Yarn: Generating Storyline Visualizations Using HTN Planning

Kalpesh Padia Kaveen Herath Bandara Christopher G. Healey*

Department of Computer Science
North Carolina State University

Figure 1: Yarn visualization depicting both actual (reality) and alternate (diegetic) timelines. Narrative progresses over time along the x-axis. Nodes on the y-axis represent character sub-goals, and vertical node clusters represent events. Solid and dashed lines trace the path of the main characters through the events on the reality and diegetic timelines, respectively

ABSTRACT

Existing storyline visualization techniques represent narratives as a node-link graph where a sequence of links shows the evolution of causal and temporal relationships between characters in the narrative. These techniques make a number of simplifying assumptions about the narrative structure, however. They assume that all narratives progress linearly in time, with a well defined beginning, middle, and end. They assume that at least two participants interact at every event. Finally, they assume that all events in the narrative occur along a single timeline. Thus, while existing techniques are suitable for visualizing linear narratives, they are not well suited for visualizing narratives with multiple timelines, nor for narratives that contain events with only one participant. In this paper we present Yarn, a system for generating and visualizing narratives with multiple timelines. Along with multi-participant events, Yarn can also visualize single-participant events in the narrative. Additionally, Yarn enables pairwise comparison of the multiple narrative timelines.

Keywords: HTN Planning, Narratives, Storyline Visualization.

Index Terms: Human-centered computing—Visualization—Visualization techniques; Human-centered computing—Visualization—Visualization application domains—Information Visualization

1 INTRODUCTION

A story or a narrative is an ordered sequence of connected events in which one or more characters (or entities) participate [19]. The events in the narrative take place at various locations, and together with the entities define the relationships that shape the course of the narrative. Understanding the evolution of these entity relationships is key to comprehending and analyzing how the narrative unfolds. To this end, storyline visualizations have been developed to represent a narrative based on the causal and temporal patterns of the entity relationships.

Existing storyline visualization techniques, inspired by Munroe's movie narrative charts [22], represent narratives as node-link graphs. These techniques lay out narrative events chronologically, from left to right, with each entity represented as a line running from one event to another. Events are shown as nodes in the storyline, and a link between a pair of nodes represents an entity that participates chronologically in both events.

Initial storyline visualization techniques [23,24,34] produced an aesthetically pleasing visualization, but at the expense of time. Liu et al. [19] described an optimization strategy, called StoryFlow, for fast generation of storyline visualizations. StoryFlow creates a visualization using a four stage pipeline that generates an initial layout, then performs ordering and alignment of nodes, and compaction of the overall layout to improve its appearance.

While these techniques can produce aesthetically pleasing and legible storylines, they do so by making simplifying assumptions about the structure of the narrative. Because of this they may not support more complex, real-world storytelling and analysis tasks. First, existing techniques assume that at least two entities participate at every event in the timeline. However, many real world narratives involve situations where the entities generate events without interaction, or by interacting with entities that are not present at the same location. This creates single-entity events in the narrative that are not supported by existing techniques. Second, many narratives involve character entities making choices. These choices directly influence the evolution of the causal relationships that shape the outcome of the narrative. For many real-world analysis tasks, it is important to not only visualize the narrative as it unfolds (reality timeline) but to also include alternative choices that can lead to alternate outcomes (diegetic timelines). Existing techniques visualize only reality timelines. They provide no support for diegetic narratives. Third, existing techniques assume that the narrative progresses linearly in time and has a well defined beginning, middle and end. While this makes them suitable for visualizing traditional, linear narratives such as a movie's plot, they cannot visualize non-linear narratives such as narratives with flashbacks or flash forwards, narratives with parallel distinctive plot lines, or narratives that present events from their characters' point of view.

In this work we present Yarn, a new system for automatic narrative construction and visualization. Unlike existing systems based on Hierarchical Task Network (HTN) like Cavazza et al. [5], our system generates all possible narratives, rather than constructing individual alternative narratives "on demand" based on user interac-

*e-mail:healey@ncsu.edu

Graphics Interface Conference 2018
8-11 May, Toronto, Ontario, Canada

tion with an initial reality timeline. This provides three advantages: (1) diegetic timelines are available for a user to examine, select from, and visualize immediately; (2) reality and diegetic timelines can be visually compared to search for similarities and differences; and (3) support for non-linear point-of-view narratives. Finally, our use of the new WebWorker-based parallelism significantly improves overall performance during the narrative generation stage. Our HTNs were specifically designed to take advantage of this capability.

Based on the above, our work makes the following novel contributions:

1. An efficient method for generating all possible timelines in a narrative using HTN planning.

2. A storyline layout for visually depicting and comparing non-linear point-of-view narratives with multiple timelines.

2 Related Work

In this paper we discuss a new system for narrative generation and visualization. Here we present some of the related work in these fields.

2.1 Automated Narrative Construction

Researchers in the field of narrative theory draw ideas from various fields, including literary theory, linguistics, cognitive science, folklore, and gender theory to define what constitutes a narrative and how it is different from other kinds of discourse, such as lyric poems, arguments, and descriptions [3, 7, 14, 26, 29]. They have studied narratives using numerous approaches such as rhetoric (discourses that inform, argue with, convince or motivate audiences), pragmatic (discourses that convey, request or perform social actions such as complaints, suggestions, compliments, requests, apologies, refusals, and warning), and antimimetic (discourses that are expressed or conveyed in non-traditional forms) to define narratives in multiple ways. All definitions, however, agree that a narrative organizes spatial and temporal data into a cause-effect chain of events with a beginning, middle and end. Each narrative has two parts. The first is the *fabula* or story comprising a chain of events and its existents, defined as characters and settings. The second is *sjužet* or discourse, defined as an expression or means by which the story is communicated. While *fabula* deals with the organization of the content of a narrative, *sjužet* deals with the manifestation—appearance in a specific material form: oral, written, musical, cinematic, and visual—and transmission of the narrative.

In recent years narrative theory has generated significant interest among computer scientists, especially in the field of artificial intelligence (AI), computational linguistics, and game design. Researchers have proposed numerous systems for automatic generation of narratives, such as Tale-Spin [21], Minstrel [37], Mexica [25], Virtual Storyteller [36], Fabulist [27], and Suspenser [8]. These systems identify a thematic pattern in a pre-existing corpus of *fabula* to generate narrative text, by selecting and ordering fabula events to create a linear progression of the story.

Significant research efforts have also been directed towards developing narrative engines which can automatically generate *sjužhet* (and thus, the narrative) based on the end goals specified in the *fabula* [9, 20, 28]. These systems represent the *fabula* as a collection of state spaces that are searched to find a sequence that satisfies the end goal. The system models *fabula* semantics such as timelines, states, events, characters, and goals as data objects. These objects can be queried, manipulated, and arranged using user-defined *sjužhet* assertions to generate the narrative text.

Two approaches are commonly used to represent *fabula* in such automatic narrative generation systems. In the first approach, a STRIPS-like [11] formalism is adopted to represent the *fabula* as a collection of world models for all possible scenarios in the narrative. A world model is constructed as a set of well-formed formulas using first-order predicate calculus. A well-formed formula is also used to state the goal condition. A set of operators enumerate possible actions and their effect on the world models. Various AI planning techniques apply these operators to the world model collection to find a model that achieves the stated goal condition . The sequence of events described in the solution model generate a narrative that achieves the goal condition.

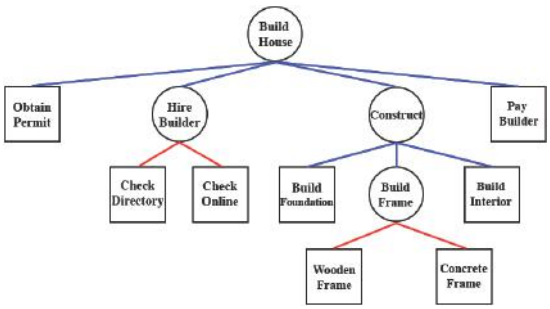


Figure 2: A hierarchical task network decomposing the task "Build House". Rectangular nodes represent primitive actions, and circular nodes represent (sub-)tasks in the decomposition. Red lines show decomposition for an OR node, while blue lines show decomposition for an AND node

In the second approach, the *fabula* is represented using a hierarchical task network (HTN) formalism. HTNs are networks that represent ordered task decomposition, based on the idea that many tasks in real life have a built-in hierarchical structure. The top-level task in an HTN is typically the main goal. Each task can be decomposed into sub-tasks, which can be further decomposed into smaller tasks until all tasks are represented as primitive actions. HTNs are commonly built using AND/OR graphs. When a task can have several possible decompositions, it is represented using an OR node. Each sub-task is a valid decomposition for the parent task. When a task has several decompositions that can be ordered in some fashion to complete the task, it is represented as an AND node. Thus, an HTN can be seen as an implicit representation for the set of possible solutions for a task [10]. Fig. 2 shows an HTN for the task "Build House". Rectangular nodes represent primitive actions, and circular nodes represent sub-tasks in the decomposition. Red lines show decomposition for an OR node, while blue lines show decomposition for an AND node. The final decomposition for the goal task contains all primitive actions, returned from a depth-first search of the HTN. As narrative descriptions can be naturally represented as task decompositions [6,17], they are well suited for representation using HTNs. Based on this idea, a number of techniques [1,5,6,35] have been developed that employ HTN planning to generate a narrative. These techniques first define a narrative goal, then decompose it to find a sequence of primitive actions that describe events in a narrative timeline. Existing techniques, however, generate only one such timeline even when the narrative goal could have multiple possible decompositions, each representing a possible alternate timeline.

Our approach builds on these techniques, improving them to support efficient generation of all possible narrative timelines, provide a method to visualize multiple timelines simultaneously and supporting a subset of non-linear narratives.

2.2 Visualizing Narratives

Static visualizations have long been used to support storytelling, usually in the form of diagrams and charts embedded in a larger body of text. In this format, the text conveys the story, and the image typically provides supporting evidence or related details. More recently, visualizations have been designed with the purpose of conveying a "data story," and guiding the viewer through the narrative generated from analysis of the data [18]. These visualizations have

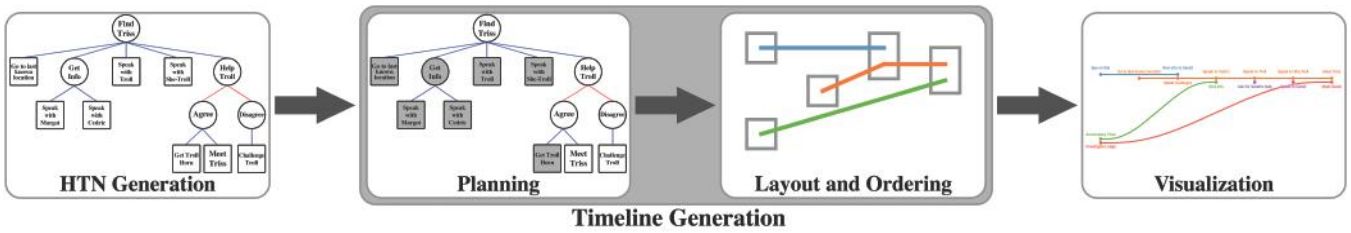


Figure 3: Overview of Yarn's four stage pipeline: (1) generate Hierarchical Task Networks for each character entity in the narrative; (2) HTN planning to generate timelines; (3) compute layout and ordering of entities in each timeline; and (4) create visualizations inside a browser window

been termed "narrative visualizations" [31], and have been extensively studied to explore how elements of narrative theory can be applied to identify patterns in visual layout to provide recommendations, or to evaluate the effectiveness of the visuals in the presentation and analysis [15, 30, 31].

Recently, another technique has also emerged to help users better understand and analyze a complex story by presenting them visually. This technique, storyline visualization, is inspired by Munroe's Movie Narrative Charts [22], and represents narratives as node-link graphs. Events in the narrative are visualized as nodes, and are laid out chronologically from left to right. Each character, or entity, is shown as a line running from one event to another. A link between a pair of nodes represents an entity that participates chronologically in both events. Interactions among the entities during different events define entity relationships.

Storyline visualizations differ from narrative visualizations in important ways. As noted before, narrative visualizations emphasize combining a data story with graphics. They present related content, in blocks, to provide clear and logical transitions without regard for temporal ordering of events. Storyline visualizations, on the other hand, are designed as an alternative manifestation of a narrative's *sjužet*. They present narrative events on a temporal timeline that follows the narrative's progression. This allows users to better understand the evolution of entity relationships from the beginning to the end of a story. This can be very important in many applications such as information exploration and understanding, interpersonal communication and storytelling, and media analysis [4, 13]. Narrative visualizations, therefore, are "visual data stories", while storyline visualizations are "visual narrative summaries."

When visualizing a real-world narrative, storyline visualizations are generally simplified by imposing application-specific constraints, thereby achieving success at the cost of loss of generality [19]. More recent research efforts have focused on developing a generic storyline visualization tool. Tanahashi and Ma presented a storyline layout technique [34] based on a genetic algorithm. They also presented an extension of their technique to generate storyline visualization for evolving narratives [33]. While their techniques create an aesthetically appealing and legible storyline visualization, they take considerable time to create the layout. Liu et al. [19] presented a strategy called StoryFlow that formulates the difficult problem of creating an effective storyline layout as an optimization problem, creating an aesthetically appealing visualization significantly faster than Tanahashi and Ma. These techniques, however, assume that at least two entities participate at every event in the timeline. Further, they provide no support for narratives with multiple timelines (diegetic narratives).

Additional techniques have been developed for visualizing narratives with multiple timelines. SemTime [16] is a temporal visualization technique that uses distinct types of directed edges and time independent stacking of multiple timelines to show relationships between events. However, it is not well suited for visualizing large narratives due to the amount of visual clutter created by line crossings. World Lines [39] is a technique for visualizing different narrative timelines generated by performing a "what-if" analysis on the narrative. It uses a combination of simulation techniques to generate multiple timelines, then visualizes them in a "horizontal tree-like visualization" that depicts the causal relationships between different timelines. However, the lack of a representation of individual entities means it is difficult to identify interactions between entities within a timeline. This makes the technique better suited for visualizing how the narrative timelines branch from each other rather than visually summarizing and comparing events within the timelines.

3 Yarn Overview

We have developed Yarn for automatic construction and visualization of multiple narrative timelines. In our approach we use HTN planning for automatic generation of all possible narrative timelines. Compared with existing HTN-based narrative generation systems like Cavazza et al. [5], no user interaction is required for generation of diegetic timelines. Our approach also visualizes the generated timelines using a storyline layout that represents all events in the timeline. To achieve this, we:

1. Represent the narrative HTN as a collection of entity HTNs, one for each character in the narrative.
2. Use a WebWorker-based HTN planner for decomposing the entity HTNs in parallel to evaluate all possible choices available to the entities and their corresponding outcomes identifying the reality timeline and possible diegetic timelines in the narrative. This is efficient and allows representation of both single and multi-entity events. Our HTN planner also supports causal events in the narrative, where past actions affect future outcomes.
3. Visualize each timeline by creating a storyline layout with minimal line crossings.
4. Allow pairwise comparison of narrative timelines for better comprehension of a timeline's progression and event outcomes.

Yarn's narrative construction and visualization pipeline (Fig. 3) consists of four stages: HTN generation, planning, layout and ordering, and visualization. Yarn is designed to run within a web browser, and each stage is fully implemented using JavaScript. In the first stage, Yarn generates an in-memory HTN representation of the narrative from the input functions. Next, a multi-threaded HTN planner concurrently generates plans for each entity in the narrative. An initial node-link graph layout is generated from these plans in the Layout and Ordering stage. An ordering algorithm is then run to compute a final layout with minimum line crossings. The output from this stage represents one narrative timeline. These two stages are repeated to generate all possible timelines in the narrative. Finally, the timelines are sent to the Visualization stage where each timeline is drawn inside a web browser, on demand. Two timelines can be displayed to enable visual comparison of the events in each timeline.

4 Narrative Generation and Visualization

As illustrated in Fig. 3, Yarn begins by generating an HTN for narrative entities, followed by narrative planning, timeline generation, and visualization.

4.1 HTN Generation

In our system, we represent a narrative as a collection of HTNs, one for each entity in the narrative. In order to use these graphs for timeline generation, we need to first convert them into alternate representations that our JavaScript HTN planner can use.

We start by expressing the narrative's *fabula*—domain description and initial state information—using a JavaScript map. Each object in the map is a key–value pair where the key is any entity, prop or trigger condition, and the value represents its state at the beginning of the narrative. A complete map with all initial state information represents a narrative state map which can be accessed, modified and updated by the operator functions during the planning stage.

Next, we express the task (end-goal), sub-tasks and primitive actions in each entity HTN using JavaScript functions. We categorize these functions into three types.

1. **Operator functions:** Each primitive action in the HTN, when executed, represents an event in the narrative. An action may also have certain pre-requisites which must be met for the action to return successfully after execution. Both successful and unsuccessful executions of the primitive action create events in the narrative.

   We define operator functions as JavaScript functions that are a manifestation of primitive actions, complete with conditional checks to detect fulfillment of pre-requisites, and if required, wait loops to wait for pre-requisite fulfillment. Execution affects the narrative state by creating an event upon completion.

2. **Method functions:** Each AND/OR node in the HTN represents a sub-task. AND sub-tasks can be completed in exactly one way by performing all of the sub-tasks/primitive actions directly beneath it. OR sub-tasks can be performed in more than one way, since execution of any sub-task/primitive action directly beneath an OR node is a valid decomposition for the sub-task.

   We define method functions as JavaScript functions that are a collection of AND/OR nodes in the HTN. When a method function represents an AND node, it calls subordinate method or operator functions in a specific order as specified by the HTN. Each of the functions must return successfully for the method function to return success. When a method function represents an OR node, it calls each subordinate method function or operator function specified by the HTN. If a function returns successfully, the method function itself returns success. If all functions return unsuccessfully, the method function returns a failure to find a (sub)plan.

   In our implementation of method functions for OR nodes, we use a random order to call subordinate functions. Normally each subordinate function is assigned an equal probability, and a random number r, $0 \leq r \leq 1$, governs which of the functions is selected for execution. We could, however, assign the functions different probabilities to simulate the probabilistic occurrence of narrative events. Additionally, by affecting the probability of execution of the possible subordinate functions, we can implement causal relationships in the narrative.

3. **Task functions:** Each goal node in the HTN is an AND/OR node that represents the task we are trying to decompose. We define task functions as special method functions that only call other method or operator functions and cannot be called by any other function. A task function serves as the entry point for our planner when finding a plan decomposition for an HTN. A plan is identified successfully only if all functions called by a task function return successfully.

After representing nodes in each HTN using the functions described above, we create a task list as a JavaScript array of task functions, one for each entity in the narrative. This task list, along with the narrative state map and the operator and method functions serve as input for the planning stage.

4.2 Planning

Character entities in real-world narratives follow unique paths to accomplish their individual goals. Along the way they participate in events with other entities (multi-entity events), or create events independently. To mimic this method of event creation, we run our HTN planner concurrently for each task in the task list. In our implementation, the planner finds the decomposition of a task by performing a left-to-right depth-first search of the associated HTN, further decomposing any sub-tasks encountered, and executing any primitive actions in the process. If a primitive action fails, the algorithm backtracks to find a suitable alternative. The complete decomposition of a task contains an ordered sequence of execution for primitive actions where a primitive action on the left sub-tree of a node in the HTN appears before a primitive action on its right subtree. Each unique sequence represents an alternate decomposition for the task. For example, one of the four possible plan decompositions for the task "Build House" in the HTN shown in Fig. 2 is made up of the primitive actions: *Obtain Permit, Check Directory, Build Foundation, Concrete Frame, Build Interior* and *Pay Builder*.

In order to perform concurrent planning of tasks in the task list, we use JavaScript WebWorkers to create background worker threads that do not interfere with the main program thread. We spawn threads, one for each task function in the task list, that call the planner concurrently to decompose the task functions. These worker threads, however, do not have access to global or shared variables. This is problematic because during task decomposition operator functions must update the state map to generate narrative events.

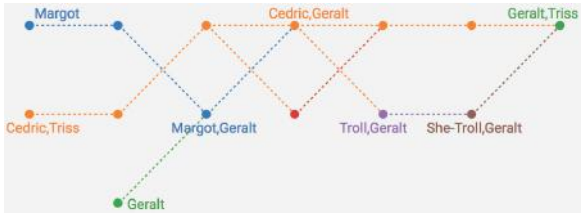
To solve this, we use IndexedDB, a type of browser storage that is accessible by WebWorkers. IndexedDB allows us to store both local and session related data within a browser session. In our implementation, we use it to store the state map. An operator function called by one thread can change the state map in a way that is visible to operator functions in other threads. To store narrative events we create an event list record in IndexedDB. After updating the state map, the operator functions store an event ID, entity name, event label and optional text to describe the narrative event, in the event list record. In order to ensure atomicity of all operations we implement mutex locks for reading and writing the IndexedDB using JavaScript Promises. After all threads finish their execution, the state map and the event list record are the final output of the planner, representing one narrative timeline.

To calculate the time complexity for our planner, let $G = (S, A, L)$ be the graph representation of an HTN for an entity, where S is the set of (sub-)task nodes represented as AND/OR nodes in the graph, A is the set of primitive action nodes in the graph, and L is the set of links in the graph. Let n be the number of *fabula* elements that are represented as keys in the key–value pairs stored in the narrative state map.

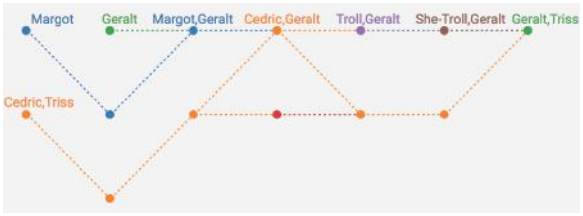
In the worst-case we perform a recursive depth-first traversal of the entire HTN graph to generate a plan decomposition, where we visit every node in S and A, and execute the associated functions. Visiting a node in S or A is the same as traversing an edge in the graph and can be performed in $O(1)$ for each edge.Visiting all nodes, therefore, can be performed in $O(1) * |L| = O(|L|)$.

Executing a function for a node in S involves removing it from the stack of functions to be called and adding a subordinate method or operator function. Both these operations can be performed in $O(1)$ allowing the function itself to be performed in $O(1)$. Executing functions for all nodes in S, therefore, can be performed in $O(1) * |S| = O(|S|)$.

Executing a function for a node in A involves updating the narra-



(a) Initial layout generated using matrices created from the output of the planning stage.



(b) Layout generated after running barycentric graph layout algorithm to re-order events vertically along the y-axis.

(c) Layout generated after removing dummy nodes from the event layout generated in the previous step.

(d) Final layout after introducing entities in the event layout from previous step and performing entity ordering.

Figure 4: Illustration of various steps of the layout and ordering stage applied to a timeline for the example narrative in Sect. 5.1

tive state map stored in IndexedDB to create a new event. This can be performed in $O(log(n))$. Executing functions for all nodes in A, therefore, can be performed in $O(log(n)) * |A| = O(|A| \cdot log(n))$.

The time complexity for generating a plan decomposition, then, is $O(|L|) + O(|S|) + O(|A| \cdot log(n)) = O(|L| + |S| + |A| \cdot log(n))$.

4.3 Layout and Ordering

After running the planner we obtain a narrative timeline as a list of events (both single-entity and multi-entity events) in chronological order and a list of associated entities. During the layout and ordering stage, we first use the event and entity lists to create an initial arrangement where events are positioned chronologically along the x-axis with entity lines moving from one event to another, and events sharing the same time step are positioned vertically along the y-axis.

Next, we order the events and entities at each time step to minimize the number of line crossings. To do so, we convert the connections between event nodes at each pair of successive time steps into a matrix, creating a list of matrices. If there is a connection between event nodes in a pair of non-successive time steps t_i and t_j, we introduce dummy event connections in the matrices for each pair of successive time steps between t_i and t_j, preserving the connection between the events. Fig. 4(a) shows the layout created using this matrix representation. Events in the planner output from the previous stage are shown as nodes labeled with the names of participating entities. Dummy events are shown as nodes without labels. Links between a pair of nodes represent the participation of entities between the two events.

We next implement a well-known barycentric graph layout algorithm by Sugiyama et al. [32] to rearrange the matrices, starting with the first, flipping rows/columns of the matrices as required. We then perform the same rearrangement of matrices starting from the last. This is repeated until the number of line crossings is minimized, or a set number of iterations is reached. In our implementation we iterate up to ten times. This process creates a layout with minimum line crossings between event nodes as shown in Fig. 4(b).

Following this we remove all dummy nodes from the layout (Fig. 4(c)) before modifying the matrices to include entity connections between event nodes. We repeat the process of rearranging matrices to re-order the entities. This further reduces line crossings between entity lines. Fig. 4(d) shows the layout created at the end of this step. Event are represented as clusters of nodes along the y-axis and colored lines trace the path of entities between nodes.

A final layout is then created using the rearranged matrices as a list of event nodes. Each node in the layout list contains an (x, y) location, entity name, an event tag, an event label, and a list of target nodes, where a target node is a node for an event that occurs after the current node and is connected to the current node. This layout list is used in the final stage to visualize the narrative timelines created from the event nodes.

The time complexity of the algorithm we are using for rearranging matrices is $O(|V| \cdot |E|)$ [12] where V is the number of nodes in the graph being optimized, and E is the number of edges in that graph. Creating the final layout list from the rearranged matrices requires examining every element in each matrix to identify links in the final layout. This also has a time complexity of $O(|V| \cdot |E|)$. The overall time complexity for generating the final layout for a timeline, then, is $O(|V| \cdot |E|) + O(|V| \cdot |E|) = O(|V| \cdot |E|)$

Each iteration of the planning, layout, and ordering stages generates one narrative timeline. The event list in the output of each iteration is compared against those from previous iterations to ensure that we only store unique timelines. This process is repeated until we generate all possible timelines in the narrative.

4.4 Visualization

Once we have iteratively identified all narrative timelines, we pass the layout lists to the final stage in our pipeline to create a visualization using D3.js [2]. We first draw event nodes at the locations specified in the layout list, then draw lines from the nodes to all their target nodes to indicate participation of the entity in both events. Lines for each entity are colored uniquely, with the same color used for their event nodes and event labels.

When creating the visualization, one layout list is identified as the reality timeline and is drawn automatically. The diegetic timelines are drawn on-demand by selecting from a drop-down menu (Fig. 1). At any point the user can choose to draw either a reality timeline, a diegetic timeline, or both. This not only visualizes each timeline independently, but also enables a visual comparison of timelines. Any timeline can be chosen to represent the reality timeline, using the drop-down menu in the interface, automatically categorizing other timelines as diegetic. When visualizing only the reality timeline, we depict entity links using solid lines at 100% opacity. When visualizing only the diegetic timelines, we depict entity links using dashed lines at 100% opacity. When both reality and diegetic timelines are visualized, we depict entity links from the reality timeline using solid lines at 66% opacity, and overlay the entity links from the diegetic timeline using dashed lines at 100% opacity. This allows us to efficiently visualize and compare both timelines simultaneously. More than two timelines could be visualized but only at the expense of additional visual clutter and line crossings.

Figure 5: Visualization of Witcher narrative using Yarn, depicting the favorable outcome as the reality timeline.

Figure 6: Visualization of Witcher narrative using Yarn, depicting the favorable outcome as the reality timeline, and the unfavorable outcome as the diegetic timeline. Note the diegetic timeline represented using dashed lines overlaid on reality timeline in 66% opacity.

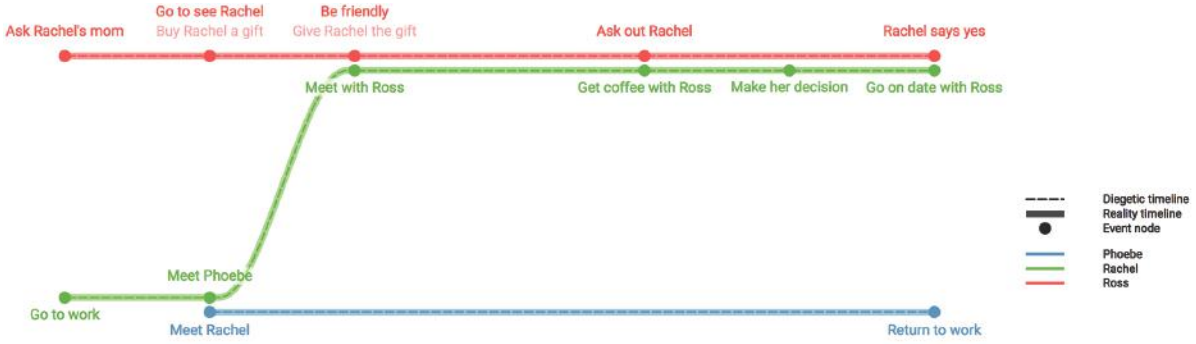


Figure 7: Visualization of Friends narrative using Yarn where Rachel agrees to go out with Ross in both reality as well as diegetic timelines.

Figure 8: Visualization of Friends narrative using Yarn where Rachel agrees to go out with Ross in the diegetic timeline but not the reality timeline.

5 Examples

In this section we show how our system can be used to visualize and compare multiple narrative timelines, first, in a choice-based role playing video game, and second, in a fictional narrative scenario.

5.1 Example: Witcher

For our first example we use a narrative adapted from the video game Witcher 2 to show the capabilities of Yarn.

In our example narrative, the hero, Geralt of Rivia needs to rescue his friend, Triss Merigold, who has mysteriously disappeared. On his journey to find her he must first determine where she was seen last, talk to multiple friends, and resolve a conflict between a troll couple before he can meet her.

There are two choice points in this narrative: first, Margot can either choose to help Geralt or refuse to provide help; second, the troll couple can either accept or reject Geralt's solution to their conflict. Each of these choices affects the outcome of the narrative creating three possible timelines.

The first timeline contains events favorable to Geralt. In this timeline, Margot agrees to help Geralt and the troll couple accept Geralt's solution leading him to find Triss. Fig. 5 shows a visualization of this outcome as the reality timeline for our narrative. We can see that Geralt was able to successfully track Triss's last known location, gather information from friends, resolve a conflict between the trolls, and meet Triss. Note that in this visualization we are showing the default, reality timeline and all links are drawn with 100% opacity. We can also observe all single-entity events in the narrative in this visualization.

In the second timeline, the trolls reject Geralt's solution to their conflict. Fig. 6 shows a visualization of this unfavorable outcome as diegetic timeline overlaid on top of the reality timeline. While most of the events are same in both timelines, the diegetic timeline ends at the event "Speak to She-Troll." The reduced opacity of the reality timeline allows the user to compare common sections of the two timelines, even when one of them is smaller than the other.

In the third timeline, Margot refuses to help Geralt preventing him from progressing further on his quest. This is the shortest timeline in our narrative.

5.2 Example: Friends

For our second example we use a fictional narrative scenario based on the popular sitcom Friends.

In this narrative, Ross's plan is to take Rachel on a date. To do so, he must acquire more information about her, find some way of talking to her, ensure she is positively disposed towards him, and eventually ask her out. He can acquire more information about her in a number of ways including calling her mother, or asking Phoebe about her by either call or text. To ensure she is in a positive mood, he can either give her a gift or say nice things to her, and finally he can either ask her out himself or ask for Phoebe's help. To illustrate causality of events in the narrative timeline, we have set up the operator functions for Rachel such that she is more inclined to say yes to Ross if he talks to her mother rather than requesting Phoebe's help. Additionally, Rachel's probability of accepting Ross's proposal is also dependent on how impressed she is by him when he asks her.

There are four choice points in this narrative: three for Ross, and one for Rachel. The choices made by Ross create 12 alternate timelines, each illustrating a different way in which he can ask Rachel out. Towards the end of each timeline Rachel can decide to either reply yes to Ross, or reply no, creating a total of 24 timelines.

Fig. 7 shows a visualization where Rachel agrees to go out with Ross in both timelines. While the timelines result in the same outcome, and visually look the same, we can inspect the labels on top of the events in Ross's timeline to identify the differences. Lighter labels correspond to the reality timeline, while darker labels correspond to the diegetic timeline. In the reality timeline, Ross decided to give a gift to Rachel after talking to her mom, while in the diegetic timeline he decided to act friendly with her.

Fig. 8 shows another visualization for the same narrative. In this example we can see the effect of causal conditions set in the operator functions for Rachel. In the reality timeline, Rachel decided not to go out with Ross because he asked Phoebe for help by messaging her. On the other hand, she agreed to go out with him in the diegetic timeline because in this case Ross decided to talk to her mother.

Table 1: Average execution time for the planning, and layout and ordering stages for timelines of various lengths.

	Timelines			Average Time (s)	
Narrative	Events	Nodes	Number	Planning	Layout
	4	6	1	1.8169	0.0048
Witcher	7	12	1	6.4473	0.0051
	8	14	1	6.5692	0.0059
Friends	9	13	18	1.3614	0.0055
	10	15	6	1.7737	0.0062

5.3 Runtime Performance

Table 1 shows time required to execute Yarn on each of our example narratives. Since the timeline generation phase (Fig. 3) is computationally the most expensive phase in Yarn's pipeline, we only record the runtime for this phase. We ran our tests on a Macbook Pro with Intel Core i7-4850HQ CPU (2.3 GHz, 8 logical cores) and 16GB of RAM. Each example narrative was constructed and visualized five times, and the execution times were averaged to produce the results shown.

For each narrative, we calculate the average execution time for generating a timeline of a specific length. While each of the three Witcher timelines are of different lengths, 18 timelines in the Friends narrative consist of 13 entity nodes, and 6 timelines consist of 15 nodes. From the results, we can observe that as the number of nodes in each timeline increases, the time required to compute the layout also increases, as expected.

From Table 1 we can also see that while the time required for planning increases with the number of nodes in the timeline, it is influenced heavily by the narrative structure. If an action in an HTN requires the output of an action in another HTN as a pre-requisite, the planning for the first HTN is suspended until the pre-requisite is satisfied. Although this preserves the chronological order of events in the timeline, it increases the runtime because fewer plans can be generated concurrently. The Witcher timelines, individually, take longer to execute because most actions in Geralt's HTN serve as pre-requisites for other entities. Also, they have slightly more fabula elements in the narrative state map than the Friends narrative, making it slightly more expensive to execute each action.

Verdú and Pajuelo have shown that in many cases spawning fewer worker threads results in a similar or slightly better runtime performance [38]. The higher planning time for the Witcher narrative can be also attributed to the higher number of concurrent planners (six) than the Friends narrative (three).

Overall, execution takes $\sim$1.4 seconds per timeline in the Friends narrative and $\sim$5 seconds per timeline in the Witcher narrative. Based on times reported by Liu et al. [19], our system runs significantly faster than the original work of Tanahashi and Ma ($\sim$150 seconds per timeline), and somewhat slower than Liu et al. ($\sim$0.16 seconds per timeline).

6 Conclusions and Future Work

We have presented Yarn, a new system for automatic narrative construction and visualization. Through iterative application of HTN planning we generate all possible timelines in a narrative. These timelines are visualized using a storyline-like technique to make it easier for a user to summarize the events in the narrative. We also support visual comparison of pairs of narrative timelines.

While we have illustrated the usefulness of Yarn using example narratives from a choice-based video game and a fictional situation in a sitcom, Yarn can visualize other types of temporal relationships with one or more timelines, such as simulation results, news stories, historical events, and so on. As part of our future work we would like to demonstrate the application of Yarn for visualizing news stories in a document corpus, non-linear narratives with flashbacks and flash forwards, and evaluate the performance of Yarn when visualizing large narratives such as those with a few dozen entities and hundreds of timelines.

Currently Yarn generates all possible timelines in the narrative before it visualizes them. This could limit its scalability in scenarios with hundreds of timelines. We would like to improve Yarn's pipeline to make the visualization of a timeline available as soon as its layout is ready. We would also like to investigate the scalability of concurrent planning in narratives with large numbers of character entities. Finally, we would like to extend Yarn to support visualization of streaming/evolving narratives.

REFERENCES

[1] N. Avradinis, R. Aylett, and T. Panayiotopoulos. Using motivation-driven continuous planning to control the behaviour of virtual agents. In *Virtual Storytelling. Using Virtual RealityTechnologies for Storytelling*, pp. 159–162. Springer, 2003.

[2] M. Bostock, O. Ogievetsky, and J. Heer. D^3 data driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[3] E. Branigan. *Narrative Comprehension and Film*. Sightlines. Routledge, 2013.

[4] M. Burch, C. Vehlow, F. Beck, S. Diehl, and D. Weiskopf. Parallel edge splatting for scalable dynamic graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2344–2353, 2011.

[5] M. Cavazza, F. Charles, and S. J. Mead. Interacting with virtual characters in interactive storytelling. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*, pp. 318–325. ACM, 2002.

[6] M. Cavazza, F. Charles, and S. J. Mead. Planning characters' behaviour in interactive storytelling. *The Journal of Visualization and Computer Animation*, 13(2):121–131, 2002.

[7] S. B. Chatman. *Story and Discourse: Narrative Structure in Fiction and Film*. Cornell University Press, 1980.

[8] Y.-G. Cheong and R. M. Young. Suspenser: A story generation system for suspense. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(1):39–52, 2015.

[9] D. K. Elson and K. R. McKeown. A platform for symbolically encoding human narratives. In *Proceedings of the AAAI Fall Symposium on Intelligent Narrative Technologies*, 2007.

[10] K. Erol, J. Hendler, D. S. Nau, and R. Tsuneto. A critical look at critics in htn planning. In *Proceedings, IJCAI-95*. Citeseer, 1995.

[11] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189–208, 1972.

[12] A. Frick. *Upper Bounds on the Number of Hidden Nodes in Sugiyama's Algorithm*, pp. 169–183. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997. doi: 10.1007/3-540-62495-3_46

[13] Y. Frishman and A. Tal. Online dynamic graph drawing. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):727–740, 2008.

[14] D. Herman, J. Phelan, P. J. Rabinowitz, B. Richardson, and R. Warhol. *Narrative Theory: Core Concepts and Critical Debates*. Ohio State University Press, 2012.

[15] J. Hullman and N. Diakopoulos. Visualization rhetoric: Framing effects in narrative visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2231–2240, 2011.

[16] M. Jensen. Visualizing complex semantic timelines. *Derived from the World Wide Web: http://newsblip.com/tr*, 2003.

[17] S. Kambhampati and J. A. Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55(2):193–258, 1992.

[18] B. Lee, N. H. Riche, P. Isenberg, and S. Carpendale. More than telling a story: Transforming data into visually shared stories. *IEEE Computer Graphics and Applications*, 35(5):84–90, 2015.

[19] S. Liu, Y. Wu, E. Wei, M. Liu, and Y. Liu. Storyflow: Tracking the evolution of stories. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2436–2445, Dec. 2013. doi: 10.1109/TVCG.2013.196

[20] B. Magerko, J. Laird, M. Assanie, A. Kerfoot, and D. Stokes. AI characters and directors for interactive computer games. *Ann Arbor*, 1001(48):109–2110, 2004.

[21] J. Meehan. Tale-Spin, an interactive program that writes stories. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pp. 91–98, 1977.

[22] R. Munroe. Xkcd #657: Movie narrative charts, 2009.

[23] M. Ogawa and K.-L. Ma. Software evolution storylines. In *Proceedings of the 5th International Symposium on Software Visualization*, SOFTVIS '10, pp. 35–42. ACM, New York, NY, USA, 2010. doi: 10.1145/1879211.1879219

[24] V. Ogievetsky. Plotweaver xkcd/657 creation tool, 2009.

[25] R. Pérez y Pérez. *MEXICA: A Computer Model of Creativity in Writing*. PhD thesis, The University of Sussex, Falmer, UK, 1999.

[26] V. Propp. *Morphology of the Folktale*. University of Texas Press, 1968.

[27] M. Riedl. *Narrative Planning: Balancing Plot and Character*. PhD thesis, North Carolina State University, Raleigh, NC, 2004.

[28] M. O. Riedl and R. M. Young. An intent-driven planner for multi-agent story generation. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pp. 186–193. IEEE Computer Society, 2004.

[29] S. Rimmon-Kenan. *Narrative Diction: Contemporary Poetics*. Routledge, 2003.

[30] G. Robertson, R. Fernandez, D. Fisher, B. Lee, and J. Stasko. Effectiveness of animation in trend visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6), 2008.

[31] E. Segel and J. Heer. Narrative visualization: Telling stories with data. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1139–1148, 2010.

[32] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, Feb 1981. doi: 10.1109/TSMC.1981.4308636

[33] Y. Tanahashi, C.-H. Hsueh, and K.-L. Ma. An efficient framework for generating storyline visualizations from streaming data. *IEEE Transactions on Visualization and Computer Graphics*, 21(6):730–742, 2015.

[34] 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. doi: 10.1109/TVCG.2012.212

[35] R. Thawonmas, K. Tanaka, and H. Hassaku. Extended hierarchical task network planning for interactive comedy. In *Intelligent Agents and Multi-Agent Systems*, pp. 205–213. Springer, 2003.

[36] M. Theune, E. Faas, A. Nijholt, and D. Heylen. The virtual storyteller: Story creation by intelligent agents. In *Proceedings of the Technologies for Interactive Digital Storytelling and Entertainment*, pp. 204–215. Springer, Berlin, 2003.

[37] S. R. Turner. *Minstrel: A Computer Model of Creativity and Storytelling*. PhD thesis, University of California at Los Angeles, Los Angeles, CA, 1993.

[38] J. Verdú and A. Pajuelo. Performance scalability analysis of javascript applications with web workers. *IEEE Computer Architecture Letters*, 15(2):105–108, 2016.

[39] J. Waser, R. Fuchs, H. Ribicic, B. Schindler, G. Bloschl, and E. Groller. World lines. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1458–1467, 2010.