Practical Course Report

On Block Crossing Optimal Drawings of Storylines

Peter Markfelder

Date of Submission: January 20, 2021
Advisors: Prof. Dr. Alexander Wolff
Dr. Thomas van Dijk

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

Abstract

Storyline visualizations show the structure of a story by depicting the interactions of the characters overtime. Each character is represented by an x-monotone curve from left to right and an interaction is represented by having the curves of the participating characters run close together for the duration of the interaction. To keep the visual complexity low, rather than minimizing pairwise crossings of curves, we count *block crossings*, which are intersecting bundles of curves. There have been various approaches to drawing storyline visualizations in an automated way.

In previous work we modeled the problem as a satisfiability problem, using modern SAT solvers to generate block crossing optimal drawings in reasonable time. As a continuation of that, we again model the problem as a satisfiability problem. We show that our new model is superior in runtime compared to the old approach, by evaluating our model on the same real-world instances and random instances as in previous work.

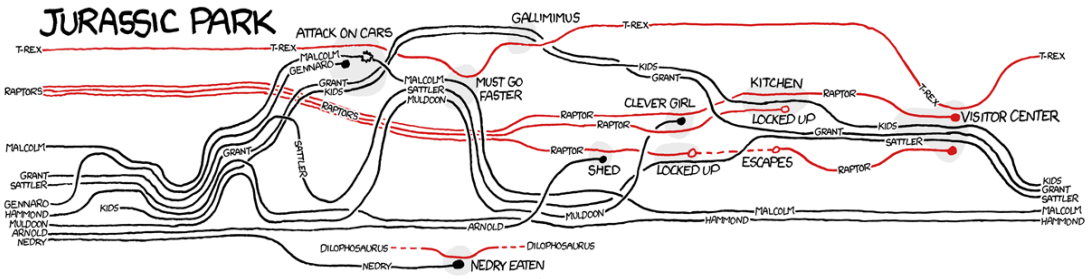

Fig. 1: Storyline visualization for *Jurassic Park* [1].

1 Introduction

A storyline visualization is a visual representation of a story. It represents characters and interactions between characters as time passes in the story. Each character is drawn as an x-monotone curve in the plane where the x-axis represents the flow of time within the narrative. An interaction of characters over a specific time span is represented by a corresponding region in the plane where the curves representing the characters are drawn closely together. This style of representation was introduced by Munroe [1] who visualized several movies in this way. Figure 1 shows one of these drawings.

While this concept of storyline originally comes from visualizing the stories of movies it can be applied to more general settings that have entities that interact with each other.

We present an algorithm to automatically create storyline visualizations. We formulate the *Storyline Visualization Problem* as an optimization problem. Minimizing crossings in drawings is often considered as a natural way to improve the aesthetics. We will however not consider pairwise crossings of curves but focus on *block crossings* instead. This type of crossing was first introduced by Fink et al. [2] for visualizing metro maps. In simple terms a block crossing consists of two sets of locally parallel curves that intersect with no other curves in the crossing area. Figure 2 shows an example of a block crossing. The idea of these block crossings comes from the assumption that structured crossings within a confined region are easier to interpret as the same number of crossings scattered across the drawing.

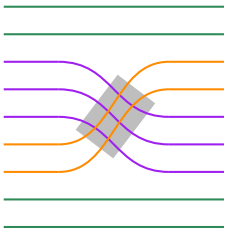

Fig. 2: A single block crossing.

Modeling Decisions. As the *Storyline Visualization Problem* stems from the desire to create visually pleasing drawings, the literature does not fully agree on what aspects of a drawing should be part of the problem. One such aspect is whether it is possible for meetings to occur at overlapping time intervals. One could argue that storyline should represent the order of events as they appear in the respective medium. The storyline for a movie for instance would in this case visualize the ordered sequence of scenes. In this case overlapping meetings are not necessary. A different approach is to visualize

the spatio-temporal structure of the story. Figure 1, showing a storyline visualization of *Jurassic Park*, for example times the events of the movie as they happened in the story and not as they appeared in the screenplay. Another common example are flashbacks that show things in the movie after they've actually occurred in the story. As in previous work [3, 4], we will focus on the second approach.

Another aspect is whether characters are allowed to appear or disappear during the time span of a storyline visualization, that is, whether all curves span the entire extent of the x-axis. Characters that are of less importance to the story for example because they are present in few meetings only could be omitted during their idle time or even omitted entirely. Further some characters that have a strong relationship with each other within the story could be summarized to one single character. Both of these techniques are present in Figure 1. For example, the character "Nerdy" dies and is not drawn afterwards, while the character "Kids" represents multiple characters.

Previous Work. Tanahashi and Ma [5] computed storyline visualizations automatically and defined three metrics to measure the quality of visualizations. One such metric is the minimization of *wiggles*, which are deviations in curves that disrupt the visual flow. Minimizing such wiggles was discussed by Fröschl [6].

Another metric is the minimization of intersections of curves. This problem was formalized for storyline visualizations by Kostitsyna et al. [7]. They did however discuss the problem in context of minimizing pairwise crossings. They proved the problem NP-hard, presented an FPT algorithm and gave an upper bound on the number of crossings in a restricted setting. Gronemann et al. [8] developed an integer linear program (ILP) to solve this pairwise crossing minimization problem. They were able to solve instances form real-world movies and books which we will take a closer look at as well in Section 4. Di Giacomo et. al. [9] allowed characters to take part in multiple meetings at the same time.

In an earlier paper [3] we introduced the concept of minimizing block crossings for storyline visualizations. We showed that block crossing minimization in storylines is NP-hard. For special cases we provided an approximation algorithm. We designed two exact algorithms, one of which is fixed-parameter tractable (FPT) in the number of characters.

In a more recent paper [4] we developed a SAT-based algorithm. The current work improves this SAT formulation to handle more general cases of the block crossing minimization problem better. While the previous formulation was able to model "dying" characters and characters being "born", this modeling decision negatively influences the runtime of the algorithm. We improve on that aspect.

Problem Definition. The definition of the problem closely follows the problem statement from previous work [4]. We allow characters to have lifespans, that is characters can have time spans in which they are not drawn. We further allow concurrent meetings, that is meeting can overlap.

A storyline $\mathcal{S}$ is a triple (C, M, E) with $C = \{1, \ldots, k\}$ being a set of *characters*, $M = \{m_1, \ldots, m_n\}$ is a set of *meetings* and $E : C \to \mathcal{P}(\mathbb{N})$ maps a character to a set

of intervals of natural numbers $\mathbb{IN}$. This set $E(i) = \{[b_i^1, d_i^1], \ldots, [b_i^{\eta_i}, d_i^{\eta_i}]\}$ contains η_i disjoint time intervals in which i is alive. We call $E(i)$ the *lifespan* of character i and call i *alive* during its lifespan, otherwise we call the character a *ghost*. For each interval in $E(i)$ we call b_i^r and d_i^r a *birth* and *death* of i respectively. A meeting m_j is triple (s_j, e_j, C_j) where $s_j \in \mathbb{N}$ is the start time of the meeting, $e_j \in \mathbb{N}$ is the end time of the meeting and $C_j \subseteq C$ the set of characters that are involved in m_j with $s_j \leq e_j$. We call a meeting m_j *active* at time t if $t \in \{s_j, \ldots, e_j\}$.

We forbid two overlapping meetings m_j, m_k to contain the same characters, that is $C_j \cap C_k = \emptyset$ if $s_j \leq s_k \leq e_j$. Further a character i can only participate in a meeting m_j during its lifespan, that is $\exists [b_i^x, d_i^x] \in E(i)$ with $s_j \geq b_i^x$ and $e_j \leq d_i^x$.

A solution for a storyline instance $\mathcal{S} = (C, M, E)$ consists of a sequence $\Pi = [\pi_1, \ldots, \pi_\lambda]$ of permutations of C and a non-decreasing function $A : \mathbb{R} \to \{1, \ldots, \lambda\}$ that maps points in time to permutations in the solution. A solution is *admissible* if the following conditions hold true:

- For any point in time $t \in \mathbb{R}$, for any meeting m_j that is active on time t, the characters in C_j must form a contiguous block on $\pi_{A(t)}$. Ghost characters do not disturb the connectivity of this block.
- For any $p \in \{2, \ldots, \lambda\}$ the permutations π_{p-1} and π_p must either differ in exactly one *block crossing* or must be equal $\pi_p = \pi_{p-1}$.

A block crossing between two permutations π_x and π_{x+1} is a transposition of two adjacent block of characters. Consider after renumbering $\pi_x = \langle 1, \ldots, a, \ldots, b, \ldots, c, \ldots, k \rangle$. An example for a block crossing would be switching the blocks $\langle a, \ldots, b \rangle$ and $\langle b+1, \ldots, c \rangle$, which results in the permutation $\pi_{x+1} = \langle 1, \ldots, a-1, b+1, \ldots, c, a, \ldots, b, c+1, \ldots, k \rangle$.

These definitions allow us to now formalize the *Storyline Block Crossing Minimization Problem*: Given a storyline $\mathcal{S} = (C, M, E)$ find an admissible solution (Π, A) that minimizes the number of block crossings. This implies that all subsequent permutations in Π must differ by exactly one block crossing.

Our results. As a continuation of our SAT formulation from previous work [4] we revise the formulation to achieve faster running times; see Section 2. We discuss the theoretical improvements of this new approach in Section 3 and experimentally compare the new approach to the one from previous work in Section 4.

As a secondary goal we implemented the model for the improved SAT formulation with easy to use interfaces. The source code of this implementation is available online under a free license[1].

[1] `https://github.com/acreter/storylinesSAT`

2 Model

This section describes how to construct an instance of SAT that encodes whether for a given storyline $\mathcal{S}$ and an integer λ there exists an admissible solution for $\mathcal{S}$ that uses exactly λ permutations of the set of characters C. If such a solution should not exist, the instance of SAT will not have a valid truth assignment. By searching for the minimum satisfiable λ we can then derive a solution to the Storyline Block Crossing Minimization Problem. In our implementation in Section 4 we use linear search.

We will not always describe the clauses for the SAT instance in conjective normal form. We will instead use other operators whenever it improves readability. The transformation to conjective normal form is straightforward however.

Describing the Permutations. To describe the permutations of C we will use Boolean variables x_{ij}^r that describe the relative position of characters i and j on permutation π_r. If x_{ij}^r is assigned true then i is above j on π_r. The relative position of two characters must the unique (Antisymmetry):

$$x_{ij}^r \iff \neg x_{ji}^r \qquad \forall i, j \in C,\, r \in \{1, \ldots, \lambda\}. \tag{1}$$

Further the relative order of three characters must not contain a cycle (Transitivity):

$$x_{ij}^r \wedge x_{jk}^r \implies x_{ik}^r \qquad \forall i, j, k \in C,\, r \in \{1, \ldots, \lambda\}. \tag{2}$$

Block Crossings. By the problem definition, there can be at most one block crossing between two permutations. We describe a block crossing as two sets $\mathcal{T}_r \subset C$ and $\mathcal{B}_r \subset C$. These are the two sets of characters that cross between π_r and π_{r+1}. We express the membership of a character i for the subsets using variables of type t_i^r and b_i^r. A character cannot be in the same subset on the same permutation $\mathcal{T}_r \cap \mathcal{B}_r = \emptyset$:

$$\neg(t_i^r \wedge b_i^r) \qquad \forall i \in C,\, r \in \{1, \ldots, \lambda\}. \tag{3}$$

If one character is a member of subset $\mathcal{T}_r$ and another character is a member of subset $\mathcal{B}_r$ on the same permutation, then the two characters must cross:

$$t_i^r \wedge b_j^r \wedge x_{ij}^r \implies \neg x_{ij}^{r+1} \qquad \forall i, j \in C\ r \in \{1, \ldots, \lambda - 1\}. \tag{4}$$

All characters of subset $\mathcal{T}_r$ must be above the characters of subset $\mathcal{B}_r$. This also guarantees, that only members of $\mathcal{T}_r$ and $\mathcal{B}_r$ cross:

$$x_{ij} \wedge \neg x_{ij}^{r+1} \implies t_i^r \wedge b_j^r \qquad \forall i, j \in C,\, r \in \{1, \ldots, \lambda - 1\}. \tag{5}$$

Members of both $\mathcal{T}_r$ and $\mathcal{B}_r$ must be coherent:

$$x_{ij}^r \wedge x_{jk}^r \wedge t_i^r \wedge t_k^r \implies t_j^r \qquad \forall i, j, k \in C,\, r \in \{1, \ldots, \lambda - 1\}, \tag{6}$$

$$x_{ij}^r \wedge x_{jk}^r \wedge b_i^r \wedge b_k^r \implies b_j^r \qquad \forall i, j, k \in C,\, r \in \{1, \ldots, \lambda - 1\}, \tag{7}$$

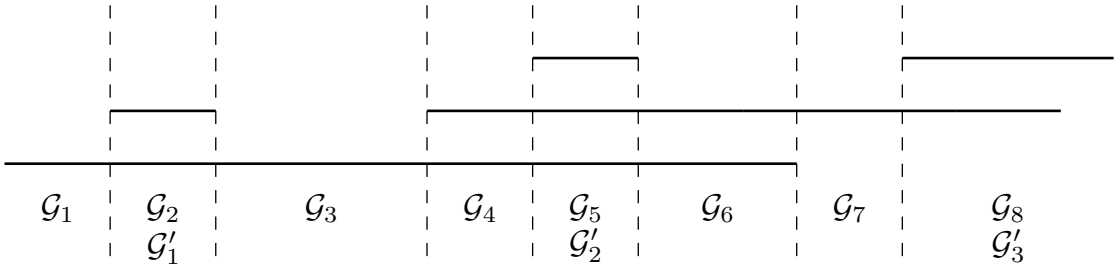

Fig. 3: Example for meeting groups. All horizontal lines represent meetings.

and must neighbor:

$$x_{ij}^r \wedge x_{jk}^r \wedge t_i^r \wedge b_k^r \implies b_j^r \vee t_j^r \qquad \forall i,j,k \in C,\, r \in \{1,\ldots,\lambda-1\}. \tag{8}$$

Meeting Groups. The last step to construct the SAT instance is to connect the meetings with the permutations. To model meetings we reuse the concept of *meetings groups* from previous work [4]. A meeting group $\mathcal{G}$ is a subset $\mathcal{G} \subseteq M$, that contains meetings that are active at the same time. We say a meeting group $\mathcal{G}$ *fits* on a permutation π_r, if the characters of each meeting $m \in \mathcal{G}$ form a contiguous block on π_r without considering ghost characters. Note that meeting groups have a natural order by time. Whenever a meeting starts or ends, a new meeting group begins. Let $\mathcal{M} = [\mathcal{G}_1, \ldots, \mathcal{G}_\mu]$ be this sequence of meeting groups of a given storyline $\mathcal{S}$.

If a meeting group $\mathcal{G}_y \in \mathcal{M}$ fits on a permutation π_r, then all subsets of $\mathcal{G}_y$ will also fit on π_r. It is therefore sufficient to only consider a meeting group $\mathcal{G}_y$ if $\mathcal{G}_y \not\subset \mathcal{G}_{y-1}$ and $\mathcal{G}_y \not\subset \mathcal{G}_{y+1}$. We call the sequence of meeting groups with this property $\mathcal{M}' = [\mathcal{G}'_1, \ldots, \mathcal{G}'_\nu]$. Consider a storyline as a function $f : \mathbb{R} \to \mathbb{N}$ that maps points in time to the number of meetings active at that time, then $\mathcal{M}'$ would reflect the local maxima of that function. Figure 3 shows an example for meeting groups.

Meeting groups are solely used to construct the SAT instance. A satisfying assignment is later transformed back into a solution for the given storyline $\mathcal{S}$. To make use of these meeting groups $\mathcal{M}'$, we introduce variables of type q_ℓ^r that map a meeting group $\mathcal{G}'_\ell$ to a permutation π_r.

A meeting group $\mathcal{G}'_\ell$ must be assigned to at least one permutation:

$$\bigvee_{r=1}^{r \le \lambda} q_\ell^r \qquad \forall \ell \in \{1,\ldots,\nu\}. \tag{9}$$

Further a meeting group $\mathcal{G}'_\ell$ can only be assigned to a permutation π_r, if $\mathcal{G}'_{\ell-1}$ is assigned to a permutation π_p with $p \in \{1,\ldots,r\}$:

$$\bigvee_{j=1}^{j \le p} \left(q_{\ell-1}^j\right) \vee \neg q_\ell^p \qquad \forall \ell \in \{2,\ldots,\nu\}, \forall p \in \{1,\ldots,\lambda\}. \tag{10}$$

To guarantee that meetings form a contiguous block, we first need to know which characters are alive during the time span of a meeting group. To achieve this we model each character as a meeting with one member, that is the character itself. We call these meetings *single-member meetings*. Note that this changes the sequence of meeting groups $\mathcal{M}'$. We can now define the set of ghosts $\mathcal{O}_\ell$ for a meeting group $\mathcal{G}'_\ell$ as $\mathcal{O}_\ell = \{x \in C \mid \nexists m_j \in \mathcal{G}'_\ell \text{ with } x \in C_j\}$ and subsequently the type of clauses that guarantee that meetings form a contiguous block when they are active:

$$q^r_\ell \implies x^r_{kj} \vee x^r_{ji} \qquad \begin{array}{l} \forall i,j,k \in C,\ r \in \{1,\dots,\lambda\},\ \mathcal{G}_\ell \in \mathcal{M}', \\ \exists m_x \in \mathcal{G}_\ell \text{ with } i,k \in C_x \text{ and } j \notin C_x, \\ j \notin \mathcal{O}_\ell. \end{array} \tag{11}$$

Inter Groups. Clauses 11 only force coherent meetings if the corresponding variable q is assigned true. This however introduces a problem whenever there are two subsequent meeting groups $\mathcal{G}'_\ell$ and $\mathcal{G}'_{\ell+1}$ that require multiple block crossings in an optimal solution, and therefore multiple permutations, between them. Let π_r be a permutation with q^r_ℓ true and π_p a permutation with $q^p_{\ell+1}$ true and $r+2 = p$. If neither $\mathcal{G}'_\ell$ nor $\mathcal{G}'_{\ell+1}$ fit on π^{r+1} then it might occur that no meeting group can fit on π^{r+1}. If however no meeting group is active on π^{r+1}, that is q^{r+1}_j is assigned false $\forall j \in \{1,\dots,\nu\}$, then meeting coherency is not guaranteed. If however $\mathcal{G}'_\ell$ and $\mathcal{G}'_{\ell+1}$ share meetings, that is $\mathcal{G}'_\ell \cap \mathcal{G}'_{\ell+1} \neq \emptyset$, then all meetings $m \in \mathcal{G}'_\ell \cap \mathcal{G}'_{\ell+1}$ should be coherent on π^{r+1}.

To address this problem we introduce *inter groups* which are special meeting groups defined as follows: For each pair of meeting groups $\mathcal{G}'_\ell, \mathcal{G}'_{\ell+1} \in \mathcal{M}'$ let $\mathcal{I}_\ell = \mathcal{G}'_\ell \cap \mathcal{G}'_{\ell+1}$ be the inter group between $\mathcal{G}'_\ell$ and $\mathcal{G}'_{\ell+1}$. Let $\mathcal{I} = \{\mathcal{I}_1,\dots,\mathcal{I}_{\nu-1}\}$ be the set of inter groups. We use variables of type z^r_ℓ to map inter group $\mathcal{I}_\ell$ to permutation π^r and force the variables to be assigned true on all permutations between π_r and π_p with q^r_ℓ and $q^p_{\ell+1}$ true:

$$(q^r_\ell \wedge q^p_{\ell+1}) \implies \bigwedge_{a=r}^{a \le p} z^a_\ell \qquad \begin{array}{l} \forall \ell \in \{1,\dots,\nu-1\} \\ r,p \in \{1,\dots,\lambda\} \text{ with } r \le p. \end{array} \tag{12}$$

To force all meeting to form a contiguous block we define the set of ghost characters $\mathcal{O}'_\ell$ for an inter group $\mathcal{I}_\ell$ analogical to the meeting groups as $\mathcal{O}'_\ell = \{x \in C \mid \nexists m_j \in \mathcal{I}_\ell \text{ with } x \in C_j\}$. We can then apply Clauses 11 analogical to the variables z^r_ℓ:

$$z^r_\ell \implies x^r_{kj} \vee x^r_{ji} \qquad \begin{array}{l} \forall i,j,k \in C,\ r \in \{1,\dots,\lambda\},\ \mathcal{I}_\ell \in \mathcal{I}, \\ \exists m_x \in \mathcal{I}_\ell \text{ with } i,k \in C_x \text{ and } j \notin C_x, \\ j \notin \mathcal{O}'_\ell. \end{array} \tag{13}$$

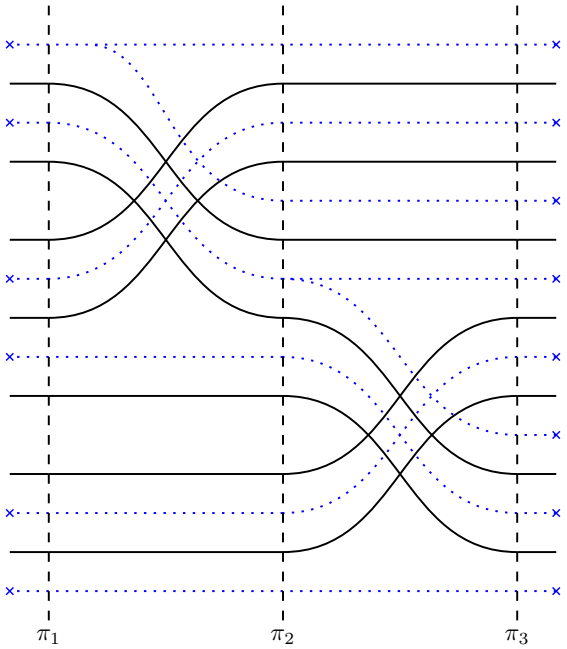

Fig. 4: Read the figure from right to left. The dotted lines represent a character that first appears on permutation π_3. All positions on π_3 can be reached by using the existing block crossing structure and inserting the character on the right position on π_1

Characters with multiple alive intervals. Whenever a character is born, it is possible to insert it on any position on the y-axis of the drawing. This might however be complicated for characters with multiple alive intervals. Consider a character $i \in C_j$ for some meeting m_j and $m_j \in \mathcal{G}'_\ell$ with q^r_ℓ assigned true. Further let $i \in C_k$ for some other meeting m_k and $m_k \in \mathcal{G}'_{\ell+1}$ with $q^{r+1}_{\ell+1}$ assigned true. If i dies at time e_j and gets born again at time s_k, then i can only use the block crossing between π^r and π^{r+1} to be assigned a position in π^{r+1}, which does not correspond to an arbitrary position.

We therefore model characters with multiple alive intervals as multiple characters. To insert a character at an arbitrary position on the permutation on which it first occurs, we now just have to place the character at the appropriate position on the first permutation and "ride along" the existing block crossing structure. Remember that Clauses 11 and 13 do not consider ghost characters. A character that uses the existing block crossing structure will therefore never disturb any meetings. Figure 4 shows an example.

This concludes the SAT formulation. By searching for the minimal number of permutations λ for which the resulting formula has a satisfying assignment we find a solution to the storyline block crossing minimization problem. The permutations can be extracted using the relative ordering variables x^r_{ij}. The function A that maps time to these permutations can be found by remembering which meeting groups maps to which point in time by using the meetings in the meeting groups. The same can be done with all inter groups.

3 Comparison of the Models

We will refer to the model from Section 2 as SAT_{new} and to the model from previous work [4] as SAT_{old}. Before experimentally evaluating SAT_{new} we will take a closer look at the theoretical improvements we get over SAT_{old}. One very simple metric to evaluate an instance of SAT is to count the number of variables the solver has to decide and count the number clauses that restrict the solver. The model SAT_{new} was mainly constructed by removing redundant information SAT_{new}. This section will therefore primarily focus on what is missing in SAT_{new}.

ABC-Blocks SAT_{new} categorized the lines into blocks $\mathcal{A}_r$, $\mathcal{B}_r$ and $\mathcal{C}_r$ to ensure that only a valid block crossing can occur between permutations π_r and π_{r+1}. Specifically the characters in $\mathcal{B}_r$ and $\mathcal{C}_r$ and only those must cross. These correspond to the blocks $\mathcal{T}_r$ and $\mathcal{B}_r$ in SAT_{new}. The $\mathcal{A}_r$ block contained all characters that are not part of the block crossing. This behavior can be modeled as not being part of either $\mathcal{B}_r$ or $\mathcal{C}_r$ and is therefore redundant.

Ghost variables SAT_{old} used special variables d_i^r that denoted whether a character i is a ghost on permutation π_r. It was needed to release all clauses should one of the characters in that clause be a ghost. In SAT_{new} we saw that this information is already encoded in the meeting groups. Specifically we constructed the set of ghost characters $\mathcal{O}_\ell$ for each meeting group $\mathcal{G}'_\ell$.

Since those ghost variables d_i^r are defined per permutation and multiple meeting groups could be assigned the same permutation, it could happen that meeting groups tried to force different dead/alive states onto the characters of a permutation. SAT_{old} therefore did not allow two meeting groups containing different sets of ghost characters to be assigned on the same permutation. This however made it necessary to introduce special permutations for the birth and death of characters which increases the number of permutations required. In SAT_{new} on the other hand the number of permutations directly corresponds to the number of block crossings.

Fewer meeting groups We have seen that it is unnecessary for meeting groups $\mathcal{G}_\ell$ which are subsets of either $\mathcal{G}_{\ell+1}$ or $\mathcal{G}_{\ell-1}$ to be considered by the model, since a permutation π_r which can fit $\mathcal{G}_{\ell+1}$ can also fit $\mathcal{G}_\ell$ if $\mathcal{G}_\ell \subset \mathcal{G}_{\ell+1}$. We called the sequence of these meeting groups which are no subset $\mathcal{M}'$. SAT_{old} however used all meeting groups, which we denoted as $\mathcal{M}$ in Section 2.

4 Experiments

In this section we will describe a series of experiments to evaluate the model from Section 2 which we refer to as SAT_{new}. The approach in previous work [4] will be denoted as SAT_{old}. Besides the usual analysis of time and memory consumption, we will additionally use our SAT-based algorithm to experimentally investigate the properties of certain

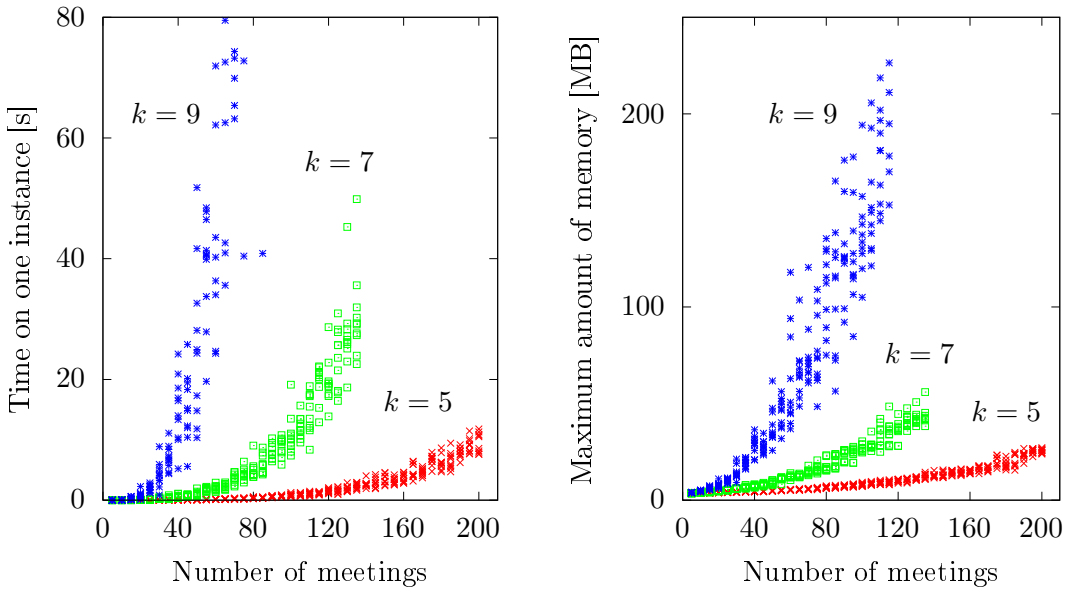

Fig. 5: Left: runtime of SAT on uniform random instances. Right: memory usage of SAT on uniform random instances. Both plots show results for different numbers of characters k.

classes of random instances. Further a brief comparison of two different SAT-Solvers is given.

Implementation Details. The implementation of SAT_{new} is written in C. The code for all SAT-Solvers used in this section is in C++. The implementation of SAT_{old} used Python code that writes CNF SAT instance files in DIMACS format which created an overhead by using the filesystem. Hence their measured time included the solver reading and parsing the input file, but not the time needed to build the instance. Our implementation of SAT_{new} bypasses that overhead by interfacing directly with the solver and linking it to one single binary. To conduct a fair comparison we will therefore measure the running time of the entire program and not just the "solving" part. In contrast to SAT_{old} however we use linear search to find the minimum number of layers , since we can expect to solve SAT_{new} with fewer layers. This is due to SAT_{new} not needing to insert layers to model the births and deaths of characters as described in Section 3. The number of layers directly corresponds to the number of block crossings. The solver used for all experiments is *minisat* [10] version 2.2.0 unless stated otherwise.

All experiments have been performed on an Intel® Core™ i5-4670K CPU running at 3.40 GHz with 8 GB RAM. This is somewhat comparable to the Intel® Core™ i5-2400 at 3.10 GHz with 8 GB RAM setup of van Dijk et al. [4]. The implementations of both SAT_{new} and SAT_{old} are single-threaded.

	Gronemann et al. [8]		SAT_{new}		SAT_{old} [4]
	cr	Time [s]	*bc*	Time [s]	Time [s]
Inception	35	2.02	12	0.29	1.54
Star Wars	39	0.99	10	0.32	3.77
The Matrix	12	0.77	4	0.05	2.86

Tab. 1: Comparison of minimal number of pairwise crossings (cr), minimal number of block crossings (bc) and running times. Note the running time improvement of SAT_{new} over SAT_{old}.

Real-World Instances. We test the implementation of SAT_{new} on the real-world instances used by Gronemann et al. [8] and van Dijk et al. [4]. These consist of three movies and three books. Some of the books are split into individual chapters.

While it was already possible to solve the movie instances in reasonable time using SAT_{old}, Table 1 shows an improvement in running time by using SAT_{old}. The book instances however were not solvable in two minutes or less with SAT_{old}. This was mainly due to the high number of characters that either "died" or were "born" during the instance. Section 3 already showed that these "births" and "deaths" are handled better in SAT_{new}. Table 2 confirms that: This can especially be seen by looking at the single chapter instances *anna1* to *anna8* and *jean1* to *jean5*, which are solvable with few block crossings. The implementation of SAT_{new} solves these in an expectantly small amount of time whereas the implementation of SAT_{old} could not solve them in two minutes or less. It is however also apparent that due to the large amount of characters some instances are still not solvable.

Random Instances. We will use random instances of two kinds as in previous work [3, 4]. The first are *uniform* instances that are generated as follows: Pick the number of characters k, the number of meetings n and a probability p. Then create a meeting by deciding independently at random with probability p whether each character is in the meeting. Repeat this process until the number of desired meetings n is reached while discarding meetings with fewer than two participants. We use probability $p = 0.5$ for all instances unless stated otherwise. Note that in the instances van Dijk et al. [3] used for their tests, all characters are alive at all times. This decision was made to be able to run all of their algorithms on the same instances. We can therefore expect to have only a small improvement over their results since the main advantage of SAT_{new}, as seen with the real-world instances, does not have any effect in this case.

Figure 5 shows the runtime of SAT_{new} for various number of characters. For every number of meetings we generate ten instances and therefore get ten data points. It is apparent that the runtime for $k = 7$ and $k = 9$ explodes at 100 meetings and 30 meetings respectively. For $k = 5$ a slight nudge upwards at 160 meetings can be observed. For memory consumption something similar can be seen, with the exception of $k = 7$ which seems to grow at a slightly lower rate.

Before moving on we will now conduct a general analysis on uniform random instances. Figure 6 shows the optimal number of block crossings for multiple numbers of characters.

	Gronemann et al. [8]		SAT_{new}		
	cr	Time [s]	*bc*	Time minisat [s]	Time CaDiCal [s]
anna1	20	13.03	9	2.86	6.11
anna2	12	0.88	5	0.69	1.52
anna3	0	0.01	0	0.06	0.13
anna4	20	4.86	8	0.87	1.96
anna5	17	2.60	7	3.45	7.41
anna6	31	3.89	12	3.78	8.06
anna7	9	7.88	5	1.56	3.82
anna8	6	0.15	3	0.03	0.10
anna3-4	34	12.66	13	22.41	
anna7-8	32	19.16	10	12.87	42.51
huck	42	111.31	14	45.59	
jean1	10	0.90	6	1.22	2.56
jean2	6	0.08	5	0.03	0.11
jean3	13	6.31	7	1.07	2.60
jean4	42	22.22	10	6.46	11.57
jean5	17	1.52	9	0.33	0.73
jean1-2	20	3.93	13	7.83	18.28
jean2-3	33	48.90	16	10.01	24.49

Tab. 2: Comparison of minimal number of pairwise crossings (cr), minimal number of block crossings (bc) and running time on the book instances. Individual chapters are denoted by their corresponding chapter number. The runtime cutoff is 90 seconds.

With increasing number of meetings, these plots seem to have linear growth, with the number of characters influencing the rate of growth. The probability p, which influences the expected size of all meetings is also an interesting factor. It seems that the instances generally require the most block crossings at $p = 0.5$ and fewer for $p \leq 0.4$ and $p \geq 0.6$. This can be seen in Figure 6 left.

The second kind of random instances require few block crossings. They are generated as follows: Let $OPT_{\max}$ be the maximum number of block crossings, $\mathcal{C}$ the set of characters, n the number of meetings and p a probability. Starting from an arbitrary order of $\mathcal{C}$ sample $OPT_{\max}$ uniformly-random block crossings to generate a sequence of $OPT_{\max}+1$ permutations. To create a meeting, choose one permutation at random and then c adjacent characters from this permutation at random, where c is binomially distributed with probability p so as to match the uniform model. Repeat this process n times. Now put the meetings in order of the permutation they appear on. This generates the sequence of meetings. These instances are guaranteed to have a solution with at most $OPT_{\max}$ block crossings.

Since for SAT_{new} we search for the minimum number of block crossings linearly, a small number of required block crossings should benefit the runtime of the implementation. Figure 7 shows that this is indeed the case. Even for $k = 13$ no runtime explosion can be observed within the first 200 numbers of meetings. Note that SAT_{new} experi-

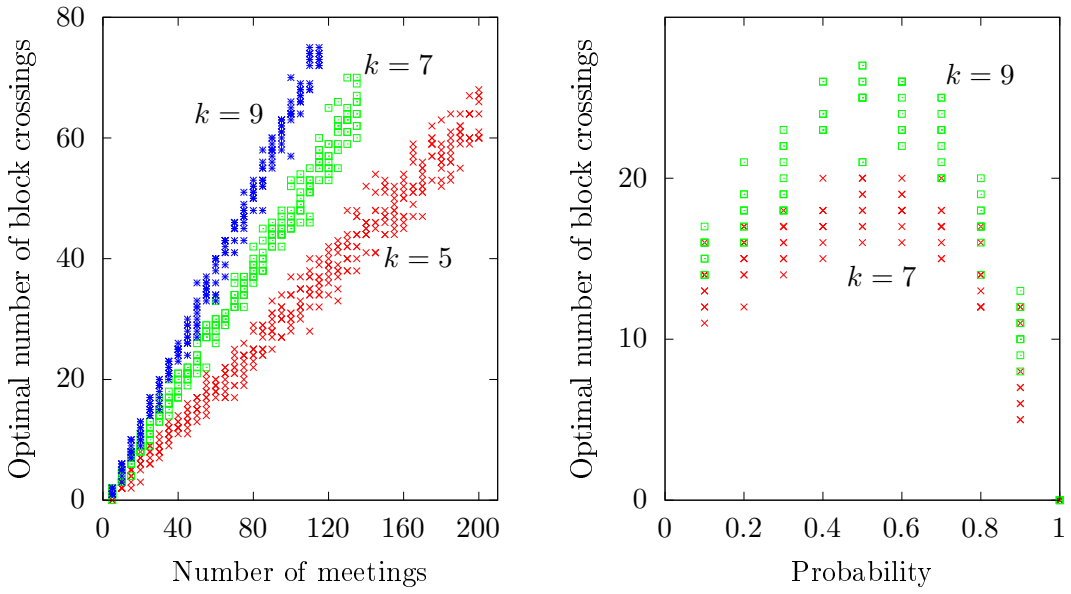

Fig. 6: Left: optimal number of block crossings for uniform random instances. Right: optimal number of block crossings for uniform random instances with varying probability p as described in the text.

ences difficulty for instances with few meetings. These instances can be solved with few block crossings as seen in Figure 7 right. For a small number of meetings the random instances that require few block crossings closely resemble uniform random instances, which explains the runtime spike.

Before we conclude the experiments we will compare two CDCL SAT solvers. Our implementation of SAT_{new} was designed to be solver agnostic and can be used with most modern SAT solvers with little modification. The first solver is *minisat* [10] version 2.2.0 which was used for all experiments in this section. The second one is *CaDiCaL* [11] version 1.3.0. Figure 8 shows that minisat has an edge over CaDiCaL on uniform random instances and practically dominates CaDiCaL. Table 2 shows that this is equally true for the tested real world instances.

Concluding Remarks In this section we have seen that SAT_{new} has a clear edge over SAT_{old} regarding the tested real world instances. This was due to the improvements of SAT_{new} described in Section 3. For the random instances no clear improvement can be observed. This was however expected since the random instances generated do not make use of dying characters and characters being born within an instance. However SAT_{new} does not perform worse than SAT_{old} on these instances either. It is therefore advisable that SAT_{new} should be used over SAT_{old} for all instances.

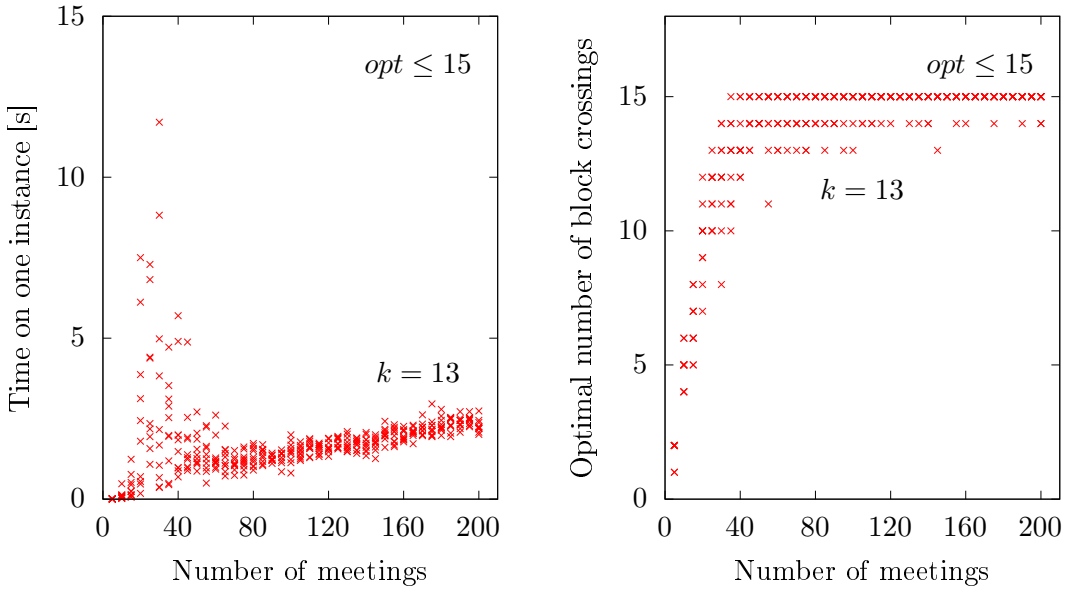

Fig. 7: Left: runtime of SAT with $opt \leq 15$ and number of characters $k = 13$. Right: optimal number of block crossings of the same instances as seen in the left plot.

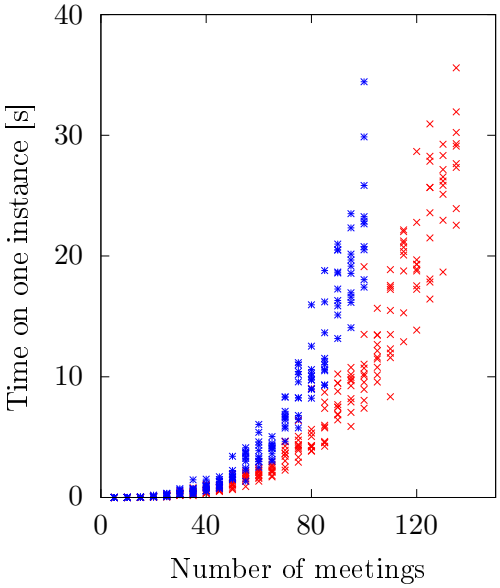

Fig. 8: Runtime of CaDiCaL (stars) and minisat (crosses) for number of characters $k = 7$.

5 Conclusion

We have presented a SAT-based algorithm to produce block crossing optimal drawings of storylines. This algorithm achieved a generally faster runtime than the SAT-based approach from previous work [4], especially for real-world instances. We also described a set of experiments on uniform random instances and random instances with a small amount of block crossings required. We've seen the optimal number of block crossings for uniform random instances seems to grow linearly with the number of meetings. The number of characters influence the rate of growth. Further uniform random instances with meetings larger than half the number of characters require less block crossings. This is equally true for meetings smaller than half the number of characters. Random instances with a small number of required block crossings are generally easier to solve than uniform random instances.

We are unaware of any heuristic techniques that produce non-optimal drawings of storylines using block crossings. This might be good direction for future work. One could also combine the concept of block crossings with different approaches to produces visually pleasing drawings of storylines, such as minimizing wiggles, pairwise crossings or white space gaps.

While it is intuitive from a graphic design perspective to use block crossings, we are unaware of research that investigate whether block crossings are a good metric for creating nice drawings, or if there are any trade-offs.

References

[1] R. Munroe, "Movie narrative charts." `https://xkcd.com/657/`, 2013. accessed 17.11.2020.

[2] M. Fink, S. Pupyrev, and A. Wolff, "Ordering metro lines by block crossings," *Journal of Graph Algorithms and Applications*, vol. 19, no. 1, pp. 111–153, 2015.

[3] T. C. van Dijk, M. Fink, N. Fischer, F. Lipp, P. Markfelder, A. Ravsky, S. Suri, and A. Wolff, "Block crossings in storyline visualizations," in *Graph Drawing and Network Visualization: 24th International Symposium, GD 2016, Athens, Greece, September 19-21, 2016, Revised Selected Papers* (Y. Hu and M. Nöllenburg, eds.), vol. 9801 of *Lecture Notes in Computer Science*, (Cham), pp. 382–398, Springer, 2016.

[4] T. C. van Dijk, F. Lipp, P. Markfelder, and A. Wolff, "Computing storyline visualizations with few block crossings," in *Graph Drawing and Network Visualization: 25th International Symposium, GD 2017, Boston, MA, USA, September 25-27, 2017, Revised Selected Papers* (F. Frati and K.-L. Ma, eds.), vol. 10692 of *Lecture Notes in Computer Science*, (Cham), pp. 365–378, Springer, 2018.

[5] Y. Tanahashi and K. Ma, "Design considerations for optimizing storyline visualizations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 12, pp. 2679–2688, 2012.

[6] T. Fröschl, "Minimizing wiggles in storyline visualizations," Master's thesis, 2018.

[7] I. Kostitsyna, M. Nöllenburg, V. Polishchuk, A. Schulz, and D. Strash, "On minimizing crossings in storyline visualizations," in *Graph Drawing and Network Visualization: 23rd International Symposium, GD 2015, Los Angeles, CA, USA, September 24-26, 2015, Revised Selected Papers* (E. Di Giacomo and A. Lubiw, eds.), (Cham), pp. 192–198, Springer, 2015.

[8] M. Gronemann, M. Jünger, F. Liers, and F. Mambelli, "Crossing minimization in storyline visualization," in *Graph Drawing and Network Visualization: 24th International Symposium, GD 2016, Athens, Greece, September 19-21, 2016, Revised Selected Papers* (Y. Hu and M. Nöllenburg, eds.), vol. 9801 of *Lecture Notes in Computer Science*, (Cham), pp. 367–381, Springer, 2016.

[9] E. Di Giacomo, W. Didimo, G. Liotta, F. Montecchiani, and A. Tappini, "Storyline visualizations with ubiquitous actors," in *Graph Drawing and Network Visualization: 28th International Symposium, GD 2020, Vancouver, BC, Canada, September 16–18, 2020, Revised Selected Papers* (D. Auber and P. Valtr, eds.), vol. 12590 of *Lecture Notes in Computer Science*, (Cham), pp. 324–332, Springer, 2020.

[10] N. Eén and N. Sörensson, "An extensible sat-solver," in *International conference on theory and applications of satisfiability testing: 6th International Conference, SAT 2003, Santa Margherita Ligure, Italy, May 5-8, 2003, Selected Revised Papers* (E. Giunchiglia and A. Tacchella, eds.), vol. 2919 of *Lecture Notes in Computer Science*, (Berlin, Heidelberg), pp. 502–518, Springer, 2003. `http://minisat.se`.

[11] A. Biere, K. Fazekas, M. Fleury, and M. Heisinger, "CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020," in *Proceedings of SAT Competition 2020 – Solver and Benchmark Descriptions* (T. Balyo, N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda, eds.), vol. B-2020-1 of *Department of Computer Science Report Series B*, pp. 51–53, University of Helsinki, 2020.