# Storyline Visualizations

Michael Höller
Matr.: 01225059
Curriculum: 066 937
e1225059@student.tuwien.ac.at

December 22, 2017

**Abstract**

This is a seminar paper about storyline visualizations, in particular minimizing block crossings in such visualizations. The main focus is on the work of Dijk et al. in "Block Crossings in Storyline Visualizations" [12]. The aim of this paper is to provide a brief overview of the topic, give a glimpse into the paper and put it into the context of other work done regarding this topic. The main part deals with the introduced approximation and runtime prove of the approximation algorithm. At last, a discussion about the topic, this paper in particular and open questions follows at the end.

## 1 Introduction

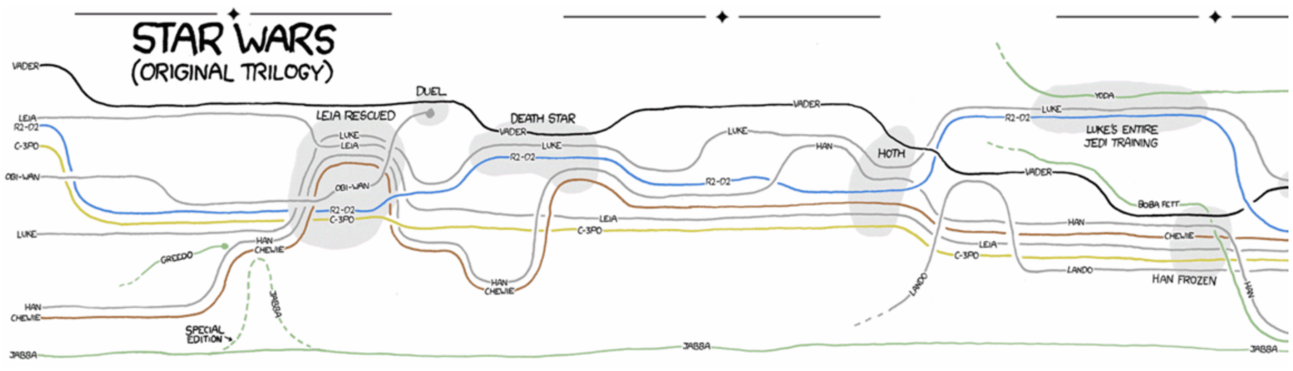

Figure 1: Cutout of Munroe's Movie narrative charts [8]

A Storyline visualization is a way to convey the information about interactions of people over time. The vocabulary is based on movies, where each scene defines the characters (actors) who meet (interact) on a certain point in time. However, storyline visualizations are not restricted to this domain, since it is also applicable to visualize any temporal relationship that any concepts have between each other. The characters are displayed as lines, drawn in an $x$-monotone manner from left to right. They have a certain vertical ordering (permutation) at each point in time. A meeting consists of a set of characters and is indicated by proximity of the corresponding lines. A meeting is 'supported', if the characters of the meeting are next to each other within the vertical ordering. It is often beneficial to provide additional context to the meetings, e.g. the location where the meetings take place.

Such visualizations can give a global overview and reveal global trends, but also show local interactions [11]. This can be beneficial for many applications. For instance, eye-tracking in TV commercials in advertising on the one hand wants to know, how many of their test persons (characters) look at the product (location) they advertise. On the other hand, they might also be interested in 'profiles' among individuals or groups of persons in order to reason about it and learn how to improve future advertisings.

The goal is to generate a good visualization for a given instance. Only the readers can say whether it is good or not. However, there are several design aspects, which evidently have a positive impact on readability and understandability.

**General problem formulation (by Kostitsyna et al. [7]):**
Given a storyline instance $S = (C, T, E)$, where $C = 1, ..., n$ is a set of characters, that meet during closed time intervals $T \subset \{[s,t] | s, t \in N, s \leq t\}$. A meeting is also called event, while the set of events is denoted as $E \subset 2^C \times T$. Each event $E_i = (C_i, [s_i, t_i]) \in E)$ $(1 \leq i \leq m)$ has a subset of characters $C_i \subseteq C$ that meet for the entire time interval $[s_i, t_i] \in T$.
The goal is to produce a 2D drawing of S (the storyline visualization) where the x-axis represents time, and characters are drawn as x-monotone curves placed in some vertical order for each point in time. During each event $E_i = (C_i, [s_i, t_i])$, curves representing characters in $C_i$ should be grouped within some small vertical distance $\sigma_{group}$ of each other, and otherwise the characters should be separated by some larger vertical distance $\sigma_{separate} > \sigma_{group}$.
This sets the foundations how a drawing has to look like at least. In order to make good drawings, there are many different problem formulations, each considering one or more aspects to optimize. One such aspect is minimizing crossings of the lines. Moreover, there are multiple slightly different variants of the crossing minimization problem. Each variant is basically tailored to the needs of the introduced algorithm. The same applies to a related problem, namely SBCM. The following is the exact problem definition this paper is using:

**Storyline Block Crossing Minimization problem (SBCM):**
Given a storyline instance $(C, M)$ with $M = [m_1, ..., m_n]$, find a solution $(\pi_0, B)$ consisting of a start permutation $\pi_0$ of $C$ and a sequence $B = (B_1, ..., B_n)$ of (possibly empty) sequences of block crossings such that the total number of block crossings is minimized and $\pi_i = B_i(\pi_{i-1})$ supports $m_i$ for $1 \leq i \leq n$.

Since there is no notion of time intervals, this problem definition assumes there is only one meeting at each point in time, thus no concurrent meetings.
$d$-SBCM is a weaker version of the problem, where $d$ is the maximum meeting size.
A block crossing is an alternative way of counting crossing lines. In a block crossing, instead of counting pairwise line crossings, the crossing of two arbitrary large sets of parallel lines with no other line between them is counted as one crossing.

## 2 State-of-the-art

**The roots** The work of Munroe [8], visualizing the story line of a range of movies such as Star Wars, Lord of the Rings and Jurassic Park inspired many researchers to visualize data for different applications in a similar way.

Back then, Tanahashi and Ma were the first ones who set up design principles for such visualizations. They found that storyline visualizations take an important role compared to other techniques such as animations or 3D visualizations, since the traditional techniques have limitations e.g. in conveying subtle trends, in aesthetics or are dependent on the viewing perspective. The principles are based on accepted guidelines and professional illustrators, a fundamental one being that a line must not deviate unless it converges or diverges with another line. Thus, a change of a line would suggest that the social interaction of this character changes. They also defined metrics to measure the quality of visualizations which are line wiggles (drawing lines as straight as possible), white-space gaps (space efficiency) and line crossovers. The aim of this work also was to introduce an automation technique which produces drawings that match the aesthetics and legibility of Munroe's drawings. For this, they provided an evolutionary algorithm considering all these design principles. First, an algorithm computes the layout, and then methods for adjusting the geometry of individual lines are applied for improved aesthetics. Additionally providing context to the visualizations e.g. places where the meetings take place, made their algorithm's solutions be close to the hand-drawings of Munroe.

Tanahashi was also the one who provided a movie dataset in [10] which is the foundation for several papers regarding this topic. The dataset includes the three popular movies Inception, The Matrix and Star Wars. Gronemann later revealed in [6] that varying versions of [10] are used in research and what the differences are. This is important to consider when benchmarking with other algorithms.

Kostitsyna et al. started to focus on minimizing line crossings [7], which is considered the hardest aspect of generating good storyline visualizations. They provided a general problem formulation for the storyline visualization problem. Moreover, they showed the fixed-parameter tractability of the problem when parameterized on the number of characters $k$ which runs in $O(k!^2 k \log k + k!^2 m)$ runtime. A special case of the problem was also considered, that is when the meetings consist of exactly two characters and can be modeled as a tree. Then, they found that one can always find a storyline with $O(n \log n)$ crossings and there are some storylines that require $\Omega(n \log n)$ crossings.

Gronemann et al. provided an exact Integer Linear Programming algorithm for the problem, which showed reasonable runtime for medium sized instances (instances with up to 100 characters having up to 100 meetings) [6]. They modeled the problem as a multi-layer crossing minimization problem with tree constraints. However, their approach is hard to be extended to optimizing other design aspects besides line crossings, which was done by Tanahashi and Ma in [11].

**Block crossings** At this time, Fink and Pupyrev already had done research on metro line maps visualizations introducing their idea of minimizing block crossings of such metro lines instead of pairwise crossings [5]. SBCM seems to be a related problem to metro-line crossings minimization, namely a special case of it where lines only go from left to right. However, metro lines have a fix geographical position, which is not the case for SBCM. Fink pursued the idea further and published another paper [4] with a different team of researchers, about block crossings (also called 'bundled crossings') in general graphs.

**This paper** Eventually, Fink et al also applied the idea to the field of storyline visualizations [12] which is the paper this work is about. The paper provides a wide range of results to the Storyline Block Crossing Minimization (SBCM):

- NP-hardness of SBCM by reducing the problem from Sorting By Transpositions problem (SBT)
- Two exact algorithms
    1. using iterative deepening depth-first search, needing polynomial space
    2. fixed-parameter tractable algorithm using breadth-first search
- Greedy heuristic for 2-SBCM
- $\alpha 3d^2(d^2-1)$-Approximation for $d$-SBCM which runs in $O(kn)$
- Check whether crossing-free solution exists in $O(k^2)$

This seminar paper focuses on the approxmation algorithm of [12].

**SBCM as SAT problem formulation** Shortly after this paper, they published a follow-up paper about formulating the Block Crossings Minimization Problem as a SAT problem [13]. The concrete problem definition differs from the one used in [12], as concurrent meetings (they may overlap), and birth and deaths (lifetime $E : C \to P(R)$ where $R$ is the set of intervals in the real numbers) are supported. Obviously, this is a harder problem than used in their first paper [12], but the algorithm still outperforms the two exact algorithms presented there. Because there can be concurrent meetings, the solution consists of a sequence of character permutations $\Pi = \pi_1, ..., \pi_\lambda$, and a non-decreasing function describing the connection between points in time and the permutations in the solution $A : R \to 1, ..., \lambda$.

The exact algorithms mentioned above are:

1. a branching algorithm that searches for the shortest sequence of block crossings using iterative deepening depth-first search
2. a fixed-parameter tractable in the number of characters and works by performing a breadth-first search in an exponentially-large state graph

Another algorithm they compare is the integer linear programming formulation for pairwise crossings (not block crossings) in [6], which shows similar runtime.

They conclude that algorithm 1 above could still be practicle, since it is very memory efficient and has reasonable runtime for small instances.

**Applications** Since lines do not always need to be characters but any concept and meetings can also be any kind of temporal relationship, these visualizations allow many application. There is already a number of application of storyline visualizations, for instance script writing [1], eye-tracking in advertisment [2], visualizing evolving communities [14] , software project contribution [9] and topic analysis [3].

## 3 Approximation Algorithm

**Theorem** $d$-SBCM admits a $(3d^2(d^2 - 1))$-approximation algorithm that runs in $O(kn)$ time

In order to prove an approximation, it is necessary to design the algorithm in a way, that makes it possible to quantify the bounds for an arbitrary problem instance. Basically, this is done by finding a start permutation of the characters, which supports as many meetings as possible (= 'free'), and then use the count of unsupported meetings (= 'paid') together with their bounds to calculate the approximation.

### 3.1 Overview

The overall algorithm has the following three main steps (also see figure 2):

- Reduce input group hypergraph $H = (C, \Gamma)$ to an interval hypergraph $H_f = (C, \Gamma \backslash \Gamma_p)$ by deleting subset $\Gamma_p <= \Gamma$ of the edges of H
- Choose start permutation $\pi_0$ that supports all groups of this interval by $H_f$
- Incrementally create support for each deleted meeting of $\Gamma_p$

The vocabulary and concepts are not yet clear, but will follow in detail in the sections below.

The hardest part of this approximation algorithm is finding a good start permutation. This is done by solving the Hypergraph Edge Deletion Problem.

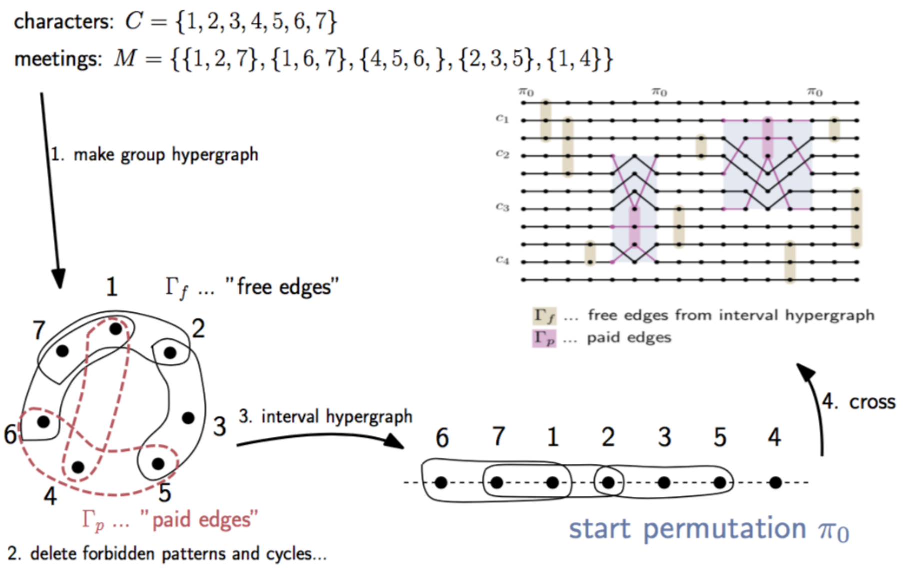
characters: $C = \{1,2,3,4,5,6,7\}$
meetings: $M = \{\{1,2,7\},\{1,6,7\},\{4,5,6,\},\{2,3,5\},\{1,4\}\}$
1. make group hypergraph
$\Gamma_f$ ... "free edges"
$\Gamma_p$ ... "paid edges"
2. delete forbidden patterns and cycles...
3. interval hypergraph
6 7 1 2 3 5 4
start permutation $\pi_0$
4. cross
$\Gamma_f$ ... free edges from interval hypergraph
$\Gamma_p$ ... paid edges

Figure 2: Algorithm overview

**Note:** Skip this section if the group hypergraph is dense

## 3.2 Hypergraph Edge Deletion Problem

**Definition** Given a hypergraph $H = (V, E)$, find a smallest set $E_p \subseteq E$ s.t. $H_f = (V, E \backslash E_p)$ is an interval hypergraph.

There are multiple things that need to be declared. A hypergraph is a generalized graph. Its edges are not restricted to have two vertices, but can also have more. Such an edge is called hyperedge and is often drawn as cycle, within its vertices are located. From now on, edge and hyperedge is used synonymously, and each hyperedge that is considered to be removed by the algorithm is put into the set of paid edges $E_p$.

Statements on the runtime use the following labels:

- n ... vertex count
- m ... hyperedge count

**Definition** An interval hypergraph $H = (V, E)$ is a special form of a hypergraph. There must exist a one-to-one function $f$, which maps elements of $V$ to points of a 'real line'. For each edge $e$, there must be an interval $I$ on the line, which contains all images of $e$, but no images that are not from $e$.

Eventually, this leads to a structure, that can be seen as interval or a spine of vertices in a specific order.

Interval hypergraphs have the characteristic that they don't contain any of the set of patterns as subhypergraphs given by figure 3.

$E_p$ is considered the 'paid' edges (meetings), whereas $E_f$ is considered the 'free' edges (meetings) as indicated in the section above. Since an interval hypergraph can be seen as a permutation of vertices, this is exactly what provides the start permutation for the storyline visualization.

**Create group hypergraph**
In order to apply the Hypergraph Edge Deletion algorithm, it needs to get an input instance $H = (V, E)$. For this, $V$ is set to the characters and $E$ is set to the meetings, beeing the so called 'group

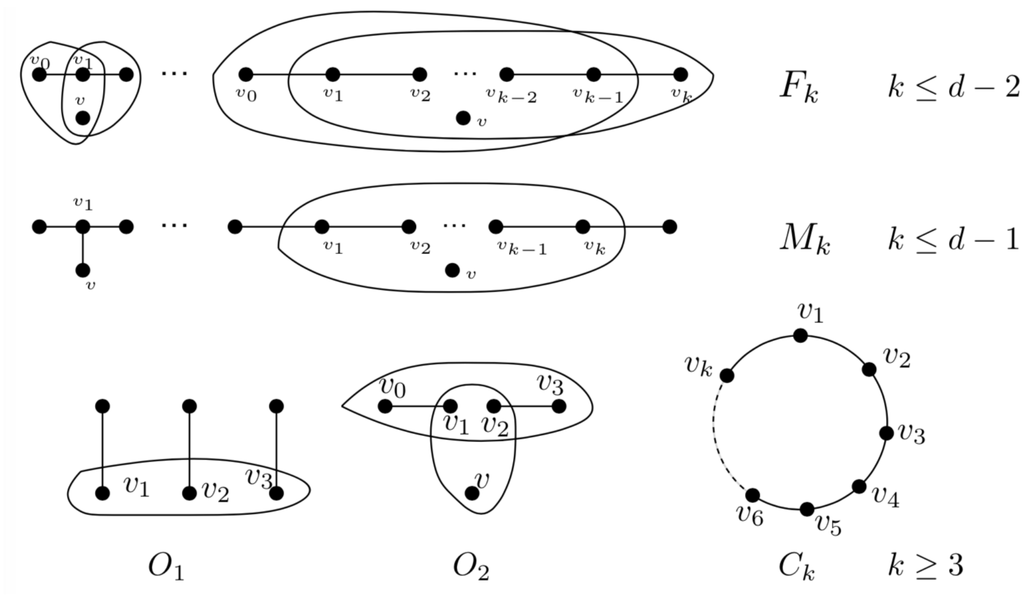

Figure 3: Forbidden sub-hypergraph patterns in interval hypergraphs

hypergraph'. Note that duplicate meetings only lead to one edge which are only relevant for the approximation calculation at a later point.

**Note on the runtime:** Filtering out duplicate meetings is done in $O(m)$ runtime. No other steps need more than linear time.

As interval hypergraphs don't have any of the given patterns, these patterns need to be removed from the hypergraph $H$. One of those patterns is a cycle of arbitrary length.

**Definition** A cycle $C$ in a hypergraph is a sequence of $l \geq 3$ hyperedges $e_1, ..., e_l$ with the property that there are vertices $v_1, ..., v_l$ such that for $2 \leq i \leq l$, (i) $v_i \in e_{i-1} \cap e_i$ (and $v_1 \in e_l \cap e_1$), and (ii) edge $e_i$ contains no vertex of $v_1, ..., v_l$ except for $v_i$ and $v_{i+1}$ (and $e_1$ conaints only $v_l$ and $v_1$).

This basically says, that not any random three or more overlapping hyperedges form a cycle. Given a hypergraph with three or more overlapping hyperedges in a sequence. For each pair of subsequent edges, choose one vertex in the intersection of the edge pair. Each edge must exactly contain two of such chosen vertices. This leads to the circumstance, that for some constellations of hyperedges, there might not exist a cycle including all those hyperedges. At the same time, multiple cycles can be possible if some of those hyperedges are not considered part of the cycle. Figure 4 shows an example for each, a valid and an invalid cycle.

### 3.2.1 Remove short cycles

The first part of the algorithm is removing cycles with a maximum length of $d$, which is the maximum meeting size. The idea to detect a cycle is to cut the cycle on one position and then try to find a path from one end of the cut cycle to the other end. In terms of detecting a cycle in a hypergraph, this works as follows:

For each edge $e \in H$, consider any pair $v_1, v_2 \in e$. Temporarily remove all hyperedges of $H$ that contain both $v_1$ and $v_2$. Then try to find a path from $v_1$ to $v_2$ via breadth-first search. If there exists such a path, then this path together with the temporarily removed hyperedges from the prior step forms a cycle and all involved hyperedges are removed.

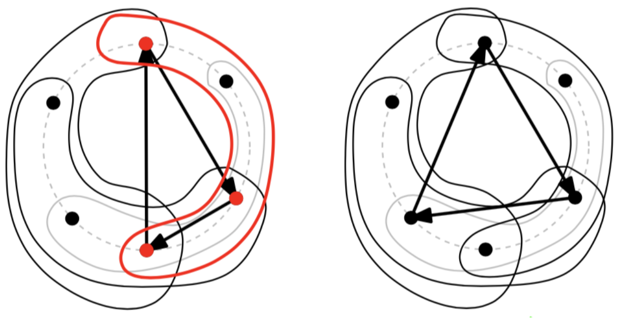

Figure 4: An invalid and a valid cycle, each consisting of 3 hyperedges in a hypergraph with 4 hyperedges

A path in a hypergraph is built similarly to a cycle in a hypergraph, but their ends are not connected. Thus, in order to find a path of maximum length $d$, one vertex in the intersection of two edges (one edge from the current path, on edge currently not part of the path) needs to be chosen to build up the path.

**Note on the runtime:** Finding the shortest path from one vertex to a certain other vertex is done in a breadth-first manner. The path length is restricted to the maximum length of $d$, which is constant. Because of this, traversing from one node of the path to the next is done constantly often, where each traveral needs linear time with regard to the meeting count, which is the maximum degree of a vertex. Overall, this algorithm step needs $O(m^2)$ runtime.

### 3.2.2 Remove remaining patterns (but long cycles)

The remaining patterns that need to be removed (in order to build an interval hypergraph), all have one thing in common. Each pattern contains a 'large edge' which contains all but 1 ($F_k$, $O_2$), all but 2 ($M_k$) or all but 3 ($O_1$) vertices of that pattern. Since edges are restricted to maximum size of $d$, all patterns but long cycles are bound to a maximum edge count of $d+1$. (TODO explain why $d+1$)

The idea now is to detect such big edges and then remove all edges of that pattern. Since an optimal solution must at least remove one edge of the pattern and there are a maximum of $d+1$ such edges in a pattern, the approximation ratio is 1 to $d+1$. (TODO: explain why all edges need to be remove?)

**Note on the runtime:** The check whether a given edge is the large edge of a pattern basically can be done in $O(m^2n^3)$ runtime, because only three further vertices of pattern can exist which are not in this large edge. However, because we already removed short cycles at this stage of the algorithm, certain long running branches of the algorithm can be ruled out, which removes the dependency on $n^3$ in the runtime. This applies for all given patterns, so we get an overall runtime of $O(m^2)$ for removing the remaining patterns but long cycles.

### 3.2.3 Remove long cycles

The only remaining pattern that might still be in the hypergraph $H$ is long cycles. An hypergraph that is free of the patterns removed so far is called $F$-free hypergraph, where $F$ is the set containing the following patterns:

- $F_1, ..., F_{d-2}$
- $M_1, ..., M_{d-1}$
- $O_1, O_2$

- $C_3, ..., C_k$ (short cycles)

Patterns $F_k$ and $M_k$ are restricted to a constant size because of their large hyperedge can contain at most $d$ vertices and starting from their smallest pattern form, there are only $d-2$ and $d-1$ different pattern sizes, respectively (see figure 3).

***F*-free hypergraph** $F$-free hypergraphs have some properties that help to prove the approximation and build an efficient algorithm.

**Lemma 1** Let $H = (V, E)$ be an $F$-free hypergraph and let $C$ be a cycle in $H$. Then two edges of $C$ can have a common vertex only if they are either consecutive in $C$ or if they share a common neighbor edge in $C$.

So, either two or three consecutive edges share a common vertex. This is important for a later step.

**Lemma 2** Let $H = (V, E)$ be an F-free hypergraph and let $C$ be a cycle in $H$. Then, for any hyperedge $e \in H$, it holds that either $e \subseteq V(C)$ or $e \subseteq V \backslash V(C)$.
$V(C) = \bigcup_{e \in C} e$ is the set of vertices of a cycle

From the lemma follows, that two cycles either have the same vertices, or they are disjoint from each other. Another implication is, that the hypergraph $H$ consist of possibly multiple 'connected components', which are either cyclic (cycles) or acyclic (paths). A connected component is a set of edges which share a common set of vertices.

**Cycle sets and cycle order** From Lemma 1 we now that a cycle contains only edges where at most three of them share a common vertex. Other edges $\in H$ may overlap, but are not part of the cycle. Thus, we can look at the case of three consecutive edges in such a cycle in isolation without loss of generality and then generalize this to the whole cycle.

Three overlapping hyperedges can be separated into five different subsets. These are (also see figure 5):

1. $e_1 \backslash e_2 \cup e_3$
2. $e_1 \cap e_2 \backslash e_3$
3. $e_1 \cap e_2 \cap e_3$
4. $e_2 \cap e_3 \backslash e_1$
5. $e_3 \backslash e_1 \cup e_2$

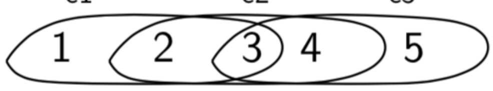

Figure 5: Cycle-sets of three overlapping hyperedges $e_1, e_2, e_3 \in$ cycle $C$

Recapitulating the 'real line' metaphor when defining interval hypergraphs, we need to find an order of vertices so that all edges in $H$ are supported. For this to happen, we can conclude that two vertices from different sets above must follow an ordering rule. Vertices of set 1 must be placed before vertices of set 2, vertices of set 2 must be placed before vertices of set 3 and so on. Within the same set, there is no such ordering rule. However, there might still exist edges in $H$, which partly overlap such sets and thus could violate the condition of $H$ beeing an interval hypergraph (see figure 6). Therefore, we need to refine this order in the next step.

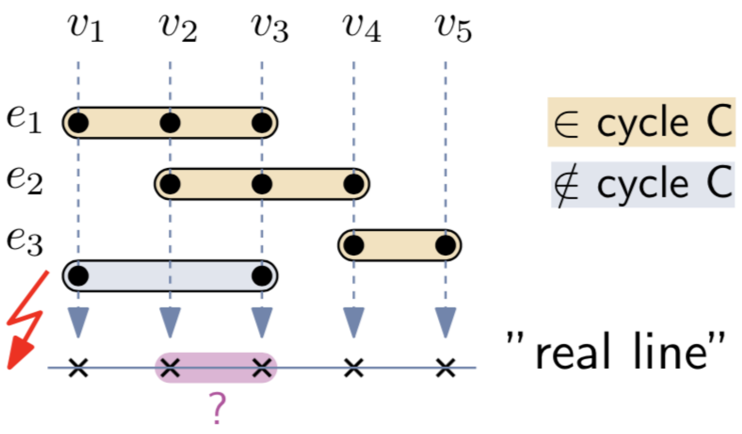

Figure 6: Possible cycle order violation before cell refinement

Generalizing the three edges of a cycle to the whole cycle, we call these sets the 'cycle-sets' and the restriction to the vertex positions the 'cycle-order'.

**Cell-order**
In order to avoid the violation of the interval hypergraph condition, we refine the cycle-order. For this we use another data structure which we call cells. A cell is a set of vertices. First, the cells are initialized to the cycle-sets (see 7). In the next step, we insert an edge $e' \in H \backslash \Gamma_p$ that contains vertices of two or more cells. Two calls might be split in up to four cells and thus, refining the order of the vertices (see 8). After this step, there are only edges left that are fully contained in single cells.

Of course, the cells are also in a cyclic order, named the cell-order.

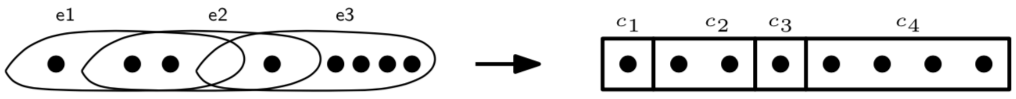

Figure 7: Cells are initialized to the cycle-sets

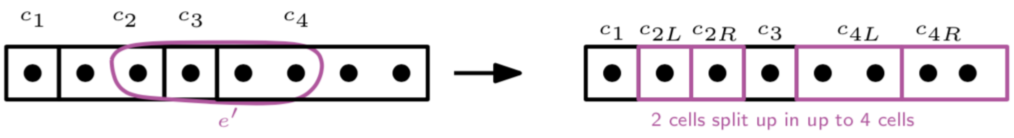

Figure 8: Cell split (order refinement) by considering new edge

**Note on the runtime:** Building up the cycle sets for the cell-order refinement is done for each cycle. This includes choosing a starting edge. Each three consecutive, overlapping hyperedges amount to exactly five possible subsets (each constructed in constant time due to constant d), therefore a constant factor when applying this to the whole cycle. Since there are at most $m$ such hyperedges in a cycle and each cycle is either disjoint or has the same vertices (only one such cycle with the same vertices needs to be considered), this can be done in linear runtime $O(m)$. For each edge that is not part of a cycle and thus not yet in the initial cells, merge them with the existing cells by potentially splitting up two cells in up to four cells. Searching for the correct cells to split amounts to $O(m)$. Doing this for each missing edges then amounts to an overall runtime of $O(m^2)$.

**Optimal cutting point for a cycle**

We now know that the vertex order does not violate the 'real line condition' of an interval hypergraph but potentially being cyclic. The last step is now to cut the cycle at a point in order to retrieve the interval hypergraph. So, in order to get a possibly good approximation, the optimal cutting point must be found such that the least amount of edges have to be removed.

Since the cells are all made from the edges, we can count the edges that overlap the cell boundaries. We cut between cells where the least edges overlap by removing those edges.

**Note on the runtime:** The paper says this can be done in linear runtime $O(m)$ without more information. Currently, I cannot imagine how this could be done.

## 3.3 Approximation

### Maximum block crossings for paid edges

Consider a meeting that is not supported by the start permutation. Therefore, there is a respective edge in the set of paid edges $\Gamma_p$. In order to support this meeting we can bound the number of needed block crossings to a maximum. The idea is to fix one line and bring all of the maximum $d-1$ other lines close to each other (see figure 9).

Bringing one line to a certain position in the permutation needs at most one block crossing, because this one line forms on arbitrary sized set of lines and the lines between this line and the target neighbor line form the second arbitrary sized set of lines. This is the only movement this line needs to do. If it is already next to the other lines, there is no block crossing needed. We do this for all $d-1$ lines of the meeting that are not fixed. Thus, we get a bound of maximum $d-1$ block crossings for supporting the meeting. In order to support the free meetings between the paid meetings we need to revert the block crossing to get the start permutation again. Thus, we need a maximum of $2(d-1)\Gamma_p$ block crossings. Since multiple occurrences of the same meeting got lost when creating the group hypergraph, a variable $\alpha$ being the maximum repeated meeting count of each meeting. Finally, we get a maximum of $\alpha 2(d-1)\Gamma_p$ block crossings.

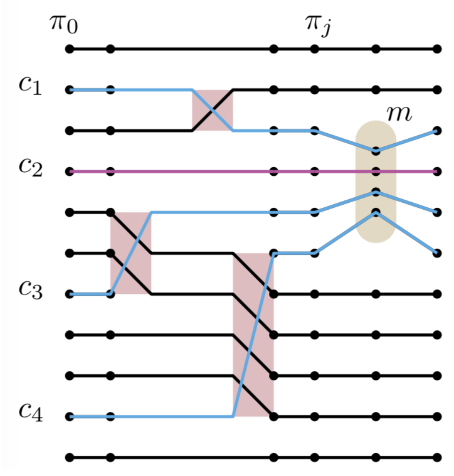

Figure 9: Block crossings needed to support a meeting

**Minimum block crossings for paid edges** One block crossing brings together at most three pairs of lines, which were not neighbors before (see figure 10). A meeting $m_{\text{new}}$ that is newly supported after a block crossing must contain at least one of these pairs. We denote $d_1$ the number of characters in meeting $m_{new}$ above this pair and $d_2$ the number of character in $m_{new}$ below this pair. Obviously, $d_1 + d_2 \leq d$ since the meeting size is restricted to $d$. Moreover, $d_1, d_2 \geq 1$, because the pair itself is included in $d_1$ and $d_2$ respectively. Then, for each $d_1 \geq 1$ we get $(d - d_1)$ possible values for $d_2$ considering each possible meeting size, thus $\sum_{d_1=1}^{d-1}(d - d_1) = \frac{d(d-1)}{2} \leq \frac{d^2}{2}$ newly supported meetings for one block crossing. Since we have up to three such pairs of lines which become neighbors after a block crossing, we get a total maximum of $3\frac{d^2}{2}$ newly supported meetings after a block crossing. The minimum count of block crossings needed is $\frac{\#\text{unsupported meetings}}{\text{maximum }\#\text{newly supported meetings per block crossing}}$, which amounts to $\frac{2|\Gamma_p|}{3d^2}$ an optimal solution needs minimally.

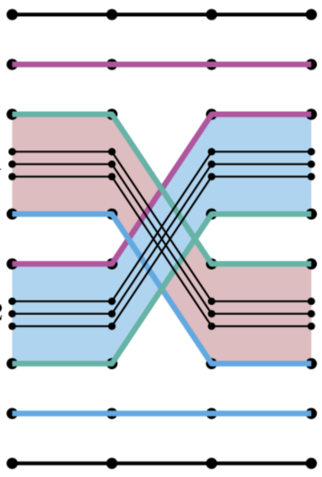

Figure 10: New neighboring lines after block crossing

**Putting everything together**

In the Hypergraph Edge Deletion algorithm we learned the approximation ratio of 1 to $d + 1$ compared to a possible optimal solution. This can be formalized as $|\Gamma_p| \leq (d + 1)|\Gamma_{OPT}|$. Together with the ALG $\leq \alpha 2(d - 1)|\Gamma_p|$ from the maximum block crossings for paid edges part, this amounts to ALG $\leq \alpha 2(d - 1)(d + 1)|\Gamma_{\text{OPT}}|$. In the last step, we can take the OPT $\geq \frac{2|\Gamma_{\text{OPT}}}{3d^2}$, umformen it to $|\Gamma_{\text{OPT}}| \leq \text{OPT}\frac{3d^2}{2}$. Putting everything together, we get the desired approximation ALG $\leq$ $\alpha 3d^2(d^2 - 1)$OPT of the algorithm.

## 3.4 Runtime

Assuming we already have a starting permutation of the lines, checking and changing the permutation of characters through block crossings can be done in $O(nm)$, where $n$ is the characters count and $m$ is the meetings count. Interval Hypergraph Edge Deletion can be done in $O(m^2)$, since all of the consecutive parts of the algorithm have a runtime of $O(m^2)$ or less (see the 'Note on the runtime' paragraphs in the approximation algorithm sections above). Therefore, this would amount to total runtime of $O(m^2)$ for the whole algorithm, but this would contractict the theorem, which says the runtime is in $O(nm)$. We can do a trick: Since the start permutation supports a maximum of $dm$ meetings, even an optimal solution must delete more than half of the edges when $m \geq 2dn$. Therefore, do the Interval Hypergraph Edge Deletion algorithm for retrieving the start permutation only if $m \in O(n)$, which we call 'sparse' instances. For dense instances, use a random start permutation and skip the Interval Hypergraph Edge Deletion algorithm, as this also leads to a good approximation without it. For sparse instances where $m \in O(n)$, $O(m^2) \subseteq O(mn)$, as would also hold for the dense instances when using the random start permutation. Therefore, we get the desired runtime of $O(mn)$.

## 4 Open Problems and Discussion

The field of storyline visualizations has multiple research tracks, not necessarily focusing on the same things. There are approaches that consider multiple aspects of good storyline visualizations (e.g. [11]), others focus on single aspects such as pairwise crossings (e.g. [6]) or block crossings (e.g. [12], [13]). The latter ones have so far each introduced their own problem definition for their tailored algorithm. I'm wondering whether this is a common practice in the field of algorithmics, as it seems that problems in algorithmics tend to be well-defined. On the one hand the community would try to find a common problem definition because it improves comparability of solutions, on the other hand through my limited view, algorithms optimizing the same thing (in our case storyline visualizations) would all share the same input structures and the same output structures. This makes a common problem definition just natural. If one uses a custom problem definition, either they just solve a more specific version of the problem and it needs to be questioned how useful this is in practice, or the problem definition used so far is insufficient. Of course, research is not just about doing what directly impacts real-world practices, but can be indirectly useful, as it is often not known beforehand if a research field leads to something better than the existing practices. It is also worth knowing that certain research areas are more promising and others less. Maybe these multiple problem definitions are also the result of a more or less subjective problem. What is a good storyline visualization? Of course, there can be user studies about which design criteria have positive impact and which tend to have a negative impact. But it is a field of infinit possibilities for design considerations, and eventually the diagrams are interpreted by humans, in a more subjective than objective way.

The thing with this paper is, there aren't any user studies about readability and understandability of storyline visualizations minimizing block crossings, so far. Obviously, it is an interesting and novel idea with possibly much improvements to bring to the current practices, but one of the next steps should probably be doing user studies. Moreover, it also remains unclear whether minimizing block crossings has impacts on other design aspects that improve storyline visualizations. This concerns the block crossings minimization algorithms introduce so far in particular, since they only consider this single aspect, neglecting other design aspects entirely. For this reason, taking Tanahashi's work on design considerations [11] as example, this would also be a very interesting approach worth trying next.

There are certainly still many points and open questions left that are worth discussing in this very new field of minimizing block crossings in storyline visualizations. Here is one more thing that I personally would like to see in future: A greedy heuristic for SBCM also including an approximation prove for it. Just a quick way to draw a solution that hopefully can meet one or more important design aspects.

To me, storyline visualizations is a very interesting and promising topic with many possible applications remaining to be discovered. I will definitely check some time in the future how this research field evolved, and maybe I will have a good reason for using one of the algorithms that are developed by then.

## References

[1] Storyline creator. `https://www.storylinecreator.com/`. Accessed: 2017-12-10.

[2] J. T. Balint, D. Arendt, and L. M. Blaha. Storyline visualizations of eye tracking of movie viewing. In *2016 IEEE Second Workshop on Eye Tracking and Visualization (ETVIS)*, pages 35–39, Oct 2016.

[3] Weiwei Cui, Shixia Liu, Li Tan, Conglei Shi, Yangqiu Song, Zekai Gao, Huamin Qu, and Xin Tong. Textflow: Towards better understanding of evolving topics in text. *IEEE transactions on visualization and computer graphics*, 17(12):2412–2421, 2011.

[4] Martin Fink, John Hershberger, Subhash Suri, and Kevin Verbeek. Bundled crossings in embedded graphs. In *Latin American Symposium on Theoretical Informatics*, pages 454–468. Springer, 2016.

[5] Martin Fink and Sergey Pupyrev. Ordering metro lines by block crossings. In *International Symposium on Mathematical Foundations of Computer Science*, pages 397–408. Springer, 2013.

[6] Martin Gronemann, Michael Jünger, Frauke Liers, and Francesco Mambelli. *Crossing Minimization in Storyline Visualization*, pages 367–381. Springer International Publishing, Cham, 2016.

[7] Irina Kostitsyna, Martin Nöllenburg, Valentin Polishchuk, André Schulz, and Darren Strash. *On Minimizing Crossings in Storyline Visualizations*, pages 192–198. Springer International Publishing, Cham, 2015.

[8] Randall Munroe. Movie narrative charts. *Xkcd. com. http://xkcd. com/657*, 2009.

[9] Michael Ogawa and Kwan-Liu Ma. Software evolution storylines. In *Proceedings of the 5th international symposium on Software visualization*, pages 35–42. ACM, 2010.

[10] Y Tanahashi. Movie interaction dataset, 2012.

[11] Y. Tanahashi and K. L. Ma. Design considerations for optimizing storyline visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2679–2688, Dec 2012.

[12] Thomas C. van Dijk, Martin Fink, Norbert Fischer, Fabian Lipp, Peter Markfelder, Alexander Ravsky, Subhash Suri, and Alexander Wolff. *Block Crossings in Storyline Visualizations*, pages 382–398. Springer International Publishing, Cham, 2016.

[13] Thomas C van Dijk, Fabian Lipp, Peter Markfelder, and Alexander Wolff. Computing storyline visualizations with few block crossings. *arXiv preprint arXiv:1709.01055*, 2017.

[14] Corinna Vehlow, Fabian Beck, Patrick Auwärter, and Daniel Weiskopf. Visualizing the evolution of communities in dynamic graphs. In *Computer Graphics Forum*, volume 34, pages 277–288. Wiley Online Library, 2015.