Offline Drawing of Dynamic Trees: Algorithmics and Document Integration

Malte Skambath[1(✉)] and Till Tantau[2]

[1] Department of Computer Science, Kiel University, Kiel, Germany
malte.skambath@email.uni-kiel.de
[2] Institute of Theoretical Computer Science, Universität Zu Lübeck,
Lübeck, Germany
tantau@tcs.uni-luebeck.de

Abstract. While the algorithmic drawing of static trees is well-understood and well-supported by software tools, creating animations depicting how a tree changes over time is currently difficult: software support, if available at all, is not integrated into a document production workflow and algorithmic approaches only rarely take temporal information into consideration. During the production of a presentation or a paper, most users will visualize how, say, a search tree evolves over time by manually drawing a sequence of trees. We present an extension of the popular TEX typesetting system that allows users to specify dynamic trees inside their documents, together with a new algorithm for drawing them. Running TEX on the documents then results in documents in the SVG format with visually pleasing embedded animations. Our algorithm produces animations that satisfy a set of natural aesthetic criteria when possible. On the negative side, we show that one cannot always satisfy all criteria simultaneously and that minimizing their violations is NP-complete.

1 Introduction

Trees are undoubtedly among the most extensively studied graph structures in the field of graph drawing; algorithms for drawing trees date back to the origins of the field [26,40]. However, the extensive, ongoing research on how trees can be drawn efficiently, succinctly, and pleasingly focuses on either drawing a single, "static" tree or on interactive drawings of "dynamic" trees [11,12,27], which are trees that change over time. In contrast, the problem of drawing dynamic trees *non*interactively in an *offline* fashion has received less attention.

It is this problem that lies at the heart of our paper.

Consider how an author could explain, in a paper or in a presentation, how a tree-based data structure such as a search tree works. In order to explain the dynamic behavior, our author might wish to show how the data structure evolves for a sequence of update operations. A typical drawing of the evolving

---

Animations in this document will only be rendered in the SVG version [32], see Sect. 2.3 for a discussion of the reasons.
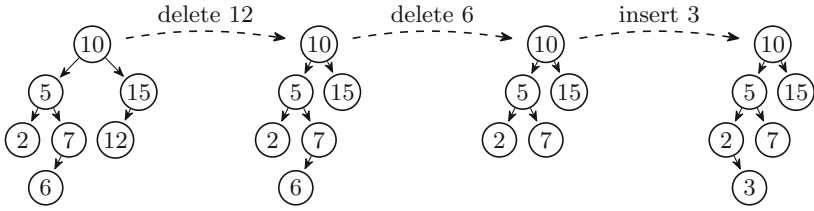
Fig. 1. A "manually" created drawing of a dynamic tree: each tree in the sequence has been drawn using the Reingold–Tilford [29] algorithm.
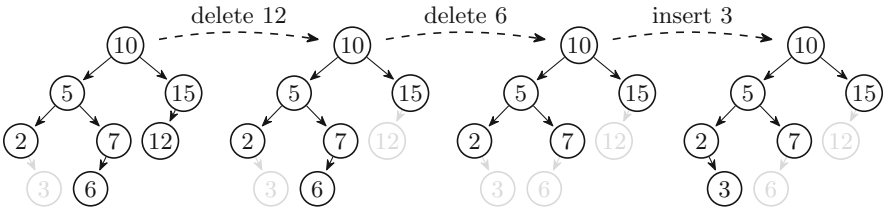
Fig. 2. The dynamic tree from Fig. 1, redrawn by drawing a "supergraph" (the union of all trees in the sequence) and then using the positions of nodes in this supergraph for the individual drawings.

sequence might look as in Fig. 1, which has been created "manually" by running the Reingold–Tilford algorithm [29] on each tree in the sequence independently. While the result is satisfactory, there are (at least) two shortcomings:

First Shortcoming: Flawed Layout. In the first step, the layout of the root's children changes (their horizontal distance decreases) even though there is no structural change at the root. While in the present graph the effect is small, one can construct examples where a single node removal causes a change in distances on all levels, obscuring where the actual structural change occurred. Since the whole sequence of trees (the whole "dynamic tree") is given by the author, the problem can be addressed by not running the Reingold–Tilford algorithm on each tree individually, but by running it on the "supergraph" resulting from uniting all trees in the sequence, resulting in the visualization in Fig. 2.

Unfortunately, this simple supergraph approach introduces new problems: First, the nodes "2" and "7" are unnecessarily far apart – the nodes "3" and "6" could use the same space since they are never both members of the same tree. Second, it is easy to construct sequences of trees whose union is not a tree itself.

We address these problems in Sect. 3, where we present a new algorithm for computing layouts of dynamic trees that addresses the above problems. For dynamic trees whose supergraphs are trees or at least acyclic, the algorithm finds an optimal layout (with respect to natural aesthetic criteria) of the dynamic tree in linear time. For cyclic supergraphs, which are also important in practice since they arise for instance from the rotations necessary to balance search trees in data structures such as AVL trees [1], we show that one has to break the cycles in order to layout the graph according to the criteria we develop. While we show

that it is NP-complete to find a minimal set of break points, a simple greedy heuristic for finding breakpoints turns out to produce visually pleasing results.

Second Shortcoming: Presentation as a Sequence of Snapshots. In order to depict the evolving nature of her dynamic tree, our author depicted different "snapshots" of the tree at different times and arranged these snapshots in a sequence. While the temporal dimension *needs* to be turned into something else when our medium of communication is printed paper, for documents presented using appropriate electronic devices we can visualize dynamic trees using *animations.* Such an animation needs much less space on a page and, perhaps more importantly, our visual system is *much* better at spotting movement than at identifying structural changes between adjacent objects.

In Sect. 2 we present a system for creating animations on-the-fly during a run of the TEX program on a text document: First, we have augmented the popular Ti*k*Z graphic package [37] (a macro package for TEX for creating graphics) by commands that compute and embed animations in the output files. Due to the way the system works, these commands have almost no overhead regarding compilation speed or resulting file size. Second, we have implemented a prototype of our algorithm from Sect. 3 for drawing dynamic trees that uses these animation commands. In result, when an author specifies the above dynamic graph appropriately in a TEX document and then runs TEX on it to convert it, the resulting file will contain the normal text and graphics as well as an embedded animation of the dynamic tree. When the document is viewed on electronic devices with a modern browser, the animation runs right inside the document.

Related Work. Approaches to drawing *static* trees date back to the early 1970s, namely to the work of Knuth, Wetherell and Shannon, and Sweet [26,35,40]. A standard algorithm still in use today is due to Reingold and Tilford [29], see also [38]. They suggested that symmetric tree structures should be drawn symmetrically and provided an algorithm that supports this objective well and runs in linear time. Instead of visualizing trees as node-link diagrams, one can also use tree maps [25], three dimensional cone trees [30], or sunburst visualizations [33].

Approaches to drawing general *dynamic* graphs are more recent. The sequence-of-snapshot visualizations sketched before as well as animations are standard ways of visualizing them [19]. One can also generally treat time as another spacial dimension, which turns nodes into tubes through space [23]. There are many further techniques that are not restricted to node-link diagrams [8,9,22,28]; for an extensive overview of the whole state of the art including a taxonomy of different visualization techniques see Beck et al. [5], or [21] for a more tree-specific overview. Diehl, Görg and Kerren [14,15] introduced a general concept, called *foresighted layout*, for drawing dynamic graphs offline. They propose to collapse nodes in the supergraph that never exist at the same time and to then draw the supergraph. While this approach produces poor results for trees, the results are better for hierarchical graphs [20].

Approaches tailored specifically to drawing dynamic *trees* are currently almost always *online* approaches. The algorithms, which expect a sequence of

update operation as input [12,27], are integrated into interactive software and create or adjust the layout for each change. An early algorithm designed for dynamic trees was developed by Moen [27]. Later Cohen et al. [11,12] presented algorithms for different families of graphs the includes trees.

Concerning the integration of tree drawing algorithms into text processing software, first implementations for the typesetting system TEX date back to Eppstein [18] and Brüggemann and Wood [6]. A more recent implementation of the Reingold–Tilford algorithm by the second author is now part of the graph drawing engine in Ti*k*Z [36].

Organisation of This Paper. This paper is structured into two parts: In the first part, Sect. 2, we present the system we have developed for generating animations of dynamic graphs that are embedded into documents. Our core argument is that the system's seamless integration into a widely used system such as TEX is crucial for its applicability in practice. In the second part, Sect. 3, partly as a case study, partly as a study of independent interest, we investigate how a dynamic tree can be drawn using animations. We derive aesthetic criteria that animations and even image sequences of dynamic trees *should* meet and present an algorithm that *does* meet them. Full proofs and pseudo-code can be found in the appendix of the full version, which also contains a gallery of dynamic trees drawn using our prototype.

2   Dynamic Trees in Documents

The problem for which we wish to develop a practical solution in the rest of this paper is the following: Visualize one or more dynamic trees inside a document created by an author from some manuscript. To make the terminology precise, by *dynamic graph* we refer to a sequence $(G_1, \ldots, G_n)$, where each $G_i = (V_i, E_i, \phi_i)$ is a directed, annotated graph with vertex set V_i, edge set E_i, and an annotation function $\phi_i \colon V_i \cup E_i \to A$ that assigns additional information to each node and edge from some set A of annotations like ordering or size information. A *dynamic tree* is a dynamic graph where each T_i is a tree with the edges pointing away from the root. A *manuscript* is a plain text written by an *author* that can be transformed by a program into an (output) *document,* a typically multi-page text document with embedded graphics or embedded animations. Note that the problem is an *offline* problem since the manuscript contains a full description of the dynamic graph and algorithms have full access to it. In rest of this section we explain how the practical obstacles arising from the problem are solved by the system we have developed, in Sect. 3 we investigate algorithmic questions.

In the introduction we saw an example of how a dynamic tree can be visualized using a series of "snapshots" shown in a row. While this way of depicting a dynamic tree is a sensible, traditional way of solving the problem (drawings on printed paper "cannot change over time"), documents are now commonly also read on electronic devices that are capable of displaying changing content and, in particular, animations. We claim that using an animation instead of a

sequence of snapshots has two major advantages: First, sequences of snapshots need a lot of space on a page even for medium-sized examples. We did a cursory survey of standard textbooks on computer science and found that typically only three to four snapshots are shown and that the individual trees are often rather small. For an animation, the length of the sequence is only limited by the (presumed) attention span of the reader and not by page size. Second, our visual system is *much* better at spotting movement than at identifying structural changes between adjacent objects. When operations on trees such as adding or deleting a leaf or moving whole subtrees are visualized using movements, readers can identify and focus on these operations on a subconscious level.

Given the advantages offered by animations, it is surprisingly difficult to integrate animations into documents. Of course, there is a lot of specialized software for creating animations and graphics output formats like PDF or SVG allow the inclusion of movie files in documents. However, this requires authors to use – apart from their main text processor like TEX or Word – one or more programs for generating animations and they then have to somehow "link" the (often very large) outputs of these different programs together. The resulting workflows are typically so complicated that authors rarely employ them. Even when they are willing to use and integrate multiple tools into their workflow, authors face the problem that using different tools makes it next to impossible to keep a visually consistent appearance of the document [36]. Very few, if any, animation software will be able to render for instance TEX formulas inside to-be-animated nodes correctly and take the sizes of these formulas into account.

We have developed a system that addresses the above problems; more precisely, we have augmented an existing system that is in wide-spread use – TEX – by facilities for specifying dynamic trees, for computing layouts for them, and for generating animations that are embedded into the output files. Our extensions are build on top of Ti*k*Z's graph drawing engine [36], which has been part of standard TEX distributions since 2014.

Authors first specify the dynamic trees they wish to draw inside TEX manuscripts using a special syntax, which we describe in Sect. 2.1 (conceptually, this is similar to specifying for instance formulas inside the TEX manuscripts). Next, authors apply a graph drawing algorithm to the specified dynamic graph by adding an appropriate option to the specification and then running the TEX program as explained in Sect. 2.2. Lastly, in Sect. 2.3, we discuss which output formats are supported by our system, how the output can be viewed on electronic devices, and how a fallback for printed paper can be generated.

2.1   The Input: Specifying Dynamic Trees

In order to make dynamic trees accessible to graph drawing algorithms, we first have to specify them. For dynamic graphs and, in particular, for dynamic trees, there are basically two different methods available to us: We can specify each graph or tree in the dynamic graph sequence explicitly. Alternatively, we can specify a sequence of update operations that transform one graph into the next such as, for the dynamic trees of search trees, the sequence of insert and delete

operations that give rise to the individual trees. Besides being easy and natural to use, the second method also provides algorithms with rich semantic information concerning the change from one graph to the next in the sequence.

Despite the fact that the second method is more natural in several contexts and more semantically rich, for our prototype we use the first method: Authors specify dynamic graphs by explicitly specifying the sequence of graphs that make up the dynamic graph. We have two reasons for this choice: First, specifying the sequence of graphs explicitly imposes the least restrictions on what kind of dynamic graphs can be drawn, in principle. In contrast, the set of update operations necessary to describe the changes occurring just for the standard data structures balanced search trees, heaps, and union–find trees is large and hard to standardize. For instance, should the root rotation occurring in AVL trees be considered a standard update operation or not? Second, it easy to convert a sequence of update operations into a sequence of graphs, while the reverse direction is harder and, sometimes, not possible. Our system can easily be extended to accept different sequences of update operations as input and convert them on-the-fly into a sequence of graphs that is then processed further.

There are different possible formats for specifying individual graphs and, in particular, trees of graph sequences, including GRAPHML, an XML-based markup language; the DOT format, used by GRAPHVIZ [17]; the GML format, used by the Open Graph Drawing Framework [10]; or the format of the \graph command of Ti*k*Z [37], which is similar to the DOT format. As argued in [36], it is not purely a matter of taste, which format is used; rather, good formats make it easy for humans to notate all information about a graph that is available to them. For instance, for static graphs the *order* in which vertices are specified is almost never random, but reflects information about them that the author had and that algorithms should take into account.

Since the algorithm and system we have implemented are build on top of the graph drawing engine of Ti*k*Z [36], we can use all of the different syntax flavors offered by this system, but authors will typically use the \graph command. Each graph in the sequence of graphs is surrounded by curly braces and, following the opening brace, we say [when=*i*] to indicate that we now specify the *i*th graph in the sequence. The graph is then specified by listing the edges, please see [36] and [37] for details on the syntax and its use in Ti*k*Z. The result is a specification of the dynamic graph such as the following for the example graph from Figs. 1 and 2:

```
\tikz \graph { {[when=1] 10->{ 5->{ 2,          7->6 },  15->12 } },
               {[when=2] 10->{ 5->{ 2,          7->6 },  15 } },
               {[when=3] 10->{ 5->{ 2,          7 },     15 } },
               {[when=4] 10->{ 5->{ 2->{ , 3 }, 7 },     15 } } };
```

2.2   Document Processing and Algorithm Invocation

Once a dynamic graph has been specified as part of a larger TEX document, we need to process it. This involves both running a dynamic graph drawing

algorithm to determine the positions of the nodes and the routing of the edges as well as producing commands that create the desired animation.

The framework provided by the graph drawing engine [36] of TikZ is well-suited for the first task. All the author has to do is to load an appropriate graph drawing library and then use a special key with the \tikz command:

```
\tikz [animated binary tree layout]
  \graph { {[when=1] 10->{ ... } };
          {[when=2] 10->{ ... } },
          {[when=3] 10->{ ... } },
          {[when=4] 10->{ ... } } };
```
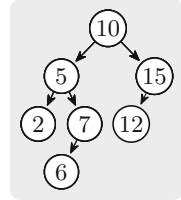
The key `animated binary tree layout` causes the graph drawing engine to process the dynamic graph. It will parse the dynamic graph, convert it to an object-oriented model, and pass it to an algorithm from the `evolving` library, which is written in the Lua programming language [24].[1] The framework also handles the later rendering of the nodes and edges and their correct scaling and embedding into the output document. Thus, the algorithm's implementation only needs to address the problem of computing node positions from an object-oriented model of the dynamic graph. The implementation need not (indeed, cannot) produce or process graphical output and primitives.

Once the algorithm has computed the positions for nodes and edges of the graphs in the sequence, actual animations need to be generated. For this, TikZ itself was extended by a new animation subsystem, which can be used independently of the graph drawing engine and allows users to specify and embed arbitrary animations in their documents. The animation subsystem adds *animation annotations* to the output file, which are statements like "move this graphics group by 1 cm to the right within 2 s" or "change the opacity of this node from opaque to transparent within 200 ms." More formally, they are XML elements in the Synchronized Multimedia Integration Language [7]. For the animation of dynamic graphs, the graph drawing engine can now map the computed positions of the nodes at different times to TikZ commands that add appropriate movement and opacity-change annotations to the output.

2.3   The Output: Scalable Vector Graphics

The annotation-based way of producing animations has two important consequences: Firstly, adding the annotations to the output does not have a noticeable effect on the speed of compilation (computing the necessary XML statements is quite easy) nor on the file size (annotations are small). However, secondly, the job of rendering the graph animations with, say, 30 frames per second does not

---

[1] When the algorithm is also implemented in the Lua language, it can be used directly by TeX without special configurations or runtime linking, but it can also be implemented in C or C++ at the cost of a more complicated deployment.

lie with TEX, but with the viewer application and we need both a format and viewer applications that support this.

Currently, there is only one graphics format that supports these annotation-based animations: The *Scalable Vector Graphics* (SVG) format [13], which is a general purpose graphics language that is in wide-spread use. All modern browsers support it, including the parsing and rendering of SVG animations. The DVISVGM program, which is part of standard TEX distributions, transforms arbitrary TEX documents into SVG files that, when viewed in a browser, are visually indistinguishable from PDF files produced by TEX – except, of course, for the animations of the dynamic graphs.

While we argued that animations are a superior way of visualizing dynamic graphs, there are situations where they are not feasible: First, documents *are* still often printed on paper. Second, the popular PDF format does not support annotation-based animations and, thus, is not able to display TikZ's animations. Third, it is sometimes desirable or necessary to display "stills" or "snapshots" of animations at interesting time steps alongside the animation. In these situations, authors can say `make snapshot of=`t to replace the animation by a static picture of what the animated graphic would look like at time t. Since the computation of the snapshot graphic is done by TEX and since no animation code is inserted into the output, this method works with arbitrary output formats, including PDF.

3   Algorithmic Aspects of Drawing Dynamic Trees

Given a dynamic tree $T = (T_1, \ldots, T_k)$ consisting of a sequence of trees $T_i = (V_i, E_i, \phi_i)$, we saw in the introduction that neither drawing each tree independently and then "morphing" the subsequent drawings to create an animation nor laying out just the supergraph $\text{super}(T) = (\bigcup_i V_i, \bigcup_i E_i)$ and then animating just the opacity of the nodes and edges will lead to satisfactory drawings of dynamic trees. Our aim is to devise a new algorithm that addresses the shortcomings of these approaches and that meets a number of sensible *aesthetic criteria* that we formulate in Sect. 3.1. The algorithm, presented in Sect. 3.2, has been implemented as a prototype [31] and we have used it to create the animations of dynamic trees in the present paper. While the prototype implementation does not even run in linear time (as would be possible by Theorem 3.2), it only needs fractions of a second for the example graphs from this paper.

3.1   Aesthetic Criteria for Drawing Dynamic Trees

Already in 1979, Wetherell and Shannon [40] explicitly defined aesthetic criteria for the layout of trees. Two years later Reingold and Tilford [29] refined these *static criteria* towards more symmetric drawings in which isomorphic subtrees must have the same layout. While the criteria were originally formulated for binary trees only, one can allow any number of children when there is an ordering on the children of each node.

Criterion (Ranking). *The vertical position of a node equals its topological distance from the root.*
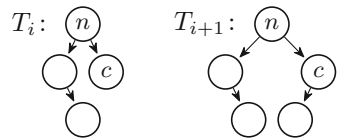
Criterion (Ordering). *The horizontal positions of a node's children respect their topological order in the tree.*

Criterion (Centering). *Nodes are horizontally centered between their leftmost and rightmost child if there are at least two children.*

Criterion (Symmetry). *All topologically order-isomorphic subtrees are drawn identically. Topologically mirrored subtrees are drawn horizontally mirrored.*

As numerous applications show, these rather sensible criteria lead to aesthetically pleasing drawings of static trees. We extend these criteria to the dynamic case. Ideally, we would like to keep all of the above criteria, but will see in a moment that this is not always possible.

Our first dynamic criterion forbids the unnecessary movement of nodes in drawings like the one shown on the right, which shows the same problem as the example in the introduction did: The horizontal offset between n and c changes from T_i to T_{i+1} even though there is no structural change at n. (Note that when a node disappears in the step from T_i to T_{i+1} and then reappears in T_{i+2}, the stability criterion does no require it to appear at the same position as before.)

Criterion (Stability). *The horizontal offset between a node n and a child c may not change between the layouts of trees T_i and T_{i+1} if c does not change its position in the ordering of the children of n.*

While the stability criterion forbids relative movements of connected nodes, it allows whole subtrees to move without changing their inner layout. This emphasizes the important parts of changes since multiple objects moving with the same speed are percieved as one connected group [4,39]. The criterion reduces movements and draws common structures identically, thereby reducing errors in understanding [2] and making it easier for viewers to correctly recognize the changes in the tree sequence [3].

While all of the above criteria are reasonable, unfortunately, there is no way of meeting all of them simultaneously, see the appendix for the proof:

Lemma 3.1. *No drawing of the dynamic tree $T = (T_1, T_2)$ from Fig. 3 meets all of the criteria Ranking, Ordering, Centering, Symmetry, and Stability.*

In view of the lemma, we will need to weaken one or more of our criteria, while still trying to meet them at least in "less problematic" cases than the dynamic tree from Fig. 3. Furthermore, even when the criteria *can* be met, this may not always be desirable.
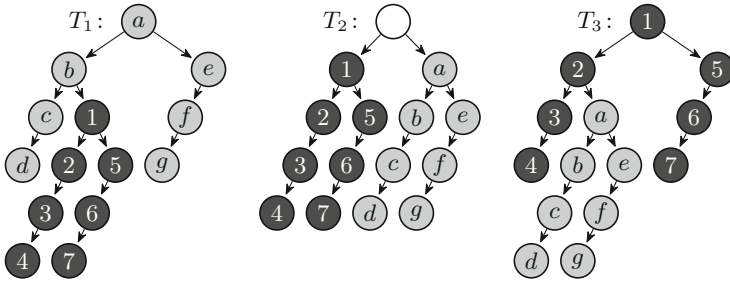
Fig. 3. A "problematic" dynamic tree. Already the dynamic tree $T = (T_1, T_2)$ cannot be drawn while meeting all of the criteria Ranking, Ordering, Centering, Symmetry, and Stability, as shown in Lemma 3.1. The whole dynamic tree $T = (T_1, T_2, T_3)$ cannot even be drawn when the Symmetry Criterion is replaced by the Weak Symmetry Criterion, see Lemma 3.3.

Consider the right example, which seems like a "reasonable" drawing of a dynamic tree. The Stability Criterion enforces the large distance between b and c already in T_1, but the Symmetry Criterion would now actually enforce the same distance between 2 and 3, which seems undesirable here. As a replacement of the Symmetry Criterion we propose a *Weak Symmetry Criterion* that our algorithm will be able to meet in many important cases, including the troublesome example from Lemma 3.1. Nevertheless, there are *still* dynamic trees that cannot be drawn in this way, see Lemma 3.3, which also turn out to be the algorithmically difficult cases.
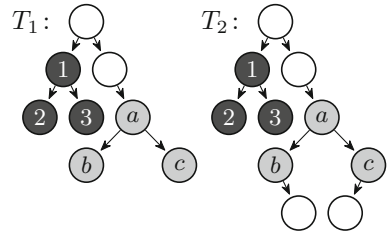
Criterion (Weak Symmetry). *Let n and n' be nodes such that for all $i \in \{1, \ldots, n\}$ the subtrees rooted at n and at n' in T_i are order-isomorphic (or all mirrored). Then in all drawings of the T_i the subtrees rooted at n and n' must all be drawn identically (or all mirrored).*

3.2   An Algorithm for Drawing Arbitrary Dynamic Trees

Our starting point for an algorithm that meets the aesthetic criteria just formulated is the classical Reingold–Tilford algorithm [29]. It will be useful to review this algorithm briefly, formulated in a "bottom-up" fashion: While there is a node that has not yet been processed, pick a node n whose children $c^1, \ldots, c^m$ have all already been processed (this is immediately the case for all leafs, where $m = 0$). For each child c^r a layout $L(c^r)$ will have been computed for the subtree $T(c^r)$ of T rooted at c^r. The algorithm now shifts the $L(c^r)$ vertically so that all c^r lie on the same horizontal line (Ranking Criterion), then shifts them horizontally so that the c^1 comes first, followed by c^2, and so on (Ordering Criterion),

such that no overlap of the $L(c^r)$ occurs. Finally, n is centered above its children (Centering Criterion). The Symmetry Criterion is satisfied automatically by this algorithm since the same shifts occur for symmetric subtrees. Using appropriate data structures, the algorithm can be implemented in linear time.

Our Algorithm A.1, see the appendix for pseudo-code, uses the same basic idea as the Reingold–Tilford algorithm, but introduces two new ideas.

Idea 1: Treat Nodes as Three-Dimensional Objects. In our algorithm, we treat nodes and subtrees as "three dimensional" objects with time as the third dimension. Given a dynamic tree $T = (T_1, \ldots, T_k)$, the algorithm does not process the T_i one at a time (as online algorithms have to do), but instead considers for each node n of the supergraph super(T) the sequence

tighten

$(T_1(n), \ldots, T_k(n))$ of trees rooted at n in the different T_i and computes a whole sequence of layouts $(L_1(n), \ldots, L_k(n))$ for these trees: The core operation of the Reingold–Tilford algorithm, the shifting of a layout $L(c^r)$ until it almost touches the previous layout $L(c^{r-1})$, is replaced by a shifting of the whole sequence $(L_1(c_1^r), \ldots, L_k(c_k^r))$, where c_j^i denotes the ith child of n in T_j, until at least one layout $L_j(c_j^r)$ (one of the gray layouts in the example) almost touches its sibling's layout $L_j(c_j^{r-1})$ (one of the dark layouts).
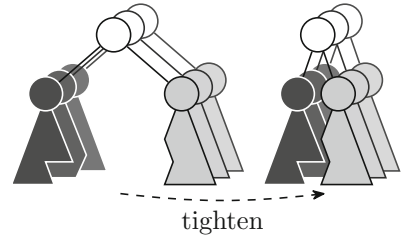
Idea 2: Processing the Supergraph Using a Topological Ordering. For static trees, there is a clear order in which the nodes should be processed by the Reingold–Tilford algorithm: from the leafs upwards. For a dynamic tree, this order is no longer clear – just consider the example from Fig. 3: Should we first process node 1 or node a? Our algorithm address this ordering problem as follows: We compute the supergraph super(T) and then check whether it is acyclic. If so, it computes a topological ordering of super(T) and then processes the nodes in this order. Observe that this guarantees that whenever a node is processed, complete layouts for its children will already have been computed.

Theorem 3.2. *Let T be a dynamic tree whose supergraph is acyclic. Then Algorithm A.1 draws T in linear time such that all of the criteria Ranking, Ordering, Centering, Weak Symmetry, and Stability are met.*

Theorem 3.2 settles the problem of drawing dynamic trees with acyclic supergraphs nicely. In contrast, for a cyclic supergraph, things get *much* harder:

Lemma 3.3. *No drawing of $T = (T_1, T_2, T_3)$ from Fig. 3 meets all of the criteria Ranking, Ordering, Centering, Weak Symmetry, and Stability.*

The lemma tempts us to just "give up" on cyclic supergraphs. However, these arise naturally in prune-and-regraft operations and from rotations in search trees – which are operations that we would like to visualize. We could also just completely ignore the temporal criteria and return to drawing each tree individually in such cases – but we might be able to draw everything nicely except for a single "small" cycle "somewhere" in the supergraph.

We propose to deal with the cycle problem by "cutting" the cycles with as few "temporal cuts" as possible. These are defined as follows: Let $G = (G_1, \ldots, G_k)$ be a dynamic graph and let n be a node of the supergraph super(G) and let $i \in \{1, \ldots, k-1\}$. The *temporal cut* of G at n and i is a new dynamic graph G' that is identical to G, except that for all $j \in \{i+1, \ldots, k\}$ in which G_j contains the node n, this node is replaced by the same new node n' (and all edges to or from n are replaced by edges to or from n').

Temporal cuts can be used to remove cycles from the supergraph of a dynamic graph, which allows us to then run our Algorithm A.1 on the resulting graph; indeed, simply "cutting everything at all times" turns every supergraph into a (clearly acyclic) collection of non-adjacent edges and isolated nodes. However, we wish to minimize the number of temporal cuts since, when we visualize G' using an animation, the different locations that may be assigned to n and n' will result in a *movement* of the node n to the new position of n'.
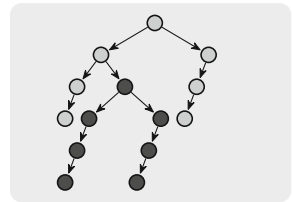
By the above discussion, we would like to find an algorithm that solves the following problem TEMPORAL-CUT-MINIMIZATION: Given a dynamic tree T, find a minimal number of temporal cuts, such that the resulting dynamic tree T' has an acyclic supergraph. Unfortunately, this problem turns out to be difficult:

Theorem 3.4. *The decision version of* TEMPORAL-CUT-MINIMIZATION *is NP-complete.*

In light of the above theorem, we have developed and implemented a simple greedy heuristic, Algorithm A.2, for finding temporal cuts that make the supergraph acyclic, which our prototype runs prior to invoking Algorithm A.1: Given a dynamic tree, the heuristic simply adds the trees T_i and their edges incrementally to the supergraph. However, whenever adding an edge $e = (v, w)$ of T_i to the supergraph creates a cycle, we cut w at $i - 1$.

4   Conclusion and Outlook

We have presented a system for offline drawings of dynamic trees using animations that are embedded in (text) documents. The system has been implemented [31] as an extension of the popular TEX system and will become part of future version of TikZ.[2] The generated animation are light-weight both in terms of file size and generation time, but require that the documents (or, at least, the graphic files) are stored in the SVG format. Our new algorithm is a natural extension of the Reingold–Tilford algorithm to the dynamic case, but while the original algorithm runs in linear time on all trees, we showed that the dynamic case leads to NP-complete problems. Fortunately, in practice, the hard subproblems can be solved satisfactorily using a greedy strategy – at least, that has been our finding

---

[2] Currently available in the development version at http://pgf.cvs.sourceforge.net.

for a limited number of examples such as the above animation; a perceptual study of animated drawings of dynamic graphs has not (yet) been conducted.

We see our algorithm as a first step towards a general set of algorithms for drawing dynamic graphs using animations, which we believe to have a great (and not yet fully realized) potential as parts of text documents. A next logical step would be a transferal of the Sugiyama method [16, 34] to the dynamic offline case.

References

1. Adelson-Velsky, G.M., Landis, E.M.: An algorithm for the organization of information. Dokl. Akad. Nauk USSR **3**(2), 1259–1263 (1962)
2. Archambault, D., Purchase, H., Pinaud, B.: Animation, small multiples, and the effect of mental map preservation in dynamic graphs. IEEE Trans. Visual. Comput. Graph. **17**(4), 539–552 (2011)
3. Archambault, D., Purchase, H.C.: The mental map and memorability in dynamic graphs. In: Proceedings of Visualization Symposium (PacificVis) 2012, pp. 89–96. IEEE Press (2012)
4. Bartram, L., Ware, C.: Filtering and brushing with motion. Inf. Visual. **1**(1), 66–79 (2002)
5. Beck, F., Burch, M., Diehl, S., Weiskopf, D.: The state of the art in visualizing dynamic graphs. In: State of the Art Reports of the 16th Eurographics Conference on Visualization, EuroVis 2014, pp. 83–103. Eurographics Association (2014)
6. Brüggemann-Klein, A., Wood, D.: Drawing trees nicely with TEX. Electron. Publishing **2**(2), 101–115 (1989)
7. Bulterman, D., Jansen, J., Cesar, P., Mullender, S., Hyche, E., DeMeglio, M., Quint, J., Kawamura, H., Weck, D., García Pañeda, X., Melendi, D., Cruz-Lara, S., Hanclik, M., Zucker, D.F., Michel, T.: Synchronized multimedia integration language (SMIL 3.0), W3C Recommendation 01 December 2008. Technical report REC-SMIL3-20081201, The World Wide Web Consortium (W3C) (2008). http://www.w3.org/TR/2008/REC-SMIL3-20081201
8. Burch, M., Beck, F., Weiskopf, D.: Radial edge splatting for visualizing dynamic directed graphs. In: Proceedings of the International Conference on Computer Graphics Theory and Applications, IVAPP 2012, pp. 603–612. SciTe Press (2012)
9. Burch, M., Diehl, S.: TimeRadarTrees: visualizing dynamic compound digraphs. Comput. Graph. Forum **27**(3), 823–830 (2008)
10. Chimani, M., Gutwenger, C., Jünger, M., Klein, K., Mutzel, P., Schulz, M.: The open graph drawing framework. In: Poster at the 15th International Symposium on Graph Drawing 2007 (GD 2007) (2007)
11. Cohen, R.F., Di Battista, G., Tamassia, R., Tollis, I.G.: Dynamic graph drawings: trees, series-parallel digraphs, and planar st-digraphs. SIAM J. Comput. **24**(5), 970–1001 (1995)
12. Cohen, R.F., Di Battista, G., Tamassia, R., Tollis, I.G., Bertolazzi, P.: A framework for dynamic graph drawing. In: Proceedings of the 8th Annual Symposium on Computational Geometry, SCG 1992, pp. 261–270. ACM Press (1992)
13. Dahlström, E., Dengler, P., Grasso, A., Lilley, C., McCormack, C., Schepers, D., Watt, J.: Scalable vector graphics (SVG) 1.1 (2nd edn.), W3C Recommendation 16 August 2011. Technical report REC-SVG11-20110816, The World Wide Web Consortium (W3C) (2011). http://www.w3.org/TR/2011/REC-SVG11-20110816

14. Diehl, S., Görg, C., Kerren, A.: Foresighted graphlayout. Technical report A/02/2000, FB Informatik, University Saarbrücken, Saarbrücken, Germany (2000)

15. Diehl, S., Görg, C., Kerren, A.: Preserving the mental map using foresighted layout. In: Proceedings of the 3rd Joint Eurographics-IEEE TCVG Conference on Visualization, vol. 1, pp. 175–184. The Eurographics Association (2001)

16. Eades, P., Sugiyama, K.: How to draw a directed graph. J. Inf. Process. **13**(4), 424–436 (1990)

17. Ellson, J., Gansner, E.R., Koutsofios, E., North, S.C., Woodhull, G.: Graphviz and dynagraph - static and dynamic graph drawing tools. In: Junger, M., Mutzel, P. (eds.) Graph Drawing Software. Mathematics and Visualization, pp. 127–148. Springer, Heidelberg (2004)

18. Eppstein, D.: Trees in TEX. TUGboat **6**(1), 31–35 (1985)

19. Federico, P., Aigner, W., Miksch, S., Windhager, F., Zenk, L.: A visual analytics approach to dynamic social networks. In: Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies, i-KNOW 2011, pp. 47:1–47:8. ACM Press (2011)

20. Görg, C., Birke, P., Pohl, M., Diehl, S.: Dynamic graph drawing of sequences of orthogonal and hierarchical graphs. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 228–238. Springer, Heidelberg (2005). doi:10.1007/978-3-540-31843-9_24

21. Graham, M., Kennedy, J.: A survey of multiple tree visualisation. Inf. Visual. **9**(4), 235–252 (2010)

22. Greilich, M., Burch, M., Diehl, S.: Visualizing the evolution of compound digraphs with TimeArcTrees. Comput. Graph. Forum **28**(3), 975–982 (2009)

23. Groh, G., Hanstein, H., Wörndl, W.: Interactively visualizing dynamic social networks with DySoN. In: Proceedings of the Workshop on Visual Interfaces to the Social and the Semantic Web, VISSW 2009 (2009)

24. Ierusalimschy, R.: Programming in Lua, 2nd edn. Lua.org, San Francisco (2006)

25. Johnson, B., Shneiderman, B.: Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In: Proceedings of the 2nd IEEE Conference on Visualization 1991, VIS 1991, pp. 284–291. IEEE Press (1991)

26. Knuth, D.E.: Optimum binary search trees. Acta Informatica **1**(1), 14–25 (1971)

27. Moen, S.: Drawing dynamic trees. IEEE Softw. **7**(4), 21–28 (1990)

28. Reda, K., Tantipathananandh, C., Johnson, A., Leigh, J., Berger-Wolf, T.: Visualizing the evolution of community structures in dynamic social networks. Comput. Graph. Forum **30**(3), 1061–1070 (2011)

29. Reingold, E.M., Tilford, J.S.: Tidier drawings of trees. IEEE Trans. Softw. Eng. **7**(2), 223–228 (1981)

30. Robertson, G.G., Mackinlay, J.D., Card, S.K.: Cone trees: Animated 3D visualizations of hierarchical information. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 1991, pp. 189–194. ACM Press (1991)

31. Skambath, M.: Algorithmic drawing of evolving trees. Master's thesis, Institute of Theoretical Computer Science, Universität zu Lübeck, Germany (2016)

32. Skambath, M., Tantau, T.: Offline drawing of dynamic trees: algorithmics and document integration. SVG Version of this document. http://www.informatik.uni-kiel. de/~msk/pub/2016-dynamic-trees/main.html

33. Stasko, J., Zhang, E.: Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In: Proceedings of the IEEE Symposium on Information Vizualization 2000, INFOVIS 2000, pp. 57–65. IEEE Press (2000)

34. Sugiyama, K., Tagawa, S., Toda, M.: Effective representations of hierarchical structures. Technical report 8, International Institute for Advanced Study of Social Information Science, Fujitsu (1979)

35. Sweet, R.E.: Empirical estimates of program entropy. Report Stan-CS-78-698, Department of Computer Science, Stanford University, Stanford, CA, USA (1978)

36. Tantau, T.: Graph drawing in TikZ. J. Graph Algorithms Appl. **17**(4), 495–513 (2013)

37. Tantau, T.: The Ti*k*Z and pgf packages, manual for version 3.0.0 (2015). http://sourceforge.net/projects/pgf/

38. Walker II, J.Q.: A node-positioning algorithm for general trees. Softw.: Pract. Exp. **20**(7), 685–705 (1990)

39. Ware, C., Bobrow, R.: Motion to support rapid interactive queries on node-link diagrams. ACM Trans. Appl. Percept. **1**(1), 3–18 (2004)

40. Wetherell, C., Shannon, A.: Tidy drawings of trees. IEEE Trans. Softw. Eng. **5**(5), 514–520 (1979)