Block Crossings in Storyline Visualizations

Thomas C. van Dijk[1], Martin Fink[2], Norbert Fischer[1], Fabian Lipp[1](✉), Peter Markfelder[1], Alexander Ravsky[3], Subhash Suri[2], and Alexander Wolff[1]

[1] Lehrstuhl für Informatik I, Universität Würzburg, Würzburg, Germany
fabian.lipp@uni-wuerzburg.de
[2] University of California, Santa Barbara, USA
{fink,suri}@cs.ucsb.edu
[3] Pidstryhach Institute for Applied Problems of Mechanics and Mathematics, National Academy of Sciences of Ukraine, Lviv, Ukraine
oravsky@mail.ru
http://www1.informatik.uni-wuerzburg.de

Abstract. Storyline visualizations help visualize encounters of the characters in a story over time. Each character is represented by an x-monotone curve that goes from left to right. A meeting is represented by having the characters that participate in the meeting run close together for some time. In order to keep the visual complexity low, rather than just minimizing pairwise crossings of curves, we propose to count *block crossings*, that is, pairs of intersecting bundles of lines.

Our main results are as follows. We show that minimizing the number of block crossings is NP-hard, and we develop, for meetings of bounded size, a constant-factor approximation. We also present two fixed-parameter algorithms and, for meetings of size 2, a greedy heuristic that we evaluate experimentally.

1 Introduction

A storyline visualization is a convenient abstraction for visualizing the complex narrative of interactions among people, objects, or concepts. The motivation comes from the setting of a movie, novel, or play where the narrative develops as a sequence of interconnected scenes, each involving a subset of characters. See Fig. 1 for an example.

The storyline abstraction of characters and events occurring over time can be used as a metaphor for visualizing other situations, from physical events involving groups of people meeting in corporate organizations, political leaders managing global affairs, and groups of scholars collaborating on research to abstract co-occurrences of "topics" such as a global event being covered on the front pages of multiple leading news outlets, or different organizations turning their attention to a common cause.

A storyline visualization maps a set of characters of a story to a set of curves in the plane and a sequence of meetings between the characters to regions in the

The full version of this paper is available on arXiv [3]. Whenever we refer to the *Appendix* we mean the appendix of arXiv:1609.00321v1.

Y. Hu and M. Nöllenburg (Eds.): GD 2016, LNCS 9801, pp. 382–398, 2016.
DOI: 10.1007/978-3-319-50106-2_30

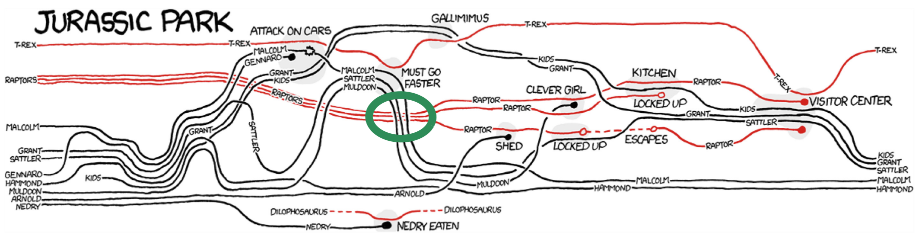


Fig. 1. Storyline visualization for *Jurassic Park* by `xkcd` [11] with a block crossing (highlighted by a bold green ellipse). (Color figure online)

plane where the corresponding curves come close to each other. The current form of storyline visualizations seems to have been invented by Munroe [11] (compare Fig. 1), who used it to visualize, in a compact way, which subsets of characters meet over the course of a movie. Each character is shown as an x-monotone curve. Meetings occur at certain times from left to right. A meeting corresponds to a point in time where the characters that meet are next to each other with only small gaps between them. Munroe highlights meetings by underlaying them with a gray shaded region, while we use a vertical line for that purpose. Hence, a storyline visualization can be seen as a drawing of a hypergraph whose vertices are represented by the curves and whose edges come in at specific points in time.

A natural objective for the quality of a storyline visualization is to minimize unnecessary "crossings" among the character lines. The number of crossings alone, however, is a poor measure: two blocks of "locally parallel" lines crossing each other are far less distracting than an equal number of crossings randomly scattered throughout the drawing. Therefore, instead of pairwise crossings, we focus on minimizing the number of *block crossings*, where each block crossing involves two arbitrarily large sets of parallel lines forming a crossbar, with no other line in the crossing area; see Fig. 1 for an example.

Previous Work. Kim et al. [6] used storylines to visualize genealogical data; meetings correspond to marriages and special techniques are used to indicate child–parent relationships. Tanahashi and Ma [12] computed storyline visualizations automatically and showed how to adjust the geometry of individual lines to improve the aesthetics of their visualizations. Muelder et al. [10] visualized clustered, dynamic graphs as storylines, summarizing the behavior of the local network surrounding user-selected foci.

Only recently a more theoretical and principled study was initiated by Kostitsyna et al. [8], who considered the problem of minimizing pairwise (not *block*) crossings in storylines. They proved that the problem is NP-hard in general, and showed that it is fixed-parameter tractable with respect to the (total) number of characters. For the special case of 2-character meetings without repetitions, they developed a lower bound on the number of crossings, as well as as an upper bound of $O(k \log k)$ when the meeting graph—whose edges describe the pairwise meetings of characters—is a tree.

Our work builds on the problem formulation of Kostitsyna et al. [8] but we considerably extend their results by designing (approximation) algorithms for general meetings—for a different optimization goal: we minimize the number of *block crossing* rather than the number of pairwise line crossings. Block crossings were introduced by Fink et al. [5] for visualizing metro maps.

Problem Definition. A storyline $\mathcal{S}$ is a pair (C, M) where $C = \{1, \ldots, k\}$ is a set of *characters* and $M = [m_1, m_2, \ldots, m_n]$ with $m_i \subseteq C$ and $|m_i| \geq 2$ for $i = 1, 2, \ldots, n$ is a sequence of *meetings* of at least two characters. We call any set $g \subseteq C$ of characters that has at least one meeting, a *group*. We define the *group hypergraph* $\mathcal{H} = (C, \mathit{\Gamma})$ whose vertices are the characters and whose hyperedges are the groups that are involved in at least one meeting. The group hypergraph does not include the temporal aspect of the storyline—it models only the graph-theoretical structure of groups participating in the storyline meetings; it can be built by lexicographically sorting the meetings in M in $O(nk \log n)$ time.

Note that we do not encode the exact times of the meetings: In a given visualization, at any time t, there is a unique vertical order π of the characters. Without changing π by crossings, we can increase or decrease vertical gaps between lines. If a group g forms a contiguous interval in π^t, then we can bring g's lines within a short distance δ_{group} without any crossing, and also make sure that all other lines are at a larger distance of at least δ_{sep}. Since any group must be supported at a time just before its meeting starts, computing an output drawing consists mainly of changing the permutation of characters over time so that during a meeting its group is supported by the current permutation. We therefore focus on changing the permutation by crossings over time, and only have to be concerned about the order of meetings; the final drawing can be obtained by a simple post-processing from this discrete set of permutations.

If $\{\pi_1, \pi_2, \ldots, \pi_k\} = \{1, 2, \ldots, k\}$, then $\langle \pi_1, \pi_2, \ldots, \pi_k \rangle$ is a permutation of length k of C. For $a \leq b < c$, a *block crossing* (a, b, c) on the permutation $\pi = \langle 1, \ldots, k \rangle$ is the exchange of two consecutive blocks $\langle a, \ldots, b \rangle$ and $\langle b+1, \ldots, c \rangle$; see Fig. 2. A meeting m *fits* a permutation π (or a permutation π *supports* a meeting m) if the characters participating in m form an interval in π. In other words, there is a permutation of m that is part of π. If we apply a sequence B of block crossings to a permutation π in the given order, we denote the resulting permutation by $B(\pi)$.

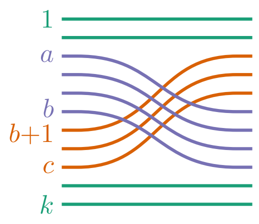


Fig. 2. Block crossing (a, b, c)

Problem 1 (Storyline Block Crossing Minimization (SBCM)). Given a storyline instance (C, M) find a solution consisting of a start permutation π^0 of C and a sequence B of (possibly empty) sequences of block crossings $B_1, B_2, \ldots, B_n$ such that the total number of block crossings is minimized and $\pi^1 = B_1(\pi^0)$ supports m_1, $\pi^2 = B_2(\pi^1)$ supports m_2, etc.

We also consider d-SBCM, a special case of SBCM where meetings involve groups of size at most d, for an arbitrary constant d. E.g., 2-SBCM allows only 2-character meetings, a setting that was also studied by Kostitsyna et al. [8].

Our Results. We observe that a storyline has a crossing-free visualization if and only if its group hypergraph is an interval hypergraph. A hypergraph can be tested for the interval property in $O(n^2)$ time, where n is the number of hyperedges. We show that 2-SBCM is NP-hard (see Sect. 3) and that SBCM is fixed-parameter tractable with respect to k (Sect. 4). The latter can be modified to handle pairwise crossings, where its runtime improves on Kostitsyna et al. [8].

We present a greedy algorithm for 2-SBCM that runs in $O(k^3 n)$ time for k characters. We do some preliminary experiments where we compare greedy solutions to optimal solutions; see Sect. 5. One of our main results is a constant-factor approximation algorithm for d-SBCM for the case that d is bounded and that meetings cannot be repeated; see Sect. 6. Our algorithm is based on a solution for the following NP-complete hypergraph problem, which may be of independent interest. Given a hypergraph $\mathcal{H}$, we want to delete the minimum number of hyperedges so that the remainder is an interval hypergraph. We develop a $(d+1)$-approximation algorithm, where d is the maximum size of a hyperedge in $\mathcal{H}$; see Sect. 7. Finally, we list some open problems in Appendix H.

2 Preliminaries

First, we consider the special case where every meeting consists of two characters. For these restricted instances, every meeting can be realized from any permutation by a single block crossing. This raises the question whether there is also an *optimal* solution that fulfills this condition. The answer is negative—if we may prescribe the start permutation; see Appendix A for details.

Observation 2. *Given an instance of 2-SBCM, there is a solution with at most one block crossing before each of the meetings. In particular, there is a solution with at most n block crossings in total.*

Detecting Crossing-Free Storylines. If a storyline admits a crossing-free visualization, then the vertical permutation of the character lines remains the same over time, and all meetings involve groups that form contiguous subsets in that permutation. (The visualization can be obtained by placing characters along a vertical line in the correct permutation and for each meeting bringing its lines together for the duration of the meeting and then separating them apart again.) In other words, a single permutation supports each group of $\mathcal{H} = (C, \Gamma)$. This holds if and only if $\mathcal{H}$ is an *interval hypergraph.* This is the case if there exists a permutation $\pi = \langle v_1, \ldots, v_k \rangle$ of C such that each hyperedge $e \in \Gamma$ corresponds to a contiguous block of characters in this permutation. As an anonymous reviewer pointed out, this is equivalent to the hypergraph having path support [1]. An interval hypergraph can be visualized by placing all of its vertices on a line, and drawing each of its hyperedges as an interval that includes all vertices of e

and no vertex of $V \setminus e$. Checking whether a k-vertex hypergraph is an interval hypergraph takes $O(k^2)$ time [13]. Recall that we can build $\mathcal{H}$ in $O(nk \log n)$ time.

Theorem 3. *Given the group hypergraph $\mathcal{H}$ of an instance of SBCM with k characters, we can check in $O(k^2)$ time whether a crossing-free solution exists.*

For 2-SBCM we only need to check (in $O(k)$ time) whether $\mathcal{H}$ is a collection of vertex-disjoint paths; this is dominated by the time ($O(n)$) for building $\mathcal{H}$.

3 NP-Completeness of SBCM

In this section we prove that SBCM is NP-complete. This is known for BCM. But SBCM is not simply a generalization of BCM because in SBCM we can choose an arbitrary start permutation. Therefore, the idea of our hardness proof is to force a certain start permutation by adding some characters and meetings. We reduce from Sorting by Transpositions (SBT), which has also been used to show the hardness of BCM [5]. In SBT, the problem is to decide whether there is a sequence of transpositions (which are equivalent to block crossings) of length at most k that transforms a given permutation π to the identity. SBT was recently shown NP-hard by Bulteau et al. [2].

We show hardness for 2-SBCM, which also implies that SBCM is NP-hard. It is easy to see that SBCM is in NP: Obviously, the maximum number of block crossings needed for any number of characters and meetings is bounded by a polynomial in k and n. Therefore also the size of the solutions is bounded by a polynomial. To test the feasibility of a solution efficiently, we simply test whether the permutations between the block crossings support the meetings in the right order from left to right. We will use the following obvious fact.

Observation 4. *If permutation π needs c block crossings to be sorted, any permutation containing π as subsequence needs at least c block crossings to be sorted.*

Theorem 5. *2-SBCM is NP-complete.*

Proof. It remains to show the NP-hardness. We reduce from SBT. Given an instance of SBT, that is, a permutation π of $\{1, \dots, k\}$, we show how to use a hypothetical, efficient algorithm for 2-SBCM to determine the minimum number of transpositions (i.e., block crossings) that transforms π to the identity $\iota = \langle 1, 2, \dots, k\rangle$. Note that π can be sorted by at most k block crossings. So k is an upper bound for an optimal solution of instance π of SBT.

We extend the set of characters $\{1, 2, \dots, k\}$ to $C = \{1, \dots, k, c_1, c_2, \dots, c_{2k}\}$. Correspondingly, we extend $\pi = \langle \pi_1, \pi_2, \dots, \pi_k\rangle$ to $\pi' = \langle c_1, \dots, c_{2k}, \pi_1, \dots, \pi_k\rangle$ and ι to $\iota' = \langle c_1, c_2, \dots, c_{2k}, 1, 2, \dots, k\rangle$. Let $M_{\pi'}$ and $M_{\iota'}$ be the sequences of meetings of all neighboring pairs in π' and ι', respectively. Let M_1 and M_2 be the concatenations of $k+1$ copies of $M_{\pi'}$ and $M_{\iota'}$, respectively. By repeating we get $M_1 = M_{\pi'}^{k+1}$ and $M_2 = M_{\iota'}^{k+1}$. This yields the instance $\mathcal{S} = (C, M)$ of 2-SBCM, where M is the concatenation of M_1 and M_2; see Fig. 3.

We show that the number of block crossings needed for the 2-SBCM instance $\mathcal{S}$ equals the number of block crossings to solve instance π of SBT.

First, let B be a shortest sequence of block crossings to sort π. Then, (π', B) is a feasible solution for $\mathcal{S}$. The start permutation π' supports all meetings in M_1 without any block crossing. Using B, the lines are sorted to ι', and this permutation supports all meetings in M_2 without any further block crossings; see Fig. 3. Hence, the number of block crossings in any solution of π is an upper bound for the minimum number of block crossings needed for $\mathcal{S}$.

For the other direction, let (π^*, B^*) be an optimal solution for $\mathcal{S}$. Any solution of 2-SBCM gives rise to a symmetric solution that is obtained by reversing the order of the characters. Without loss of generality, we assume that π' (rather than the reverse permutation π'^R) occurs somewhere in M_1.

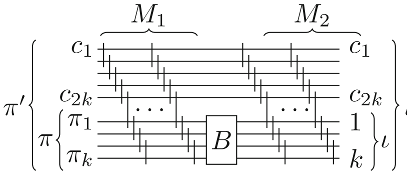


Fig. 3. Solution for the 2-SBCM instance $\mathcal{S}$ corresponding to a solution B of instance π of SBT. The box B represents the block crossings.

Next, we show that the start permutation π' occurs somewhere in M_1 and that ι' occurs somewhere in M_2. If there is a sequence $M_{\pi'}$ of meetings between which there is no block crossing, the permutation at this position can only be the start permutation π' or its reverse. For a contradiction, assume that π' does not occur during M_1 in the layout induced by (π^*, B^*). Then there is no such sequence without any block crossing in it. As this sequence is repeated $k+1$ times, the solution would need at least $k+1$ block crossings. This contradicts our upper bound, which is k. Analogously, we can show that the permutation ι' or its reverse occurs in M_2.

We now want to show that the unreversed version of ι' occurs in M_2. For a contradiction, assume the opposite. We forget about the lines $1, \ldots, k$ and only consider the sequence $\pi'' = \langle c_1, \ldots, c_{2k} \rangle$ in π' which is reversed to $\iota'' = \langle c_{2k}, \ldots, c_1 \rangle$ in ι'^R. Eriksson et al. [4] showed that we need $\lceil (l+1)/2 \rceil$ block crossings to reverse a permutation of l elements. This implies that we need $k+1$ block crossings to transform π'' to ι''. As π' and ι'^R contain these sequences as subsequences, Observation 4 implies that the transformation from π' to ι'^R also needs at least $k+1$ block crossings. As the optimal solution uses at most k block crossings, we know that we cannot reach ι'^R and thus the sequence of permutations contains π' and ι'.

The sequence of block crossings that transforms π' to ι' yields a sequence B of block crossings of the same length that transforms π to ι. This shows that the length of a solution for $\mathcal{S}$ is an upper bound for the length of an optimal solution of the corresponding SBT instance π. Thus, the two are equal. $\square$

Hardness Without Repetitions. With arbitrarily large meetings, SBCM is hard even without repeating meetings. We can emulate a repeated sequence of 2-character meetings by gradually increasing group sizes; see Appendix B.

4 Exact Algorithms

We present two exact algorithms. Conceptually, both build up a sequence of block crossings while keeping track of how many meetings have already been accomplished. The first uses polynomial space; the second improves the runtime at the cost of exponential space.

We start with a data structure that keeps track of permutations, block crossings and meetings. It is initialized with a given permutation and has two operations. The Check operation returns whether a given meeting fits the current permutation. The BlockMove operation performs a given block crossing on the permutation and then returns whether the most-recently Checked meeting now fits. See Appendix C for a detailed description.

Lemma 6. *A sequence of arbitrarily interleaved* BlockMove *and* Check *operations can be performed in $O(\beta + \mu)$ time, where β is the number of block crossings and μ is sum of cardinalities of the meetings given to* Check*. Space usage is $O(k)$.*

A block crossing can be represented by indices (a, b, c) with $1 \le a \le b < c \le k$; hence, there are $\frac{k^3-k}{6}$ distinct block crossings on a permutation of length k.

Now we provide an output-sensitive algorithm for SBCM whose runtime depends on the number of block crossings required by the optimum.

Theorem 7. *An instance of SBCM can be solved in $O(k! \cdot (\frac{k^3-k}{6})^\beta \cdot (\beta + \mu))$ time and $O(\beta k)$ working space if a solution with β block crossings exists, where $\mu = \sum_{i \in M} |m_i|$.*

Proof. Consider a branching algorithm that starts from a permutation of the characters and keeps trying all possible block crossings. This has branching factor $\frac{k^3-k}{6}$ and we can enumerate the children of a node in constant time each by enumerating triples (a, b, c). While applying block crossings, the algorithm keeps track of how many meetings fit this sequence of permutations using the data structure from Lemma 6. We use depth-first iterative-deepening search [7] from all possible start permutations until we find a sequence of permutations that fulfills all meetings. Correctness follows from the iterative deepening: we want an (unweighted) shortest sequence of block crossings. The runtime and space bounds follow from the standard analysis of iterative-deepening search, observing that a node uses $O(k)$ space and it takes $O(\beta + \mu)$ time in total to evaluate a path from root to leaf. $\square$

We have that μ is $O(kn)$ since there are n meetings and each consists of at most k characters. At the cost of exponential space, we can improve the runtime and get rid of the dependence on β, showing the problem to be fixed parameter linear for k. We note that the following algorithm can easily be adapted to handle pairwise crossings rather than block crossings; in this case the runtime improves upon the original result of Kostisyna et al. [8] by a factor of $k!$.

Theorem 8. *An instance of SBCM can be solved in $O(k! \cdot k^3 \cdot n)$ time and $O(k! \cdot k \cdot n)$ space.*

Proof. Let $f(\pi, \ell)$ be the optimal number of block crossings in a solution to the given instance when restricted to the first ℓ meetings and to have π as its final permutation. Note that by definition the solution for the actual instance is given by $\min_{\pi^*} f(\pi^*, n)$, where the minimum ranges over all possible permutations. As a base case, $f(\pi, 0) = 0$ for all π, since the empty set of meetings is supported by any permutation. Let π and π' be permutations that are one block crossing apart and let $0 \leq \ell \leq \ell'$. If the meetings $\{m_{\ell+1}, \ldots, m_{\ell'}\}$ fit π', then $f(\pi', \ell') \leq f(\pi, \ell) + 1$: if we can support the first ℓ meetings and end on π, then with one additional block crossing we can support the first ℓ' meetings and end with π'.

We now model this as a graph. Let G be an unweighted directed graph on nodes (π, ℓ) and call a node *start node* if $\ell = 0$. There is an arc from (π, ℓ) to (π', ℓ') if and only if π and π' are one block crossing apart, $\ell \leq \ell'$, and the meetings $\{m_{\ell+1}, \ldots, m_{\ell'}\}$ fit π'. Note that we allow $\ell = \ell'$ since we may need to allow block crossings that do not immediately achieve an additional meeting (cf. Proposition 18), so G is not acyclic. In the constructed graph, $f(\pi, \ell)$ equals the graph distance from the node (π, ℓ) to the closest start node. Call a path to a start node that realizes this distance *optimal.*

In G, consider any path $[(\pi_1, \ell_1), (\pi_2, \ell_2), (\pi_3, \ell_3)]$ with $\ell_3 > \ell_2$. If meeting $\ell_2 + 1$ fits π_2, then $[(\pi_1, \ell_1), (\pi_2, \ell_2 + 1), (\pi_3, \ell_3)]$ is also a path. Repeating this transformation shows that for all π, the node (π, n) has an optimal path in which every arc maximally increases ℓ. Let G' be the graph where we drop all arcs from G that do not maximally increase ℓ. Note that G' still contains a path that corresponds to the global optimum.

The graph G' has $O(k! \cdot n)$ nodes and each node has outdegree $O(k^3)$. Then a breadth-first search from all start nodes to any node (π^*, n) achieves the claimed time and space bounds, assuming we can enumerate the outgoing arcs of a node in constant time each.

For a given node (π, ℓ) we can enumerate all possible block crossings in constant time each, as before. In G', we also need to know the maximum ℓ' such that all meetings $\ell + 1$ up to ℓ' fit π'. Note that ℓ' only depends on ℓ and π'. We precompute a table $M(\pi, \ell)$ that gives this value. Computing $M(\pi, \ell)$ for given π and all ℓ takes a total of $O(kn)$ time: first compute for every m_i whether it fits π, then compute the implied 'forward pointers' using a linear scan. So using $O(k! \cdot k \cdot n)$ preprocessing time and $O(k! \cdot n)$ space, we have an efficient implementation of the breadth-first search. The theorem follows. □

5 SBCM with Meetings of Two Characters

A Greedy Algorithm. To quickly draw good storyline visualizations for 2-SBCM, we develop an $O(kn)$-time greedy algorithm. Given an instance $\mathcal{S} = (C, M)$, we reserve a list $B = []$ that the algorithm will use to store the block crossings. The algorithm starts with an arbitrary permutation π^0 of C. In every step the algorithm removes all meetings from the beginning of M that fit the current permutation π^i of the algorithm. Subsequently, the algorithm picks a block crossing b such that the resulting permutation $\pi^{i+1} = b(\pi^i)$ supports the maximum number of meetings from the beginning of M. Then b is appended to the list B. This process repeats until M is empty. The algorithm returns (π^0, B).

Note that there are at most $O(k^3)$ possible block crossings. Thus to find the appropriate block crossings, the algorithm could simply check all of them. Many of those, however, will result in permutations that do not even support the next meeting, which would be a bad choice. Hence, our algorithm considers only *relevant* block crossings, i.e., block crossings yielding a permutation that supports the next meeting. Let $\{c, c'\}$ be the next meeting in M. If x and y are the positions of c and c' in the current permutation, i.e., $\pi^i_x = c$ and $\pi^i_y = c'$ (without loss of generality, assume $x < y$), the relevant block crossings are:

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

So the number of relevant block crossings in each step is $k + 1$. Let n_i be the maximum number of meetings at the beginning of M we can achieve by one of these block crossings. We use the data structure in Lemma 6 and check for each relevant block crossing how many meetings can be done with this permutation. Hence, we can identify a block crossing achieving the maximum number in $O(kn_i)$ time since we have to check $k + 1$ paths containing up to n_i meetings each. Clearly, the numbers of meetings n_i in each iteration of the algorithm sum up to n and therefore the algorithm runs in $O(kn)$ total time.

The way we described the greedy algorithm, it starts with an arbitrary permutation. Instead, we could start with a permutation that supports the maximum number of meetings before the first block crossing needs to be done. In other words, we want to find a maximal prefix M' of M such that (C, M') can be represented without any block crossings. We can find M' in $O(kn)$ time: we start with an empty graph and add the meetings successively. In each step we check whether the graph is still a collection of paths, which can be done in $O(k)$ time. It is easy to construct a permutation that supports all meetings in M'. While this is a sensible heuristic, we do not prove that this reduces the total number of block crossings. Indeed, we experimentally observe that while the heuristic is generally good, this is not always the case; see Fig. 4 for an example that uses the heuristic start permutation.

Note that the greedy algorithm yields optimal solutions for special cases of 2-SBCM. The proof for the following theorem can be found in Appendix D.

Theorem 9. *For $k = 3$, the greedy algorithm produces optimal solutions.*

Experimental Evaluation. In this section, we report on some preliminary experimental results. We only consider 2-SBCM. We generated random instances as follows. Given n and k, we generate n pairs of characters as meetings, uniformly at random using rejection sampling to ensure that consecutive meetings are different. (Repeated meetings are not sensible.)

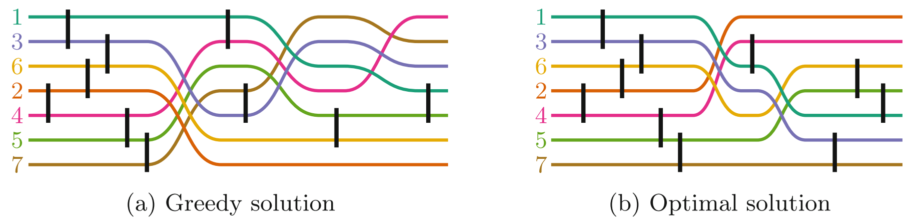


Fig. 4. The greedy algorithm is not optimal.

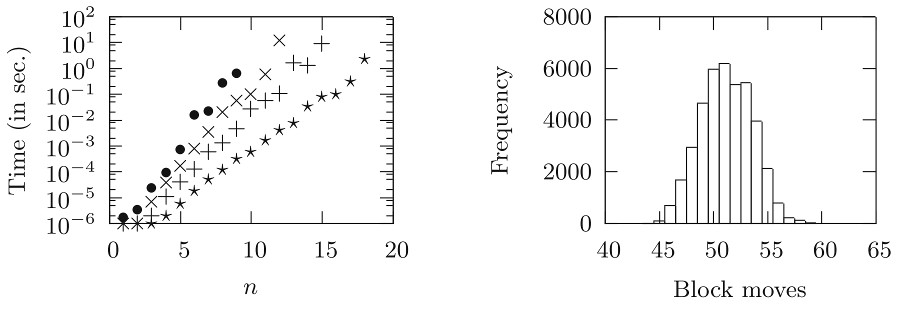
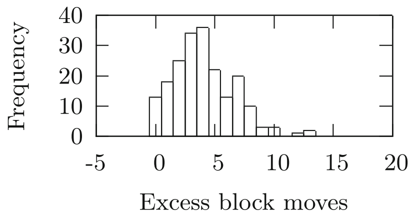


Fig. 5. Left: Runtime of the exact algorithm of Theorem 7 on random instances with $k = 4(\star), 5(+), 6(\times), 7(\bullet)$. Each data point is the average of 50 random instances. Right: Histogram of the number of block crossings used by the greedy algorithm for all $k!$ different start permutations, on a single random instance with $n = 100$ and $k = 8$.

First, we consider the exact algorithm of Theorem 7. As expected, its runtime depends heavily on k (Fig. 5, left). Perhaps unexpectedly, we observe exponential runtime in n. This is actually a property of our random instances, in which β tends to increase linearly with n. Note that this does not invalidate the algorithm since we may be interested in instances for which β is indeed small.

Since the exact algorithm is feasible only for rather small instances, we now shift our focus to the greedy algorithm. Recall that it starts with an arbitrary permutation and proceeds greedily. The histogram in Fig. 5 (right) shows the number of block crossings used by the greedy algorithm depending on the start permutation, for a single random instance: this bell curve is typical. We see that there are "rare" start permutations that do strictly better than almost all others. Indeed, for the reported instance, a random start permutation does 7.2 block crossings worse in expectation than the best possible start permutation.

We call the best possible result of the greedy algorithm over all start permutations BestGreedy, which we calculate by brute force. Let RandomGreedy start with a permutation chosen uniformly at random, and let HeuristicGreedy start with the heuristic start permutation that we have described above. The histogram in Fig. 6 (left) shows how many more block crossings HeuristicGreedy uses than BestGreedy on random instances. This distribution is heaviest near zero, but there are instances where performance is poor.



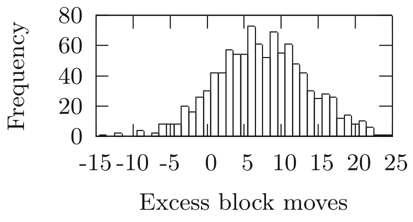


Fig. 6. Left: histogram of HeuristicGreedy minus BestGreedy, 200 instances with with $k = 7$ and $n = 100$. Right: histogram of RandomGreedy minus Heuristic-Greedy, 1000 instances with $k = 30$ and $n = 200$.

Note that we do not know how to compute BestGreedy efficiently. Compared to RandomGreedy, we see that HeuristicGreedy fares well (Fig. 6, right).

Lastly, we compare the greedy algorithm to the optimum, which we can only do for small k and n. On 1000 random instances with $k = 5$ and $n = 12$, HeuristicGreedy was optimal 56% of the time. It was sometimes off by one (38%), two (5%), or three (1%), but never worse. This is a promising behavior, but clearly cannot be extrapolated verbatim to larger instances.

Based on these experiments, we recommend HeuristicGreedy as an efficient, reasonable heuristic.

6 Approximation Algorithm

We now develop a constant-factor approximation algorithm for d-SBCM where d is a constant. We initially assume that each group meeting occurs exactly once, but later show how to extend our results to the setting where the same group can meet a bounded number of times.

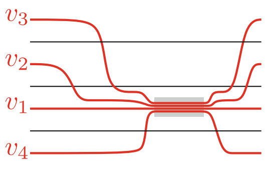


Fig. 7. Meeting $\{v_1, v_2, v_3, v_4\}$

Overview. Our approximation algorithm has the following three main steps.

1. Reduce the input group hypergraph $\mathcal{H} = (C, \Gamma)$ to an *interval hypergraph* $\mathcal{H}_f = (C,\ \Gamma \setminus \Gamma_p)$ by deleting a subset $\Gamma_p \subseteq \Gamma$ of the edges of $\mathcal{H}$.
2. Choose a permutation π^0 of the characters that supports all groups of this interval hypergraph $\mathcal{H}_f$. Thus, π^0 is the order of characters at the beginning of the timeline.
3. Incrementally create support for each deleted meeting of Γ_p in order of increasing time, as follows. Suppose that $g \in \Gamma_p$ is the group meeting to support. Keep one of the character lines involved in this meeting fixed and bring, for the duration of the meeting, the remaining (at most $d - 1$) lines close to it. Then retract those lines to their original position in π^0; see Fig. 7.

Step 2 is straightforward: Sect. 2 shows how to find a permutation supporting all the groups for an interval hypergraph. In Step 3, we introduce at most $2(d-1)$ block crossings for each meeting $g \in \Gamma_p$ not initially supported. The main technical parts of the algorithm are Step 1 and an analysis to charge at most a constant number of block crossings in Step 3 to a block crossing in the optimal visualization. Step 1 requires solving a hypergraph problem; this is technically the most challenging part, and consumes the entire Sect. 7.

Bounds and Analysis. We call Γ_p *paid* edges, and the remainder $\Gamma_f = \Gamma \setminus \Gamma_p$ *free* edges. Intuitively, free edges can be realized without block crossings because $\mathcal{H}_f$ is an interval hypergraph, while the edges of Γ_p must be charged to block crossings of the optimal drawing. We initialize the drawing by placing the characters in the vertical order π^0, which supports all the groups in Γ_f. Now we consider the paid edges in left-to-right order. Suppose that the next meeting involves a group $g' \in \Gamma_p$. We have $|g'| \leq d$. We arbitrarily fix one of its characters, leaving its line intact, and bring the remaining $(d-1)$ lines in its vicinity to realize the meeting. This creates at most $(d-1)$ block crossings, one per line. When the meeting is over, we again use up to $(d-1)$ block crossings to revert the lines back to their original position prescribed by π^0; see Fig. 7.

We do this for each paid hyperedge, giving rise to at most $2(d-1)|\Gamma_p|$ block crossings. We now prove that this bound is within a constant factor of optimal. We first establish a lower bound on the optimal number of block crossings *assuming* that π^0 is the optimal start permutation.

Lemma 10. *Let π be a permutation of the characters, let Γ_f be the groups supported by π, and let $\Gamma_p = \Gamma \setminus \Gamma_f$. Any storyline visualization that uses π as the start permutation has at least $4|\Gamma_p|/(3d^2)$ block crossings.*

Proof. Let $g \in \Gamma_p$. Since g is not supported by π, the optimal drawing does not contain the characters of g as a contiguous block initially. However, in order to support this meeting, these characters must eventually become contiguous before the meeting starts. The order changes only through (block) crossings; we bound the number of groups that can become supported after each block crossing.

After a block crossing, at most three pairs of lines that were not neighbors before can become neighbors in the permutation: after the blocks $C_1, C_2 \subseteq C$ cross, there is one position in the permutation where a line of C_1 is next to a line of C_2, and two positions with a line of C_1 (C_2, respectively) and a line of $C \setminus (C_1 \cup C_2)$. Any group that was not supported, but is supported after the block crossing, must contain one of these pairs. We can describe each such group in the new permutation by specifying the new pair and the numbers d_1 and d_2 of characters of the group above and below the new pair in the permutation. Since the group size is at most d, we have $d_1 + d_2 \leq d$. The product $d_1(d - d_1)$ achieves its maximum value for $d_1 = d_2 = d/2$, and so there are at most $d^2/4$ possible groups for each new pair. Thus, the total number of newly supported groups after a block crossing is at most $3d^2/4$, which shows that the optimal number of block crossings is at least $4|\Gamma_p|/(3d^2)$, completing the proof. □

We now bound the loss of optimality caused by not knowing the initial permutation used by the optimal solution. The key idea here is to use a constant-factor approximation for the problem of deleting the minimum number of hyperedges from $\mathcal{H}$ so that it becomes an interval hypergraph (Interval Hypergraph Edge Deletion). We prove the following theorem in Sect. 7.

Theorem 11. *We can find a $(d+1)$-approximation for* Interval Hypergraph Edge Deletion *on group hypergraphs with n meetings of rank d in $O(n^2)$ time.*

Let Γ_{OPT} be the set of paid edges in the optimal solution, and Γ_p the set of paid edges in our algorithm. By Theorem 11, we have $|\Gamma_p| \le (d+1)|\Gamma_{\mathrm{OPT}}|$. Let ALG and OPT be the numbers of block crossings for our algorithm and the optimal solution, respectively. By Lemma 10, we have $\mathrm{OPT} \ge 4|\Gamma_{\mathrm{OPT}}|/(3d^2)$, which gives $|\Gamma_{\mathrm{OPT}}| \le 3d^2/4 \cdot \mathrm{OPT}$. On the other hand, we have $\mathrm{ALG} \le 2(d-1)|\Gamma_p| \le 2(d-1)(d+1)|\Gamma_{\mathrm{OPT}}|$. Combining the two inequalities, we get $\mathrm{ALG} \le 3(d^2-1)d^2/2 \cdot \mathrm{OPT}$, which establishes our main result.

Theorem 12. *d-SBCM admits a $(3(d^2-1)d^2/2)$-approximation algorithm.*

Remark. We assumed that each group meets only once, but we can extend the result if each group can meet c times, for constant c. Our algorithm then yields a $(c \cdot 3(d^2-1)d^2/2)$-factor approximation; each repetition of a meeting may trigger a constant number of block crossings not present in the optimal solution.

Runtime Analysis. We have to consider the permutation (of length k) of characters before and after each of the n meetings, as well as after each of the $O(n)$ block crossings. This results in $O(kn)$ time for the last part of the algorithm, but this is dominated by the time ($O(n^2)$) needed for finding Γ_p and for determining the start permutation.

We can improve the running time to $O(kn)$ by a slight modification: using the approximation algorithm for Interval Hypergraph Edge Deletion is only necessary for sparse instances. If $\mathcal{H}$ has sufficiently many edges, any start permutation will yield a good approximation. Since no meeting involves more than d characters, no start permutation can support more than dk meetings. If $n \ge 2dk$, then even the optimal solution must therefore remove at least half of the edges. Hence, taking an arbitrary start permutation yields an approximation factor of at most $2 < d+1$.

We now change the algorithm to use an arbitrary start permutation if $n \ge 2dk$ and only use the approximation for Interval Hypergraph Edge Deletion otherwise, i.e., especially only if there are $O(k)$ edges. Hence, for sparse instances we have $O(n^2) = O(k^2)$, and for dense instances, the $O(n^2)$ runtime is not necessary. We get the following improved result. (The runtime is worst-case optimal since the output complexity is of the same order.)

Theorem 13. *d-SBCM admits an $O(kn)$-time $(3(d^2-1)d^2/2)$-approximation algorithm.*

Using some special properties of the 2-character case, we can improve the approximation factor for 2-SBCM from 18 to 12; see Appendix E.

7 Interval Hypergraph Edge Deletion

We now describe the main missing piece from our approximation algorithm: how to approximate the minimum number of edges whose deletion reduces a hypergraph to an interval hypergraph, i.e., how to solve the following problem.

*Problem 14 (*Interval Hypergraph Edge Deletion*).* Given a hypergraph $\mathcal{H} = (V, E)$ find a smallest set $E_p \subseteq E$ such that $\mathcal{H}_f = (V, E \setminus E_p)$ is an interval hypergraph.

Note that a graph contains a Hamiltonian path if and only if one can remove all but $n-1$ edges so that only vertex-disjoint paths (here, a single path) remain; hence, our problem is hard even for graphs.

Theorem 15. Interval Hypergraph Edge Deletion *is NP-hard.*

We now present a $(d+1)$-approximation algorithm for rank-d hypergraphs, in which each hyperedge has at most d vertices. In this section we give all main ideas. Detailed proofs can be found in Appendix F; they are mostly not too hard to obtain, but require the distinction of many cases.

For our algorithm, we use the following characterization: A hypergraph is an interval hypergraph if and only if it contains none of the hypergraphs shown in Fig. 8 as a subhypergraph [9,13]. Due to the bounded rank, the families of F_k and M_k are finite with F_{d-2} and M_{d-1} as largest members. Cycles are the only arbitrarily large forbidden subhypergraphs in our setting. Let $\mathcal{F} = \{O_1, O_2, F_1, \ldots, F_{d-2}, M_1, \ldots, M_{d-1}, C_3, \ldots, C_{d+1}\}$. A hypergraph is $\mathcal{F}$-free if it does not contain any hypergraph of $\mathcal{F}$ as a subhypergraph. Note that a cycle in a hypergraph consists of hyperedges $e_1, \ldots, e_k$ so that there are vertices $v_1, \ldots, v_k$ with $v_i \in e_{i-1} \cap e_i$ for $2 \leq i \leq k$ (and $v_1 \in e_1 \cap e_k$) and no edge e_i contains a vertex of $v_1, \ldots, v_k$ except for v_i and v_{i+1}.

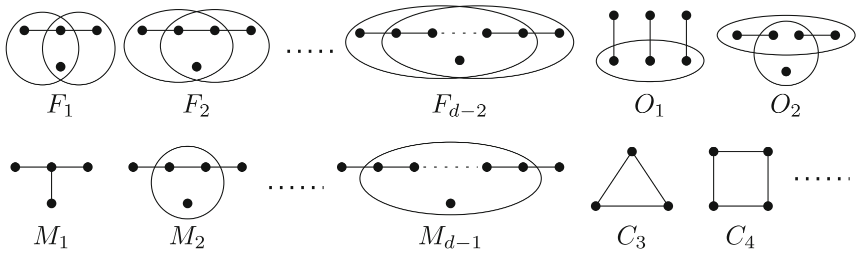


Fig. 8. Forbidden subhypergraphs for interval hypergraphs (edges represent pairwise hyperedges, circles/ellipses show hyperedges of higher cardinality).

Our algorithm consists of two steps. First, we search for subhypergraphs contained in $\mathcal{F}$, and remove all edges involved in these hypergraphs. In the second step, we break remaining (longer) cycles by removing some more hyperedges after carefully analyzing the structure of connected components. Subhypergraphs in $\mathcal{F}$ consist of at most $d+1$ hyperedges. A given optimal solution must remove at least one of the hyperedges; removing all of them instead yields a factor of at most $d+1$. The second step will not negatively affect this approximation factor.

Intuitively, allowing long cycles, but forbidding subhypergraphs of $\mathcal{F}$, results in a generalization of interval hypergraphs where the vertices may be placed on a cycle instead of a vertical line. This is not exactly true, but we will see that the connected components after the first step have a structure similar to this, which will help us find a set of edges whose removal destroys all remaining long cycles.

Lemma 22 (Appendix F) shows that any vertex is contained in at most three hyperedges of a cycle, where the case of three hyperedges with a common vertex occurs only if a hyperedge is contained in the union of its two neighbors in the cycle. Assume that e_1, e_2, and e_3 are consecutive edges of a cycle C. If all three edges are present in an interval representation, we know that we will first encounter vertices that are only contained in e_1, then vertices that are in $(e_1 \cap e_2) \setminus e_3$, then vertices in $e_1 \cap e_2 \cap e_3$, followed by vertices of $(e_2 \cap e_3) \setminus e_1$, and vertices of $e_3 \setminus (e_1 \cup e_2)$. Some of the sets (except for pairwise intersections) may be empty. We do not know the order of vertices within one set, but we know the relative order of any pair of vertices of different sets. By generalizing this to the whole cycle, we get a cyclic order—describing the local order in a possible interval representation—of sets defined by containment in 1, 2, or 3 hyperedges. We call these sets *cycle-sets* and their cyclic order the *cycle-order* of C.

We can analyze how an edge $e \notin C$ relates to the order of cycle-sets; e can contain a cycle-set completely, can be disjoint from it, or can contain only part of its vertices. We call a consecutive sequence of cycle-sets contained in edge e—potentially starting and ending with cycle-sets partially contained in e—an *interval* of e on C. The following lemma shows that every edge forms only a single interval on a given cycle.

Lemma 16. *If a hyperedge $e \in E$ intersects two cycle-sets of a cycle C, then e fully contains all cycle-sets lying in between in one of the two directions along C.*

We now know that by opening the cycle at a single position within a cycle-set not contained in e, $C + e$ forms an interval hypergraph. Edge e adds further information: If only part of the vertices of a cycle-set are contained in e and also vertices of the next cycle-set in one direction, we know that the vertices of e in the first cycle-set should be next to the second cycle-set. We use this to refine the cycle-sets to a cyclic order of *cells*, the *cell order* (a cell is a set of vertices that should be contiguous in the cyclic order). Initially, the cells are the cycle-sets. In each step we refine the cell-order by inserting an edge containing vertices of more than one cell, possibly splitting two cells into two subcells each. The following lemma shows that during this process of refinements, as an invariant each remaining edge forms a single interval on the cell order.

Lemma 17. *If a hyperedge $e \in E$ intersects two cells, then e fully contains all cells lying in between in one of the two directions along the cyclic order.*

After refining cells as long as possible, each edge of the connected component that we did not insert lies completely within a single cell. Several edges can lie within the same cell, forming a hypergraph that imposes restrictions on the order of vertices within the cell. However, the cell contains fewer than d vertices. Hence, this small hypergraph cannot contain any cycles, since we removed all short cycles, and must be an interval hypergraph.

With this cell-structure, it is not too hard to show that the following strategy to make the connected component an interval hypergraph is optimal (see Lemmas 24, 25 and 26 in Appendix F): For each pair of adjacent cells we determine the number of edges containing both cells, select the pair minimizing that number, and remove all edges containing both. The cell order then yields an order of the connected component's vertices that supports all remaining edges. Since this last step of the algorithm is done optimally, we do not further change the approximation ratio, which, overall, is $d + 1$, because we never remove more than $d + 1$ edges for at least one edge that the optimal solution removes.

Runtime. Our algorithm can be implemented to run in $O(m^2)$ time for m hyperedges. We give the main ideas here and present details in Appendix G. When searching for forbidden subhypergraphs, we first remove all cycles of length $k \leq d$ using a modified breadth-first search in $O(m^2)$ time. The remaining types of forbidden subhypergraphs each contain an edge that contains all but one (O_2 and F_k), two (M_k), or three (O_1) vertices of the subhypergraph. We always start searching from such an edge and use that all short cycles have already been removed. In the second phase, we determine the connected components and initialize the cell order for each of them, in $O(n + m)$ time. Stepwise refinement requires $O(m^2)$ time. Counting hyperedges between adjacent cells, determining optimal splitting points, and finding the final order can all be done in linear time.

Theorem 11 *We can find a $(d+1)$-approximation for* INTERVAL HYPERGRAPH EDGE DELETION *on hypergraphs with m hyperedges of rank d in $O(m^2)$ time.*

References

1. Buchin, K., van Kreveld, M.J., Meijer, H., Speckmann, B., Verbeek, K.: On planar supports for hypergraphs. J. Graph Algorithms Appl. **15**(4), 533–549 (2011)
2. Bulteau, L., Fertin, G., Rusu, I.: Sorting by transpositions is difficult. SIAM J. Discrete Math. **26**(3), 1148–1180 (2012)
3. van Dijk, T.C., Fink, M., Fischer, N., Lipp, F., Markfelder, P., Ravsky, A., Suri, S., Wolff, A.: Block crossings in storyline visualizations (2016). http://arxiv.org/abs/1609.00321
4. Eriksson, H., Eriksson, K., Karlander, J., Svensson, L., Wästlund, J.: Sorting a bridge hand. Discrete Math. **241**(1), 289–300 (2001)

5. Fink, M., Pupyrev, S., Wolff, A.: Ordering metro lines by block crossings. J. Graph Algorithms Appl. **19**(1), 111–153 (2015)
6. Kim, N.W., Card, S.K., Heer, J.: Tracing genealogical data with timenets. In: Proceedings of the International Conference Advanced Visual Interfaces (AVI 2010), pp. 241–248 (2010)
7. Korf, R.E.: Depth-first iterative-deepening: an optimal admissible tree search. Artif. Intell. **27**(1), 97–109 (1985)
8. Kostitsyna, I., Nöllenburg, M., Polishchuk, V., Schulz, A., Strash, D.: On minimizing crossings in storyline visualizations. In: Di Giacomo, E., Lubiw, A. (eds.) GD 2015. LNCS, vol. 9411, pp. 192–198. Springer, Heidelberg (2015). doi:10.1007/978-3-319-27261-0_16
9. Moore, J.I.: Interval hypergraphs and D-interval hypergraphs. Discrete Math. **17**(2), 173–179 (1977)
10. Muelder, C., Crnovrsanin, T., Sallaberry, A., Ma, K.: Egocentric storylines for visual analysis of large dynamic graphs. In: Proceedings of the IEEE International Conference Big Data, pp. 56–62 (2013)
11. Munroe, R.: Movie narrative charts. https://xkcd.com/657/
12. Tanahashi, Y., Ma, K.: Design considerations for optimizing storyline visualizations. IEEE Trans. Vis. Comput. Graph. **18**(12), 2679–2688 (2012)
13. Trotter, W.T., Moore, J.I.: Characterization problems for graphs, partially ordered sets, lattices, and families of sets. Discrete Math. **16**(4), 361–381 (1976)