutive time steps optimally using the (PLO) ILP formulation. The first $\hat{s} \leq \hat{\ell}$ layers of those are saved as the heuristic solution. Then, the last layer is fixed (i.e. layer $\hat{s}$) by fixing the ordering variables accordingly, in order to compute the solution for the next interval, and so on. Initial testing showed that $\hat{s} = 5$ and $\hat{\ell} = 30$ yields a good tradeoff between runtime and solution quality. A more detailed description is given in the full version [6].

6.2 Rounding and Local Improvement Heuristics

We propose a rounding heuristic that exploits fractional LP-solutions. Furthermore, we try to improve these solutions as well as incumbent solutions found by the solver software by proposing three local improvement heuristics **Rem-DC**, **Push-CR**, and **SL-Bary**.

Rounding Heuristic. We propose a strategy to round fractional solutions of the ordering variables to valid integer solutions corresponding to a drawing of the storyline instance. Roughly, the orders π_i are computed by considering for each ordering variable $x_{i,u,v}$ its rounded up or down value if the fractional value is different from 0.5 in the LP solution, and propagating the relative order of u and v in π_{i-1} if the fractional value is 0.5. Then, the characters are sorted by the sums of the rounded values, characters belonging to the same interaction are "treated as one character". A detailed description is given in the full version [6].

Local Improvement Heuristics.

Rem-DC (remove double crossings) This heuristic finds pairs of characters that cross twice, and both crossings can be removed without increasing the total number of crossings. Formally, this is possible for a drawing S and two characters c, c' if there exist $1 \leq i < j \leq \ell$ with $j - i > 1$ such that

- c and c' cross between t_i and t_{i+1}, and t_{j-1} and t_j, and
- for all $k \in \mathbb{N}$ with $i < k < j$, c and c' either belong to the same interaction in t_k, or they both are in no interaction for t_k.

Then for all k as above we can exchange c and c' in π_k. This removes the double-crossing between c and c' and further does not introduce new crossings.

Push-CR This heuristic proceeds from $i = 2, \ldots, \ell$ in this order and tries to push crossings between π_{i-1} and π_i forward by one time step: Let C be a maximal set of characters such that (1) all characters in C appear consecutively in π_i, (2) $C \subseteq \mathrm{AC}(t_{i-1}, t_i)$, and (3) all characters in C either appear in the same interaction in t_i or no character in C is part of an interaction. For each such set of characters C we replace in π_i, $\pi_i[C]$ by $\pi_{i-1}[C]$. By similar arguments as in Section 4 this never increases the number of crossings.

Bary-SL Lastly, we describe a variant of the barycenter heuristic [25] for storylines that iteratively improves a storyline drawing by updating π_i for $1 \leq i \leq \ell$ based on π_{i-1} and π_{i+1} or one of them if not both exist. It is only applied to π_i if $|\mathcal{I}(t_i)| = 1$. Informally, we say that a pair of characters c, c' is comparable if c and c' have the same relative order in π_{i-1} and π_{i+1}. We compute an ordering π_i such that most comparable pairs have the same relative order in $\pi_{i-1}, \pi_i, \pi_{i+1}$ as follows. We compute the directed auxiliary graph G_C whose vertex set is a subset S of $\mathrm{AC}(t_i)$ and which contains an arc from c to c' for each comparable pair c and c' such that c is before c' in π_{i-1} and π_{i+1}. Then, an order of S is built by iteratively selecting the vertex from G_C with the fewest incoming arcs. We also ensure that characters c that are not part of I are above or below the characters in I depending on which option leads to fewer crossings between c and $\mathrm{char}(I)$ with respect to the considered time steps t_{i-1}, t_i, t_{i+1}. The algorithm computing the order based on the graph G_C is then applied to the characters in the interaction yielding π_I, and those not in the interaction yielding π_C, respectively. The ordering π_C is inserted into the maximum position of π_I such that all characters before π_C "prefer" being above the interaction with regard to crossings with $\mathrm{char}(I)$. The new π_i is only accepted if it decreases the number of crossings.

Both **Bary-SL** and **Push-CR** are applied successively to layers $2, \ldots, \ell$. This is repeated five times and applied to valid integer solutions found by the solver and the rounding heuristic described above. If enabled, the rounding heuristic is applied to every LP solution found by the solver. **Rem-DC** is applied five times to each pair of characters.

6.3 Max-Cut Implementation Details

Since the original implementation of Gronemann et al. [11] is not available, we provide our own implementation that was optimized beyond their algorithm. After reading the input, we first find an initial starting solution by applying adapted barycenter techniques as described in [11]. We start the root relaxation with the objective function and the tree constraints (TRC) as the only constraints, and start separating the odd cycle (CYC) (as suggested by Charfreitag et al. [4]) and the (LOPC) constraints. The (LOPC) constraints are separated by complete enumeration. Whenever a new LP solution is available, all nonbinding inequalities are eliminated, and we try to exploit the information in the (fractional) solution in order to obtain a better incumbent solution.

The root phase ends when no violated inequalities are found. Then the branch-and-cut phase is started by changing the variable types from continuous in the interval $[0, 1]$ to binary. In the Gurobi "MIPSOL" callbacks at branch-and-cut nodes with an integer solution, we check if the integral solution is the characteristic vector of a storyline drawing. If so and if the number of crossings is lower than the one of the incumbent solution, the latter is updated, otherwise, the exact (CYC) and (LOPC) separators are called to provide violated inequalities that are passed to Gurobi as *lazy constraints*. In the Gurobi "MIPNODE" callbacks it is tried to exploit the fractional solution for a possible update of the incumbent solution.

6.4 Implementation of the ILP Models

The models (QDR), (LIN), and (PLO) include many symmetries regarding ordering variables and crossing variables. For each $t_i \in T$ and pair of characters $c_u, c_v \in \mathrm{AC}(t_i)$ we only keep the ordering variables $x_{i,u,v}$ and crossing variables $y_{i,u,v}$ with $u < v$. The constraints are adjusted with standard projections [11, 12].

We take the linearized model (LIN) as basis for our refined ILP model described in Section 5, because preliminary experiments showed no performance gain from refining (QDR) instead. Further, the linearized model (LIN) is competitive with the max-cut approach when implemented in Gurobi. Therefore, we decided on refining the linearized model that is simpler to implement and more accessible when compared with the max-cut approach. Furthermore, implementing any of the ILP models naively includes up to $\mathcal{O}(\ell n^3)$ (LOP)-constraints in the model. We have experimented with adding these constraints during a cutting-plane approach and also by including them into the Gurobi solver as *lazy constraints*, i.e., constraints that the solver can decide to include at later stages during the solving process. We decided to always add (LOP) as lazy constraints, as this leads to the best performance. Hence, we consider the following algorithms for our experimental evaluation.

- **MC**: the max-cut formulation (as a baseline) implemented as described in Section 6.3
- **LIN**: the linearized model (LIN) with (LOP)-constraints included as lazy constraints
- **QDR**: the quadratic model (QDR) with (LOP)-constraints included as lazy constraints
- **PLO**: the PLO formulation with (LOP)-constraints included as lazy constraints

The latter three algorithms are by default extended with the symmetry breaking constraints described in Section 5.2 (**SBC**), the initial heuristic from Section 6.1 (**INIT**), and the rounding and local improvement heuristics from Section 6.2 (**RND**). This is not done for **MC**, as it should serve as a state-of-the-art baseline and allow comparison with Gronemann et al. [11].

7 Experiments and Evaluation

In our experimental evaluation, we are interested in the following research questions.

Q1: Does the algorithm **PLO** based on our new ILP model dominate the state-of-the-art model **MC**? Will we be able to solve hard instances that have not been solved to optimality before? How do the various algorithms compare to each other?

Q2: What effect do the structural insights have when applied to the **LIN**-formulation?

Q3: What is the effect of the newly introduced components **SBC**, **INIT**, and **RND**?

In the following we describe our experimental setup, our benchmark instances, and the results of our study. We also provide our results and analysis on `https://osf.io/3bua2/`.

7.1 Setup

Systems employed for all experiments have AMD EPYC 7402, 2.80GHz 24-core CPUs and 1024GB of RAM, running Ubuntu 18.04.6 LTS; experiments were run using a single thread.

MC is implemented in `C` and compiled with `gcc 7.5.0`, `GNU make 4.1`, and flag `-O3`, all remaining code is written in `C++17`, compiled with `cmake 3.10.2` and `g++ 11.4.0` in `Release` mode. To solve the ILPs we used Gurobi 11.0.1. The time limit is 3600s (same as Gronemann et al. [11]) and the memory limit is 16GB for all experiments. We do not know the memory limit for Gronemann, however memory was certainly not our limiting factor. The time for the initial heuristic is negligible ($< 1\%$ of the overall runtime), so it is not counted towards the solving time.

To mitigate performance variability, we ran each instance-setting combination with five different seeds provided to Gurobi; data displayed below corresponds to the seed with the median runtime. With this, an instance counts as "solved in the time limit", if the majority of the five runs does not time out. Source code for the new formulations is available on `https://osf.io/3bua2/`.

7.2 Test Data

The instances used in our computational study are taken from the literature [7, 10, 11, 26]. However, we also present a new data set, including existing instances, in a specifically designed storyline data format, together with tools for transformation and visualization of the storyline layouts on `https://osf.io/3bua2/`. We also provide data on best known crossing numbers.

The existing instances from Gronemann et al. [11] consist of three book instances from the Stanford GraphBase database [16], i.e., *Anna Karenina* (anna), *Les Misérables* (jean) and *Adventures of Huckleberry Finn* (huck), and the movie instances TheMatrix, Inception, and StarWars. The instances gdea10, gdea20 from Dobler et al. [7] consist of publication data from 10 (resp. 20) authors from the GD conference. The publication instances ubiq1, ubiq2 are from Di Giacomo et al. [10]. Furthermore, anna and jean are split up into slices of 1-4 consecutive chapters as was done by Gronemann et al. [11][1]. The new instances consist of scenes from nine blockbuster movies, namely Avatar, Back to The Future, Barbie, Forrest Gump, Harry Potter 1, Jurassic Park, Oceans 11, Oppenheimer, and Titanic.

In all 59 resulting instances, characters are active from their first interaction to their last interaction, and most instances have one interaction per time step.

7.3 Evaluation

Several instances could be solved within 30s by all algorithms, others could not be solved within the time limit by any of the algorithms. The three instances TheMatrix, Inception, and StarWars used commonly in heuristic storyline visualization were all solved within 450ms. We exclude all these instances and focus on the 23 challenging instances that remain. Out of these, the maximum number of characters is 88, and the maximum number of layers is 234. An extended experimental evaluation and more detailed statistics are given in the full version [6].

Answering Q1 and Q2. Figure 3 displays the number of instances solved over time for each algorithm. We observe that the algorithms differ in their ability to solve challenging instances: **PLO** solves the most, followed by **LIN** and **MC**, with **QDR** last. In fact, **PLO** solves one instance that cannot be solved by any other algorithm, and additionally solves six instances that could not be solved by Gronemann et al. [11] and three more than **MC** within the same time limit. Hence, we answer the first part in **Q1** positively. For further illustration, the exact runtimes per instance are also shown in Figure 4.

Answering **Q2**, the structural insights as applied in **PLO** reduce the number of constraints by a factor of five on average, comparing **LIN** and **PLO**. More so, they enhance Gurobi's capabilities of strengthening the LP relaxation, as the two instances not solved by **LIN** are

[1] We could not replicate this process fully equivalently, as sometimes our optimal crossing numbers are different to those of Gronemann et al. [11].

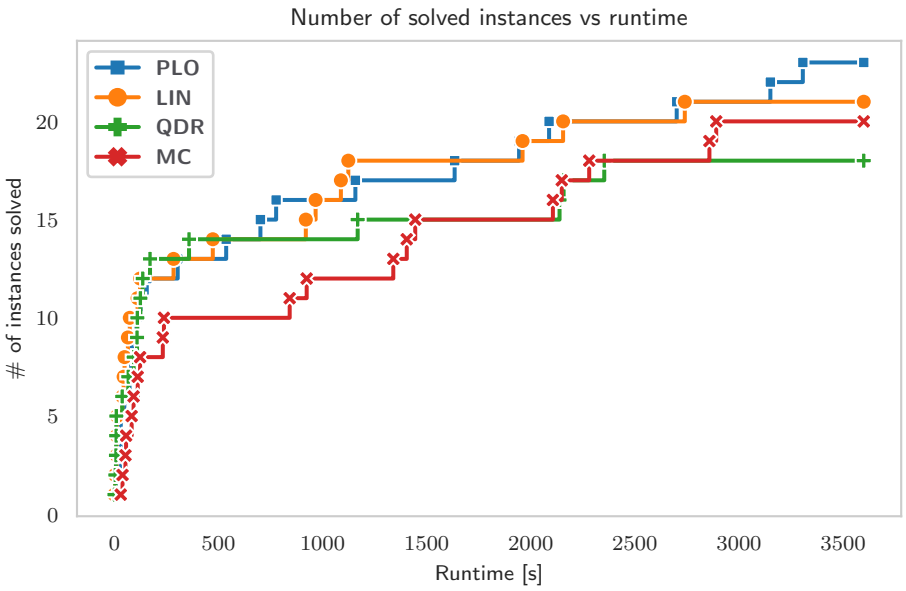


Figure 3 Number of solved instances per time limit given.

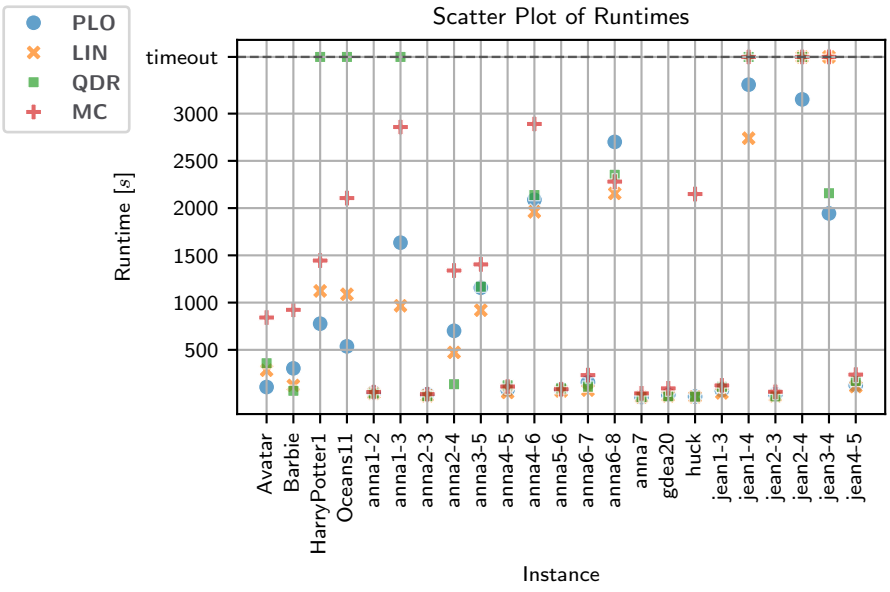


Figure 4 Runtimes of algorithms per instance.

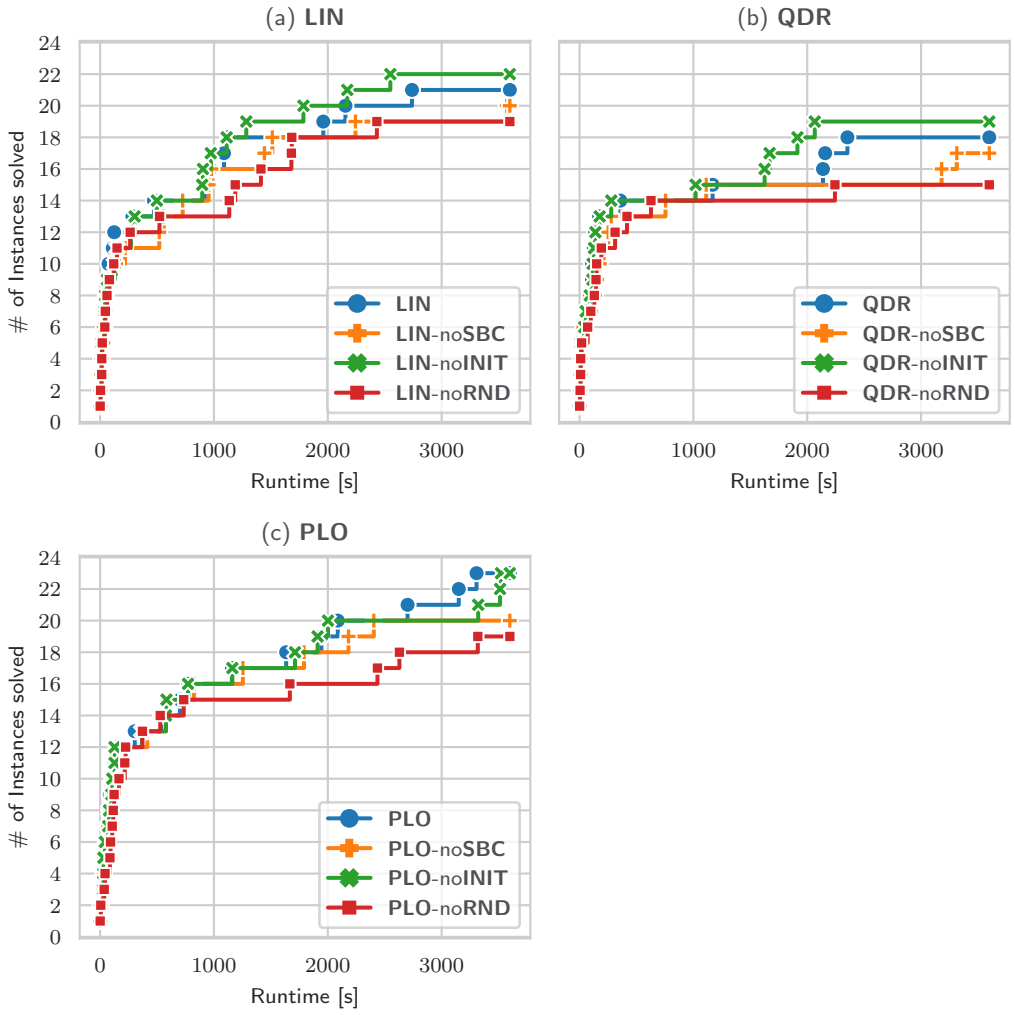


Figure 5 Number of instances solved per time limit given, broken down by algorithm. Algorithms are compared with their counterparts where exactly one component is disabled.

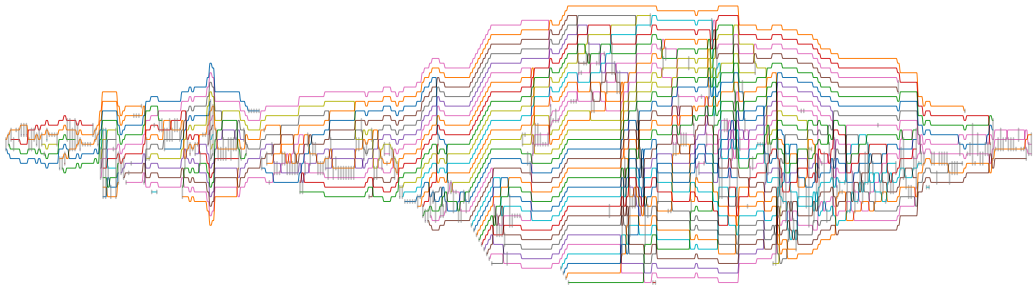
solved by **PLO** in the root, while **LIN** starts branching early and times out. **QDR** enters branching in all 23 instances, **PLO** in two, **MC** in three, **LIN** in seven instances. **PLO** solves 21 out of 23 instances in the root, the remaining two with branching.

Furthermore, we computed the *speedup factor* of **PLO**, **LIN**, and **QDR**, when compared with **MC** on instances where both respective algorithms did not time out. This factor is the runtime of **MC** divided by the runtime of, e.g., **PLO**. The geometric means of these values are 2.6 for **PLO**, 3.2 for **LIN**, and 2.7 for **QDR**. Hence, our new algorithms are 2.6–3.2 times faster than the state-of-the-art algorithm **MC**.

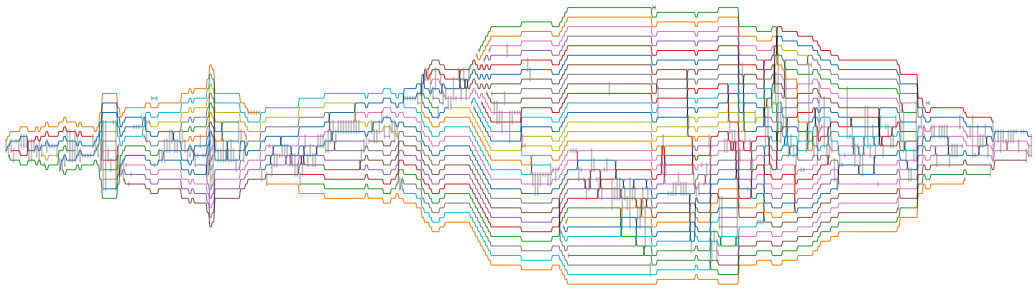
Ablation study to answer Q3. We conduct an ablation study to discern the impact each of the methods proposed in Sections 5 and 6 has on the algorithms' performance. To this end, we enable all the proposed methods as the baseline configuration for **PLO**, **LIN**, and **QDR**, namely **SBC**, **INIT** and **RND**. Then, each component is disabled one at a time to measure the component's impact on overall performance. In Table 1 we present the speedup factors of

Table 1 Geometric means of the speedup factor of each baseline algorithm vs. its counterpart where the respective component is disabled.

speedup factor	PLO	LIN	QDR
SBC	1.12	1.35	1.42
INIT	1.05	1.09	0.97
RND	1.50	1.37	1.52



(a) Solution with 765 crossings computed in 0.57 s by a greedy heuristic.



(b) With the minimum number 244 of crossings computed in ≈ 7 hours.

Figure 6 Storyline of Les Misérables (`jean`) with 80 characters and 402 layers.

the algorithms vs. their counterparts with the specific component disabled. From this table, we conclude that SBC and the RND are beneficial for all algorithms, while INIT has a small to no noticeable impact. This is further supported by Figure 5, which shows that disabling SBC or RND, results in all the formulations solving fewer instances (curves with noSBC and noRND are below the baseline). This is because SBC introduces equalities between two variables, and hence improve presolving capabilities and reduce the search space that solvers have to explore. The heuristics of RND help the solver find optimal solutions early in the process. This answers Q3.

Large Instances. Finally, we demonstrate that our implementations are capable of solving even very large instances to proven optimality: Figure 6(a) shows the raw drawing of the data for Victor Hugo's Les Misérables [15] as provided in the data file `jean.dat` of the Stanford GraphBase [16]. After roughly 7 hours of single thread computation, we obtained the proven crossing minimum layout shown in Figure 6(b).

8 Conclusion and Future Work

As shown in our experimental study, our new methods and algorithms dominate the state-of-the-art algorithms and are able to solve large instances to optimality, while the newly introduced improvements are beneficial towards all considered formulations. We observe two directions for future work.

- Our new components for improvement could be implemented into the max-cut formulation. However, initial experiments have shown that the simple linearized formulation (LIN) performs comparably to the more complex max-cut formulation, hence we expect that this will result in a negligible or no improvement over our proposed formulations.
- Out of our 59 instances (see `https://osf.io/3bua2/`) we were able to solve 55 when increasing the time limit. The remaining four unsolved instances should pose a challenge to engineer new exact methods for crossing minimization in storylines.

References

1 Vanessa Peña Araya, Tong Xue, Emmanuel Pietriga, Laurent Amsaleg, and Anastasia Bezerianos. Hyperstorylines: Interactively untangling dynamic hypergraphs. *Inf. Vis.*, 21(1):38–62, 2022. doi:10.1177/14738716211045007.

2 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

3 Christoph Buchheim, Angelika Wiegele, and Lanbo Zheng. Exact algorithms for the quadratic linear ordering problem. *INFORMS J. Comput.*, 22(1):168–177, 2010. doi:10.1287/IJOC.1090.0318.

4 Jonas Charfreitag, Michael Jünger, Sven Mallach, and Petra Mutzel. *McSparse: Exact Solutions of Sparse Maximum Cut and Sparse Unconstrained Binary Quadratic Optimization Problems*, pages 54–66. Proc. Symposium on Algorithm Engineering and Experiments (ALENEX'22), 2022. doi:10.1137/1.9781611977042.5.

5 Alexander Dobler, Michael Jünger, Paul J. Jünger, Julian Meffert, Petra Mutzel, and Martin Nöllenburg. Revisiting ILP Models for Exact Crossing Minimization in Storyline Drawings: Supplementary Material. Software (visited on 2024-10-14). URL: `https://doi.org/10.17605/OSF.IO/3BUA2`.

6 Alexander Dobler, Michael Jünger, Paul Jünger, Julian Meffert, Petra Mutzel, and Martin Nöllenburg. Revisiting ILP models for exact crossing minimization in storyline drawings. *CoRR*, abs/2409.02858, 2024. doi:10.48550/arXiv.2409.02858.

7 Alexander Dobler, Martin Nöllenburg, Daniel Stojanovic, Anaïs Villedieu, and Jules Wulms. Crossing minimization in time interval storylines. *CoRR*, abs/2302.14213, 2023. doi:10.48550/arXiv.2302.14213.

8 Michael Forster. A fast and simple heuristic for constrained two-level crossing reduction. In János Pach, editor, *Proc. Graph Drawing and Network Visualization (GD'04)*, volume 3383 of *LNCS*, pages 206–216. Springer, 2004. doi:10.1007/978-3-540-31843-9_22.

9 Michael R. Garey and David S. Johnson. Crossing number is NP-complete. *SIAM J. on Algebraic and Discrete Methods*, 4(3):312–316, 1983. doi:10.1137/0604033.

10 Emilio Di Giacomo, Walter Didimo, Giuseppe Liotta, Fabrizio Montecchiani, and Alessandra Tappini. Storyline visualizations with ubiquitous actors. In David Auber and Pavel Valtr, editors, *Proc. Graph Drawing and Network Visualization (GD'20)*, volume 12590 of *LNCS*, pages 324–332. Springer, 2020. doi:10.1007/978-3-030-68766-3_25.

11 Martin Gronemann, Michael Jünger, Frauke Liers, and Francesco Mambelli. Crossing minimization in storyline visualization. In Yifan Hu and Martin Nöllenburg, editors, *Proc. Graph Drawing and Network Visualization (GD'16)*, volume 9801 of *LNCS*, pages 367–381. Springer, 2016. doi:10.1007/978-3-319-50106-2_29.

12 Martin Grötschel, Michael Jünger, and Gerhard Reinelt. Facets of the linear ordering polytope. *Math. Program.*, 33(1):43–60, 1985. doi:10.1007/BF01582010.

13 Patrick Healy and Nikola S. Nikolov. Hierarchical drawing algorithms. In Roberto Tamassia, editor, *Handbook on Graph Drawing and Visualization*, chapter 13, pages 409–453. Chapman and Hall/CRC, 2013.

14 Tim Herrmann. Storyline-Visualisierungen für wissenschaftliche Kollaborationsgraphen. Master's thesis, Universität Würzburg, 2022. URL: https://www1.pub.informatik.uni-wuerzburg.de/pub/theses/2022-herrmann-masterarbeit.pdf.

15 Victor Hugo. *Les Misérables*. Jules Rouff et Cie editeurs, Paris, 1862.

16 Donald E. Knuth. *The Stanford GraphBase – A platform for combinatorial computing*. ACM, 1993.

17 Irina Kostitsyna, Martin Nöllenburg, Valentin Polishchuk, André Schulz, and Darren Strash. On minimizing crossings in storyline visualizations. In Emilio Di Giacomo and Anna Lubiw, editors, *Proc. Graph Drawing and Network Visualization (GD'15)*, volume 9411 of *LNCS*, pages 192–198. Springer, 2015. doi:10.1007/978-3-319-27261-0_16.

18 Shixia Liu, Yingcai Wu, Enxun Wei, Mengchen Liu, and Yang Liu. Storyflow: Tracking the evolution of stories. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2436–2445, 2013. doi:10.1109/TVCG.2013.196.

19 Randall Munroe. Movie narrative charts, 2009. URL: https://xkcd.com/657/.

20 Michael Ogawa and Kwan-Liu Ma. Software evolution storylines. In Alexandru C. Telea, Carsten Görg, and Steven P. Reiss, editors, *Software Visualization (SoftVis'10)*, pages 35–42. ACM, 2010. doi:10.1145/1879211.1879219.

21 Helen Purchase. Which aesthetic has the greatest effect on human understanding? In *Proc. Graph Drawing and Network Visualization (GD'97)*, volume 1353 of *LNCS*, pages 248–261. Springer, 1997. doi:10.1007/3-540-63938-1_67.

22 Marcus Schaefer. *Crossing Numbers of Graphs*. CRC Press, 2018.

23 Yang Shi, Chris Bryan, Sridatt Bhamidipati, Ying Zhao, Yaoxue Zhang, and Kwan-Liu Ma. Meetingvis: Visual narratives to assist in recalling meeting context and content. *IEEE Trans. Vis. Comput. Graph.*, 24(6):1918–1929, 2018. doi:10.1109/TVCG.2018.2816203.

24 Caterina De Simone. The cut polytope and the boolean quadric polytope. *Discret. Math.*, 79(1):71–75, 1990. doi:10.1016/0012-365X(90)90056-N.

25 Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst. Man Cybern.*, 11(2):109–125, 1981. doi:10.1109/TSMC.1981.4308636.

26 Y. Tanahashi. Movie data set, 2013. URL: http://vis.cs.ucdavis.edu/~tanahashi/datadownloads/storylinevisualizations/story_data.tar.

27 Yuzuru Tanahashi, Chien-Hsin Hsueh, and Kwan-Liu Ma. An efficient framework for generating storyline visualizations from streaming data. *IEEE Trans. Vis. Comput. Graph.*, 21(6):730–742, 2015. doi:10.1109/TVCG.2015.2392771.

28 Yuzuru Tanahashi and Kwan-Liu Ma. Design considerations for optimizing storyline visualizations. *IEEE Trans. Vis. Comput. Graph.*, 18(12):2679–2688, 2012. doi:10.1109/TVCG.2012.212.

29 Tan Tang, Renzhong Li, Xinke Wu, Shuhan Liu, Johannes Knittel, Steffen Koch, Lingyun Yu, Peiran Ren, Thomas Ertl, and Yingcai Wu. Plotthread: Creating expressive storyline visualizations using reinforcement learning. *IEEE Trans. Vis. Comput. Graph.*, 27(2):294–303, 2021. doi:10.1109/TVCG.2020.3030467.

30 Tan Tang, Sadia Rubab, Jiewen Lai, Weiwei Cui, Lingyun Yu, and Yingcai Wu. iStoryline: Effective convergence to hand-drawn storylines. *IEEE Trans. Vis. Comput. Graph.*, 25(1):769–778, 2019. doi:10.1109/TVCG.2018.2864899.

31 Thomas C. van Dijk, Martin Fink, Norbert Fischer, Fabian Lipp, Peter Markfelder, Alexander Ravsky, Subhash Suri, and Alexander Wolff. Block crossings in storyline visualizations. *J. Graph Algorithms Appl.*, 21(5):873–913, 2017. doi:10.7155/JGAA.00443.

32 Thomas C. van Dijk, Fabian Lipp, Peter Markfelder, and Alexander Wolff. Computing storyline visualizations with few block crossings. In Fabrizio Frati and Kwan-Liu Ma, editors, *Proc. Graph Drawing and Network Visualization (GD'17)*, volume 10692 of *LNCS*, pages 365–378. Springer, 2017. doi:10.1007/978-3-319-73915-1_29.

33 Günter Wallner, Letian Wang, and Claire Dormann. Visualizing the spatio-temporal evolution of gameplay using storyline visualization: A study with league of legends. *Proc. ACM Hum. Comput. Interact.*, 7(CHI):1002–1024, 2023. doi:10.1145/3611058.

34 Yunchao Wang, Guodao Sun, Zihao Zhu, Tong Li, Ling Chen, and Ronghua Liang. E^2Storyline: Visualizing the relationship with triplet entities and event discovery. *ACM Trans. Intell. Syst. Technol.*, 15(1):16:1–16:26, 2024. doi:10.1145/3633519.