**Lines:** 385

# Visualizing Evolving Trees

Anonymized submission

**Abstract.** Evolving trees arise in many real-life scenarios from computer file systems and dynamic call graphs, to fake news propagation and disease spread. Most layout algorithms for static trees do not work well in an evolving setting (e.g., they are not designed to be stable between time steps). Dynamic graph layout algorithms are better suited to this task, although they often introduce unnecessary edge crossings. With this in mind we propose two methods for visualizing evolving trees that guarantee no edge crossings, while optimizing (1) desired edge length realization, (2) layout compactness, and (3) stability. We evaluate the two new methods, along with five prior approaches (three static and two dynamic), on real-world datasets using quantitative metrics: stress, desired edge length realization, layout compactness, stability, and running time. The new methods are fully functional and available on github.

## 1 Introduction

Dynamic graph visualization is used in many fields including social networks [28], bibliometric networks [48], software engineering [14], and pandemic modeling [7]; see the survey by Beck *et al.* [10]. Here we focus on a special case, *evolving trees*. In evolving trees the dynamics are captured only by growth (whereas in general dynamic graphs, nodes and edges can also disappear). While this is a significant restriction of the general dynamic graph model, evolving trees are common in many domains including the Tree of Life [36] and the Mathematics Genealogy Graph [35]. An evolving tree can also model disease spread, where nodes correspond to infected individuals and a new node $v$ is added along with an edge to existing node $u$ if $u$ infected $v$. Visualizing this process can help us see how the infection spreads, the rate of infection, and to identify "super-spreaders."

There are several methods and tools that can be used to visualize evolving trees [2, 15, 16, 38], however, most of them have limitations that can impact their usability in this domain. Some represent nodes only as points ignoring labels [15, 16], which makes them less useful in real-life applications where it is important to see what each node represents. Others utilize the level-by-level approach for drawing hierarchical graphs [27, 40], which does not capture the underlying graph structure well. Force-directed algorithms tend to better capture the underlying graph structure [31], although they may introduce unnecessary edge crossings. With this in mind, we propose two methods for drawing crossing-free evolving trees that optimize the following desirable properties:

1. **Desired edge length realization:** The Euclidean distance between two nodes $u$ and $v$ in the layout should realize the corresponding pre-specified edge length $l(u, v)$, or be uniform when no additional information is given. This is important in several domains, e.g., when visualizing phylogenetic trees [6], where the edge length represents evolutionary distance between two species.
2. **Layout compactness**: The drawing area should be proportional to the total area needed for all the labels. A good visualization should have the labeled graph drawn in a compact way [39]. This prevents the trivial solution of scaling the layout until all overlaps and crossings are removed, which can create vast empty spaces in the visualization.
3. **Stability:** Between time steps, nodes should move as little as possible. This helps the viewer maintain a mental map of the graph [37]. If the graph moves around too much, it is difficult to see where new nodes and edges are added and we lose the context of the new node's relation to the rest of the graph.

We propose two force-directed methods that ensure no edge crossings and optimize desired edge length, compactness and stability. Minimizing edge crossings is important in graph readability [42], and since we work with trees, a layout without edge crossings is possible and desirable. We use two trees extracted from Tree of Life [36] and the Mathematics Genealogy [35] projects to demonstrate the new methods and quantitatively evaluate their performance, measuring desired edge length realization, compactness, stability, stress, crossings, and running time. We also evaluate the performance of five earlier methods, showing the two proposed methods perform well overall; see Fig. 1.

## 2 Related Work

Dynamic graph drawing has a long history [11, 46] and two broad categories: offline and online. In the easier offline setting we assume that all the data about the dynamics is known in advance. Algorithms for offline dynamic visualization use different approaches including combining all time-slice instances into a single supergraph [18–20], connecting the same node in consecutive time-slices and optimizing them simultaneously [22–24], providing animation [4], and showing small multiples type visualization [3]. *DynNoSlice* by Simonetto *et al.* [44] is one of the most recent approaches for this setting and is different from the prior methods as it does not rely on discrete time-slices.

Online dynamic graph drawing deals with the harder problem – when we do not know in advance what changes will occur. One can optimize the current view, given what has happened in the past, but cannot look into the future, as the information is not available. Cohen *et al.* [15,16] and Workman *et al.* [50] describe algorithms for dynamic drawing of trees that place nodes that are equidistant from the root on the same level (same $y$ coordinate). These algorithms do not take edge lengths into consideration, and the hierarchical nature of the layout can lead to exponential differences between the shortest and longest edges.

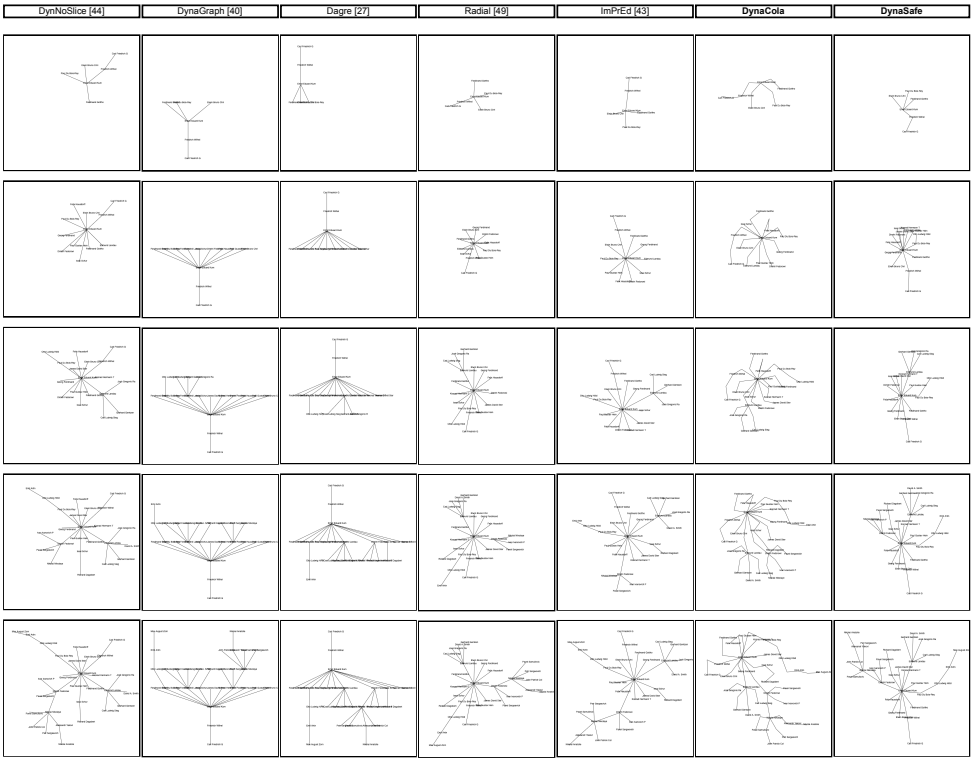



**Fig. 1.** Layouts from DynNoSlice, DynaGraph, Dagre, Radial, ImPrEd, DynaCola and DynaSafe of the same evolving math genealogy tree; each row adds six new nodes.

*DynaDAG* is an online graph drawing method for drawing dynamic directed acyclic graphs as hierarchies [40]. This method moves nodes between adjacent ranks based on the median sort. It was not specifically developed[1] for trees and may introduce crossings; see Fig. 2. Other approaches for online dynamic graph drawing have maintained the horizontal and vertical position of nodes [37], used node aging methods [29], and adapted multilevel approaches [17] (using FM$^3$ [30]). Online approaches have also been implemented on the GPU [25]. However, these methods do not guarantee crossing-free layouts for trees and do not take into account desired edge lengths.

*Dagre* is a multi-phase algorithm for drawing directed graphs based on [27]. The initial phase finds an optimal rank assignment using the network simplex algorithm. Then it sets the node order within ranks by an iterative heuristic incorporating a weight function and local transpositions to reduce crossings (via the barycenter heuristic) [32]. However, since Dagre draws graphs in a hierarchical structure, the edge lengths may vary arbitrarily; see Fig. 2.

[1] We have used the implementation available in the DynaGraph system: https://www.dynagraph.org/.

(a) (b)

**Fig. 2.** (a) A DynaDAG layout of a tree with 100 nodes that introduces **edge crossings**. (b) A Dagre layout of a tree with 300 nodes, with **large edge length variability**. Both examples show that **nodes are too close to each other** if labeled. The trees are extracted from the math genealogy dataset.

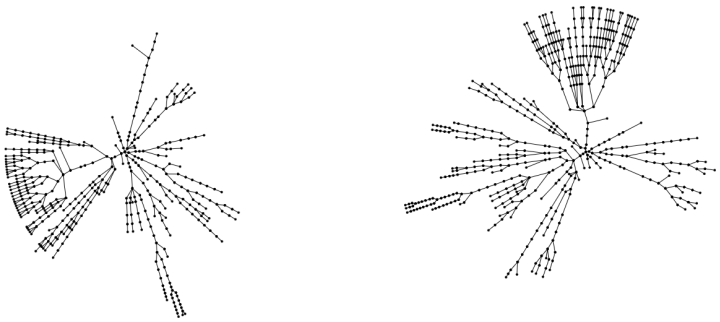
The *radial* layout implemented in yFiles [49] displays each biconnected component in a circular fashion[2]. Radial layouts were introduced by Kar *et al.* [33] for static graphs. Dougrusoz *et al.* [21] described an interactive tool for dynamic graph visualization based on the radial layout. Six and Tollis [45] adapted the radial idea to circular drawings of static biconnected graphs and experimentally showed that their layout has fewer edge crossings. Kaufmann *et al.* [34] extended this model to handle dynamic graphs, providing the basis of the yFiles *radial* implementation. Pavlo *et al.* [41] adapted the idea to make the radial layout computation parallelizable. Bachmaier [5] further improved the radial layout algorithm for static graphs by adapting the hierarchical approach [47] to minimize edge crossings. Radial layout methods have more freedom than the traditional level-by-level tree layout methods. Nevertheless, they are still constrained and can result in unstable visualization for dynamic graphs in general and evolving trees in particular; see Fig. 3.

Force-directed algorithms [18–20, 43, 44] underlie many static and dynamic graph visualization methods. Unlike hierarchical and radial approaches, force-directed algorithms place nodes at arbitrary positions, and so tend to generate more compact layouts that better realize desired edge lengths. By adjusting forces appropriately, one can also generate stable layouts by this approach. In particular, ImPrEd, by Simonetto *et al.* [43], provides a force-directed approach to improve a given initial layout, without introducing new edge crossings. To the best of our knowledge, there are no force-directed methods for evolving trees.

## 3 Algorithms for Visualizing Evolving Trees

Here, we describe two force-directed algorithms for evolving tree visualization, *DynaCola* and *DynaSafe*, that realize desired edge lengths without creating crossings, and optimize compactness and stability. DynaCola avoids edge crossings by creating and maintaining a "collision region" for each edge. While collision detection/prevention is usually applied to nodes, by carefully applying it to the

[2] We use this radial layout algorithm later for our experiments.



(a) (b)

**Fig. 3.** A radial layout of a 400-node evolving tree of life (a) and the layout after adding 100 nodes (b). As most of the growth occurred on the right side, it is easy to see the **instability** – a lot of movement was needed to accommodate it. Also, **nodes are too close to each other** if labeled.

edges we can prevent all edge crossings. DynaSafe prevents edge crossings with a "safe" coordinate update at every step of the algorithm. Before updating a coordinate, it first checks whether the update will introduce a crossing and then limits the update magnitude to avoid the crossing.

### 3.1 DynaCola

DynaCola stands for Dynamic Collision, as the algorithm uses the collision forces to prevent edge crossings. This is a force-directed algorithm, augmented with edge-regions used to prevent crossings; see Algorithm 1 in the Appendix. Recall that we are gradually growing a tree, one node at a time, while maintaining a crossing-free layout and optimizing desirable properties (desired edge lengths, compactness, stability). The DynaCola force-directed algorithm relies on the following forces and is implemented in d3.js [12]:

- A force $f_E$ for each edge, to realize the desired edge length. The strength of this force is proportional to the difference between the edge distance in the layout and the desired edge length.
- A general repulsive force $f_R$ defined for all pairs of nodes and implemented with the Barnes-Hut quad-tree data structure [8]. This helps realize the global structure of the underlying tree.
- A collision force $f_C$ for each edge, described in details below. This force prevents edge crossings.
- A gravitational force $f_G$ that attracts all nodes to the center of mass. This force draws the nodes closer together and improves compactness.

To ensure that no edges cross during an update, we define a collision region around each edge: if any edge/node moves too close to another edge, it will be

pushed away. To create a collision region for an edge $e = (u, v)$, we can create collision circles with diameter equal to the length of $e$ for both $u$ and $v$. Then every point of $e$ will be either inside the collision region of $u$ or $v$. However, the sum of all collision regions for all nodes will be unnecessarily large and the layout will not be compact. With the help of subdivision nodes along the edges, we can reduce the sum of all collision regions. Let $e = (u, v)$ be an edge in the graph. We use a set of subdivision nodes $V_s = \{v_1, v_2, \cdots, v_k\}$ and replace the edge $(u, v)$ by a set of edges $E_s = \{(u, v_1), (v_1, v_2), \cdots, (v_k, v)\}$. We assign the desired edge length of an edge in $E_s$ equal to $l(u, v)/|E_s|$, where $l(u, v)$ is the desired edge length. In general, the number of subdivision nodes per edge should be a small constant $n_s$ (by default $n_s = 1$), since the complexity of the algorithm increases as $n_s$ increases. Also, note that $n_s$ determines the number of bends per edge (no bends when $n_s = 0$, one bend when $n_s = 1$, and so on). Note that, the collision force does not follow any hard constraint, even after having a collision region edge crossings may happen. If existing edges introduce crossings, then we roll back to previous crossing-free coordinates.

When a new node is added to the tree, a new edge also is added, with one of its endpoints already placed. To place the new node, we randomly sample a set of 100 nearby points at a distance equal to the desired edge length, trying to find a crossing-free position. If we cannot find such a suitable point, we gradually reduce the distance and repeat the search until we find a crossing free position. Once the new node has been placed, we subdivide its adjacent edge as described above.

### 3.2 DynaSafe

DynaSafe stands for Dynamic Safety, as the algorithm prioritizes safe moves and will not make a move if it introduces an edge crossing, see Algorithm 2 in Appendix. DynaSafe is also a force-directed algorithm, however, it differs from DynaCola as it draws straight-line edges (rather than edges with bends). The algorithm utilizes the following forces and is implemented in d3.js [12]

- A force $f_E$ on the edges to realize the desired edge length. The strength of the force is proportional to the difference between the edge distance in the layout and the desired edge length.
- A stress-minimizing force $f_S$ on every pair of nodes not connected by an edge, used to improve global structure. The desired distance is the shortest-path distance between the pair, and the magnitude of the force is proportional to the difference between the realized and desired distance.
- A repulsive force $f_R$ between nodes, which somewhat counter-intuitively, improves the compactness of the layout. The force has an elliptical contour since we are optimizing for labeled nodes, where the value of the $x$ coordinate must be farther away from other labels than the value of the $y$ coordinate.
- A gravitational force $f_G$ that attracts all nodes to the center of mass. This force draws the nodes closer together and improves compactness.

DynaSafe prevents edge crossings from occuring at any time by updating the coordinates safely: if the proposed new coordinate of a node introduces crossings, we gradually reduce the magnitude of the movement until the crossing is avoided. To place the new node, we randomly sample a set of 100 nearby points to find a crossing-free position for its adjacent edge. If we cannot find a crossing free position using the sample points, we continuously reduce the edge length until we find a crossing free position. Once the node is added, an iteration of force-directed algorithm optimizes the layout (again without introducing crossings).

By the nature of force-directed algorithms, after one phase of force computations each node has a proposed new position. Before moving any node to its proposed new position, we check that the move is "safe," i.e., it does not introduce a crossing. If the movement of a node introduces any crossings, then the magnitude of the move is set to $p\%$ of the original movement. This is repeated (if needed) at most $q$ times, and if the crossing is still unavoidable then the node is not moved in this phase. By default $p = 0.8$ and $q = 12$.

## 4 Experimental Evaluation

We evaluate DynaCola and DynaSafe, along with five earlier methods: DynNoSlice, DynaGraph, Dagre, Radial, and ImPrEd. We use two evolving trees to visually compare the results, as well quantitatively evaluate the desired properties.

### 4.1 Datasets

We use two real-world datasets to extract evolving trees for our experiments.

**The Tree of Life:** captures the evolutionary progression of life on Earth [36]. The underlying data is a tree structure with a natural time component. As a new species evolves, a new node in the tree is added. The edges give the parent-child relation of the nodes, where the parent is the original species, and the child is the new species. We use a subset of this graph with 500 nodes. The maximum node degree of this tree is 5, and the radius is 24.

**The Mathematics Genealogy:** shows advisor-advisee relationships in the world of mathematics, stretching back to the middle ages [35]. The dataset includes the thesis titles, students, advisors, dates, and number of descendants. The total number of nodes is around 260,000 and is continuously updated. While this data is not quite a tree (or even connected, or planar), we extract a subset to create a tree with 500 nodes. The maximum node degree of this tree is 5 and the radius is 14.

### 4.2 Evaluation Metrics

We use standard metrics for each of our desired properties: desired edge length preservation, compactness, and stability. Additionally, we compute the stress of

the drawing and the number of crossings. This gives a total of five quantitative measures. For each of these measures we define a loss function as follows:

**Desired Edge Length (DEL):** To measure how close the realized edge lengths are to the desired edge lengths, we find the mean squared error between these two values. Given the desired edge lengths $\{l_{ij} : (i,j) \in E\}$ and coordinates of the nodes $X$ in the computed layout, we evaluate with the following formula:

$$\texttt{Desired edge length loss} = \sqrt{\frac{1}{|E|}\sum_{(i,j)\in E}\left(\frac{||X_i - X_j|| - l_{ij}}{l_{ij}}\right)^2} \tag{1}$$

This measures the root mean square of the relative error as in [1], producing a non-negative number, with 0 corresponding to a perfect realization.

**Compactness:** To measure the compactness of each layout, we use the ratio between the drawing area and the sum of the areas for all labels [9]. We assume that a label is at most 16 characters, as we abbreviate longer labels. The sum of the areas for all labels gives the minimum possible area needed to draw all labels without overlaps (ignoring any space needed for edges). The area of the actual drawing is given by the smallest bounding rectangle, once the drawing has been scaled up until there are no overlapping labels. Once we have this scaled drawing, we find the positions of the nodes with the largest and smallest x and y values ($X_{max,0}$, $X_{min,0}$, $X_{max,1}$ and $X_{min,1}$). Using these values we calculate the area of the bounding rectangle.

$$\texttt{Compactness loss} = \frac{(X_{max,0} - X_{min,0})(X_{max,1} - X_{min,1})}{\sum_{v\in V} \text{label\_area}(v)} \tag{2}$$

This formula produces a non-negative number; the ideal value for this measure is 1 and corresponds to a perfect space utilization.

**Stability:** To measure stability, we consider how much each of the nodes moved after adding a new node. We then sum the movements of all nodes over all time steps. Since different algorithms use different amounts of drawing areas, we divide the value by the drawing area to normalize the results. This measure is similar to that used in DynNoSlice [44], but since DynNoSlice does not use time slices, it is closer to the measure found in [13]:

$$\texttt{Stability loss} = \frac{\sum_{v\in V}\sum_{t=1}^{T-1} ||X_v(t+1) - X_v(t)||}{(X_{max,0} - X_{min,0})(X_{max,1} - X_{min,1})} \tag{3}$$

Here, $T$ is the maximum time (500 in our two datasets). This formula produces a non-negative number; the ideal value is 0 and corresponds to a perfectly stable layout (no movement of any already placed nodes).

**Stress:** This measure evaluates the global quality of the layout, looking at the differences between the realized distance between any pair of nodes and the actual distance between them. This measure is used in a variety of graph drawing algorithms [13, 26, 44]:

$$\texttt{Stress loss} = \frac{\left(\sum_{i \neq j} \left(D_{i,j} - ||X_i - X_j||\right)^2\right)^{1/2}}{\sum_{i \neq j} ||X_i - X_j||} \tag{4}$$

This formula produces a non-negative number; the ideal value is 0 and corresponds to a perfect embedding (that captures all graph distances by the realized Euclidean distances).

**Edge Crossings:** Finally, we measure the number of edge crossings in each of the outputs. Note that our algorithms DynaSafe and DynaCola enforce "no edge crossings" as a hard constraint. However, DynNoSlice and DynaGraph do not have such a constraint and so can and indeed do, introduce crossings. Therefore we include the number of edge crossings for a complete comparison.

### 4.3 Experimental Setup

We compare these algorithms to five previous algorithms: DynNoSlice, DynaGraph, Dagre, Radial, and ImPrEd. We note that while Dagre, Radial, and ImPrEd are not specifically designed for dynamic graphs, they can be modified for this purpose. Specifically, we can use the layout of a tree at step $i$ to initialize the layout of the tree at step $i + 1$, add the new edge, and update the layout.

We consider the simplest case for the desired edge length by using a uniform length of 100 for all edges. This is a necessary parameter for our algorithms DynaCola and DynaSafe, but only needed in the other four algorithms in order to compute the desired edge length measure. To be able to compare our methods to the other four (that do not take desired edge length into account), we set the desired edge length equal to the average edge length obtained in the layout. We then normalize these values for a fair comparison.

The performance of DynNoSlice depends heavily on two parameters, $\tau$ and $\delta$ that must be tuned. With the help of the authors, we found $\tau = 16$ and $\delta = 4$ worked well for our 500-node trees. The performance of ImPrEd depends on two parameters: repulsion force and the number of iterations. The default values of repulsion force and the number of iterations are equal to one and 200 respectively. We have used the default values. The larger the number of iterations is, the better the output of ImPrEd is. However, the running time increases as the number of iterations increases. We keep the number of iterations equal to 200 since it already takes more than 4 hours to compute the 500-node trees. The performance of DynaCola depends on the number of subdivision nodes $n_s$; we use $n_s = 1$ for the experiments. For the radial layout algorithm, we have used the default settings in the yFiles [49] implementation. The other algorithms are also used with their default settings.

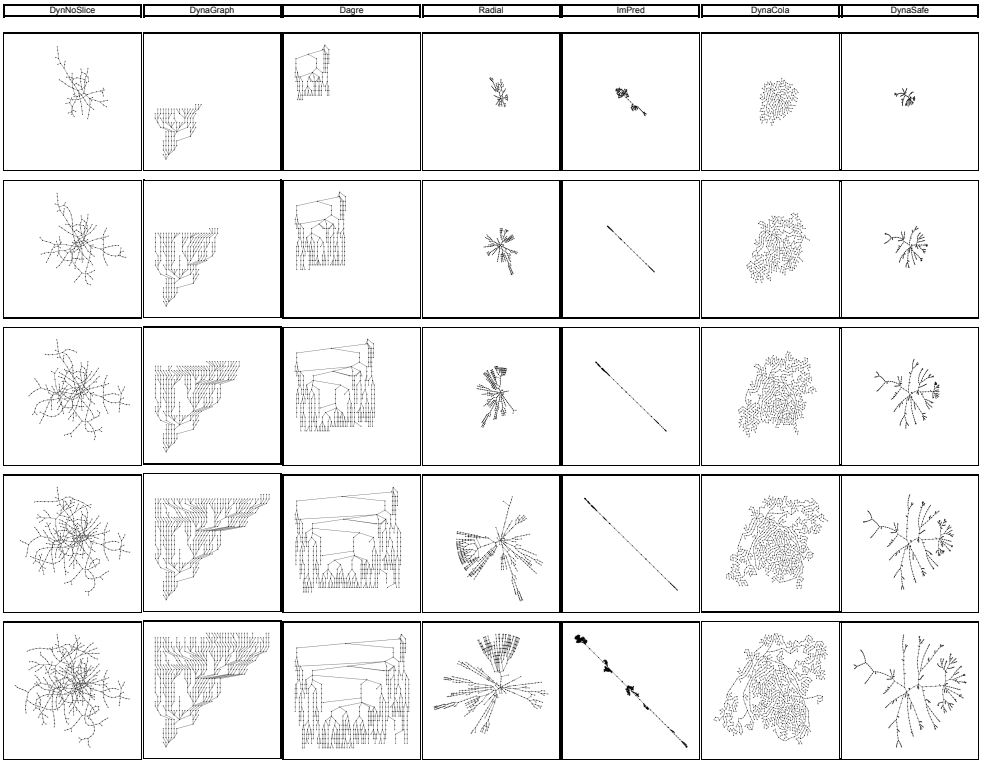



**Fig. 4.** Layouts obtained by the seven methods for the tree of life dataset.

### 4.4 Results

Both the visual and quantitative results indicate that the two new methods perform well overall; see Fig. 4.

**Desired Edge Lengths:** The quantitative results are shown in Table 1. We use green to show the best results and yellow for the second best. and indicate that DynNoSlice, DynaCola, and ImPrEd perform well. For the math genealogy 500-node tree, ImPrEd is the best. However, both DynNoSlice and ImPrEd have significantly larger running times (measured in hours, rather than minutes or seconds). Moreover, while ImPrEd does well on the math genealogy graph, it does not do well on the tree of life graph. For the math genealogy 500-node tree, DynaCola is the second best and DynNoSlice is third. For the tree of life dataset, DynaCola is the best, DynNoSlice is the second best and DynaSafe is third. DynaGraph has the worst performance – not surprising given that it is a hierarchical layout, which is forced to use some very long edges near the root.

**Compactness:** The quantitative results are shown in Table 2 and indicate that DynNoSlice outperforms the rest of the algorithms. DynaCola is second best and DynaSafe is third. Here, Dagre has the worst performance.

| Nodes | DynNoSlice | DynaGraph | Dagre | Radial | ImPrEd | DynaCola | DynaSafe |
|---|---|---|---|---|---|---|---|
| 100 MG | 0.37691 | 1.95933 | 0.682568 | 0.653853 | 0.219103 | 0.282521 | 0.589865 |
| 200 MG | 0.36552 | 1.95179 | 0.679827 | 0.640628 | 0.213615 | 0.270322 | 0.575430 |
| 300 MG | 0.35007 | 1.94213 | 0.666440 | 0.63058 | 0.204821 | 0.253877 | 0.564747 |
| 400 MG | 0.34402 | 1.93822 | 0.646479 | 0.619203 | 0.193037 | 0.243184 | 0.553141 |
| 500 MG | 0.33377 | 1.91979 | 0.639766 | 0.592694 | 0.182071 | 0.237139 | 0.548756 |
| 100 TOL | 0.21675 | 1.28710 | 0.448483 | 0.448205 | 0.45402 | 0.158071 | 0.411747 |
| 200 TOL | 0.21972 | 1.37271 | 0.494261 | 0.460161 | 0.49120 | 0.166190 | 0.443935 |
| 300 TOL | 0.23986 | 1.40404 | 0.510473 | 0.481034 | 0.52016 | 0.176748 | 0.453332 |
| 400 TOL | 0.25597 | 1.45660 | 0.553543 | 0.510352 | 0.59326 | 0.183856 | 0.470334 |
| 500 TOL | 0.26652 | 1.52650 | 0.581648 | 0.530249 | 0.61093 | 0.189373 | 0.485759 |

**Table 1.** Desired edge lengths of math genealogy tree (MG) and tree of life (TOL).

| Nodes | DynNoSlice | DynaGraph | Dagre | Radial | ImPrEd | DynaCola | DynaSafe |
|---|---|---|---|---|---|---|---|
| 100 MG | 85.60 | 192.07 | 219.20 | 153.53 | 161.23 | 124.90 | 147.80 |
| 200 MG | 87.94 | 196.29 | 224.54 | 162.80 | 161.20 | 130.87 | 153.31 |
| 300 MG | 95.24 | 201.41 | 225.86 | 169.34 | 171.92 | 137.00 | 159.87 |
| 400 MG | 98.89 | 206.48 | 227.74 | 175.07 | 181.20 | 145.47 | 169.68 |
| 500 MG | 106.82 | 208.39 | 236.94 | 192.53 | 187.94 | 149.43 | 174.93 |
| 100 TOL | 96.46 | 196.79 | 223.43 | 179.05 | 179.20 | 147.82 | 160.58 |
| 200 TOL | 100.24 | 214.29 | 231.38 | 183.06 | 218.29 | 154.10 | 169.38 |
| 300 TOL | 110.56 | 216.16 | 239.98 | 190.82 | 329.27 | 157.19 | 170.04 |
| 400 TOL | 119.98 | 233.85 | 255.76 | 203.92 | 416.27 | 167.52 | 194.28 |
| 500 TOL | 126.85 | 235.72 | 272.82 | 214.09 | 528.01 | 173.94 | 196.03 |

**Table 2.** Compactness of math genealogy tree (MG) and the tree of life (TOL).

**Stability:** The quantitative results are shown in Table 3 and idicate that DynaCola does best. DynaGraph is second, and DynaSafe is third. The radial layout performs worse in this metric because it rotates the subtrees as more edges are added.

**Stress:** The quantitative results are shown in Table 4. In general, DynaSafe does much better on this measure than the rest. Again, ImPrEd performs well for the regular-shaped math genealogy tree but does not perform well for the tree of life. For the tree of life DynNoSlice is second and DynaCola third. DynaGraph and Dagre perform the worst in this metric due to the limitations inherent in the hierarchical layout. Note that the stress is normalized, so the numbers are comparable.

**Edge Crossings:** The quantitative results are shown in Table 5. There are five winners here – the five algorithms that prevent any edge crossings: Dyna-

| Nodes | DynNoSlice | DynaGraph | Dagre | Radial | ImPrEd | DynaCola | DynaSafe |
|---|---|---|---|---|---|---|---|
| 100 MG | 0.001584 | 0.001393 | 0.0016502 | 0.001998 | 0.001530 | 0.001348 | 0.001459 |
| 200 MG | 0.000752 | 0.000447 | 0.0012497 | 0.001839 | 0.000598 | 0.000264 | 0.000410 |
| 300 MG | 0.000577 | 0.000227 | 0.0010083 | 0.001450 | 0.000437 | 0.000225 | 0.000295 |
| 400 MG | 0.000249 | 0.000190 | 0.0009504 | 0.001203 | 0.000391 | 0.000164 | 0.000216 |
| 500 MG | 0.000037 | 0.000014 | 0.0007591 | 0.001047 | 0.000026 | 0.000011 | 0.000019 |
| 100 TOL | 0.000437 | 0.000139 | 0.001241 | 0.003609 | 0.000491 | 0.000105 | 0.000163 |
| 200 TOL | 0.000323 | 0.000125 | 0.001196 | 0.003408 | 0.008305 | 0.000097 | 0.000146 |
| 300 TOL | 0.000263 | 0.000099 | 0.001163 | 0.003174 | 0.005305 | 0.000072 | 0.000106 |
| 400 TOL | 0.000235 | 0.000073 | 0.001136 | 0.002490 | 0.001937 | 0.000064 | 0.000101 |
| 500 TOL | 0.000199 | 0.000071 | 0.000834 | 0.001941 | 0.000810 | 0.000052 | 0.000101 |

**Table 3.** Stability of math genealogy tree (MG) and tree of life (TOL).

| Nodes | DynNoSlice | DynaGraph | Dagre | Radial | ImPrEd | DynaCola | DynaSafe |
|---|---|---|---|---|---|---|---|
| 100 MG | 113.10 | 150.59 | 230.75 | 125.37 | 89.43 | 76.51 | 49.44 |
| 200 MG | 161.45 | 186.17 | 250.98 | 172.05 | 120.45 | 151.74 | 68.42 |
| 300 MG | 179.48 | 264.90 | 286.77 | 205.98 | 148.02 | 184.47 | 94.05 |
| 400 MG | 186.68 | 292.66 | 286.83 | 262.09 | 173.92 | 227.98 | 107.56 |
| 500 MG | 249.61 | 393.11 | 396.72 | 314.93 | 203.54 | 291.39 | 109.00 |
| 100 TOL | 136.52 | 210.06 | 263.98 | 192.64 | 163.02 | 128.45 | 59.77 |
| 200 TOL | 165.75 | 262.24 | 325.65 | 243.59 | 349.28 | 201.76 | 65.10 |
| 300 TOL | 181.62 | 305.15 | 369.57 | 287.93 | 427.09 | 220.63 | 81.81 |
| 400 TOL | 254.20 | 328.59 | 398.11 | 317.28 | 509.32 | 306.89 | 93.99 |
| 500 TOL | 285.19 | 400.81 | 461.43 | 374.02 | 593.19 | 351.23 | 119.77 |

**Table 4.** Stress scores of math genealogy tree (MG) and tree of life (TOL).

Cola, DynaSafe, Radial, Dagre, and ImPrEd. DynNoSlice and DynaGraph do introduce some crossings.

**Running time:** The Radial layout has the lowest running time, taking 34.93 seconds and 28.03 seconds, respectively, to draw the 500-node math genealogy tree and tree of life. On the other end, DynNoSlice is the slowest algorithm, taking more than 6 hours to draw the 500-node trees. Both DynNoSlice and ImPrEd take significantly longer running time compared to other algorithms. ImPrEd takes more than four hours to draw the 500-node trees. Our two new methods are not as fast as the Radial algorithm and not as slow as DynNoSlice, taking about 5 minutes on the 500-node trees; see Fig. 5.

## 5 Discussion and Limitations

While there are many algorithms and tools for drawing static trees, only a few can handle dynamic trees well. Among those, even fewer takes edge labels into

| Nodes | DynNoSlice | DynaGraph | Dagre | Radial | ImPrEd | DynaCola | DynaSafe |
|---|---|---|---|---|---|---|---|
| 100 MG | 43 | 8 | 0 | 0 | 0 | 0 | 0 |
| 200 MG | 82 | 11 | 0 | 0 | 0 | 0 | 0 |
| 300 MG | 168 | 11 | 0 | 0 | 0 | 0 | 0 |
| 400 MG | 217 | 13 | 0 | 0 | 0 | 0 | 0 |
| 500 MG | 277 | 13 | 0 | 0 | 0 | 0 | 0 |
| 100 TOL | 21 | 0 | 0 | 0 | 0 | 0 | 0 |
| 200 TOL | 67 | 0 | 0 | 0 | 0 | 0 | 0 |
| 300 TOL | 106 | 0 | 0 | 0 | 0 | 0 | 0 |
| 400 TOL | 176 | 0 | 0 | 0 | 0 | 0 | 0 |
| 500 TOL | 231 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 5.** Edge crossings of math genealogy tree (MG) and tree of life (TOL).

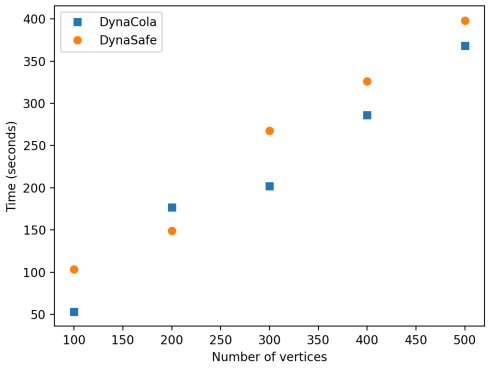



**Fig. 5.** Running time w.r.t. number of nodes in the math genealogy tree.

account while also preventing edge crossings. With this in mind, we described two methods that give better, readable layouts for evolving trees. We compared these two algorithms with others that have been set up for dynamic trees. With respect to the criteria that we have put forward, our algorithms match or exceed each of these algorithms. Fully functional prototypes and videos showing them in action are available online https://evolving-trees.github.io/dynamic-trees/. Source code and all experimental data can be found on github https://github.com/evolving-trees/evolving_tree.

Naturally, our work comes with several limitations that could be addressed in future work.

**Anticipating the Future**: Currently, the two new methods, DynaCola and DynaSafe perform well for evolving trees, where growth is the only type of change. A natural question is whether these algorithms can be generalized to the more challenging problems of online dynamic tree visualization. Answering

such question may need more precise modeling of the graph dynamics. Even though online dynamic graph drawing assumes no knowledge about the actual changes to the graph in the future, some prior knowledge of the graph may be available or predictable in advance. For example, knowing the expected depth or size of a tree or maximum degree of nodes (e.g., from domain knowledge about the specific type of graph) may help the layout algorithm reserve enough space for growth. In general, we anticipate that if one can model the evolving dynamics of the graph (e.g., probabilistically), incorporating knowledge of such dynamics into the drawing algorithm may help improve the resultant drawing; conversely, carefully defining compatible graph dynamics for a particular drawing algorithm will also allow us to identify the limitations of the given algorithm.

**Multi-level Label Display**: For simplicity, in this work we assume labels to be always shown in the drawing in a fixed font size. In practice, however, labels may come with different levels of importance and different desired font size. In that case, one might prefer to see only important labels displayed first in a zoomed out view of the graph, and later see more labels when zooming in. Incorporate such multi-level label display into the node placement strategy seems like an interesting and relevant problem.

**Finding Desired Properties**: We have proposed two different algorithms to solve the same evolving tree visualization problem, and each is associated with different benefits. Finding a continuous *spectrum* of algorithms with tunable parameters to balance the multiple desired properties would provide more flexibility. On the other hand, a careful human-subjects study may also help prioritize existing properties of the drawing, or help identify new desired properties from the specific tasks.

### Acknowledgements

We thank the authors of DynNoSlice and DynaGraph for their assistance with running and tuning algorithms. We also thank yFiles whose radial layout implementation we use in the evaluation.

## References

1. Ahmed, R., Luca, F.D., Devkota, S., Kobourov, S., Li, M.: Graph drawing via gradient descent, $(GD)^2$. In: 28th International Symposium on Graph Drawing and Network Visualization (GD). pp. 3–17. Springer (2020)
2. Archambault, D., Purchase, H., Pinaud, B.: Animation, small multiples, and the effect of mental map preservation in dynamic graphs. IEEE Transactions on Visualization and Computer Graphics **17**(4), 539–552 (2010)
3. Bach, B., Henry-Riche, N., Dwyer, T., Madhyastha, T., Fekete, J.D., Grabowski, T.: Small multipiles: Piling time to explore temporal patterns in dynamic networks. Computer Graphics Forum **34**(3), 31–40 (2015)
4. Bach, B., Pietriga, E., Fekete, J.D.: Graphdiaries: Animated transitions and temporal navigation for dynamic networks. IEEE Transactions on Visualization and Computer Graphics **20**(5), 740–754 (2013)

5. Bachmaier, C.: A radial adaptation of the sugiyama framework for visualizing hierarchical information. IEEE Transactions on Visualization and Computer Graphics **13**(3), 583–594 (2007)
6. Bachmaier, C., Brandes, U., Schlieper, B.: Drawing phylogenetic trees. In: International Symposium on Algorithms and Computation. pp. 1110–1121 (2005)
7. Balcan, D., Gonçalves, B., Hu, H., Ramasco, J.J., Colizza, V., Vespignani, A.: Modeling the spatial spread of infectious diseases: The global epidemic and mobility computational model. Journal of Computational Science **1**(3), 132–145 (2010)
8. Barnes, J., Hut, P.: A hierarchical o(n log n) force-calculation algorithm. Nature **324**(6096), 446–449 (1986)
9. Barth, L., Kobourov, S.G., Pupyrev, S.: Experimental comparison of semantic word clouds. In: Experimental Algorithms. pp. 247–258. Springer (2014)
10. Beck, F., Burch, M., Diehl, S., Weiskopf, D.: The state of the art in visualizing dynamic graphs. In: 16th Eurographics Conference on Visualization, (EuroVis). Eurographics Association (2014)
11. Beck, F., Burch, M., Diehl, S., Weiskopf, D.: A taxonomy and survey of dynamic graph visualization. In: Computer Graphics Forum. vol. 36, pp. 133–159. Wiley Online Library (2017)
12. Bostock, M., Ogievetsky, V., Heer, J.: $D^3$ data-driven documents. IEEE Transactions on Visualization and Computer Graphics **17**(12), 2301–2309 (2011)
13. Brandes, U., Mader, M.: A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing. In: 19th International Symposium on Graph Drawing (GD). pp. 99–110. Springer (2011)
14. Burch, M., Müller, C., Reina, G., Schmauder, H., Greis, M., Weiskopf, D.: Visualizing dynamic call graphs. In: Vision, Modeling, and Visualization (VMV). pp. 207–214 (2012)
15. Cohen, R.F., Di Battista, G., Tamassia, R., Tollis, I.G.: Dynamic graph drawings: Trees, series-parallel digraphs, and planar st-digraphs. SIAM Journal on Computing **24**(5), 970–1001 (1995)
16. Cohen, R.F., Di Battista, G., Tamassia, R., Tollis, I.G., Bertolazzi, P.: A framework for dynamic graph drawing. In: Proceedings of the eighth annual symposium on Computational geometry. pp. 261–270 (1992)
17. Crnovrsanin, T., Chu, J., Ma, K.L.: An incremental layout method for visualizing online dynamic graphs. In: 23rd International Symposium on Graph Drawing (GD). pp. 16–29. Springer (2015)
18. Diehl, S., Görg, C.: Graphs, they are changing. 10th International Symposium on Graph Drawing (GD) pp. 23–31 (2002)
19. Diehl, S., Görg, C., Kerren, A.: Foresighted graph layout. Technical Report, University of Saarland (2000)
20. Diehl, S., Görg, C., Kerren, A.: Preserving the mental map using foresighted layout. Proceedings of Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym) (2001)
21. Doğrusöz, U., Madden, B., Madden, P.: Circular layout in the graph layout toolkit. In: 4th International Symposium on Graph Drawing (GD). pp. 92–100. Springer (1996)
22. Erten, C., Harding, P.J., Kobourov, S.G., Wampler, K., Yee, G.: Graphael: Graph animations with evolving layouts. In: 11th International Symposium on Graph Drawing (GD). pp. 98–110. Springer (2003)
23. Erten, C., Kobourov, S.G., Le, V., Navabi, A.: Simultaneous graph drawing: Layout algorithms and visualization schemes. In: 11th International Symposium on Graph Drawing (GD). pp. 437–449. Springer (2003)

24. Forrester, D., Kobourov, S.G., Navabi, A., Wampler, K., Yee, G.V.: graphael: A system for generalized force-directed layouts. In: 12th International Symposium on Graph Drawing (GD). pp. 454–464. Springer (2004)
25. Frishman, Y., Tal, A.: Online dynamic graph drawing. IEEE Transactions on Visualization and Computer Graphics **14**(4), 727–740 (2008)
26. Gansner, E.R., Koren, Y., North, S.: Graph drawing by stress majorization. In: 12th International Symposium on Graph Drawing (GD). pp. 239–250. Springer (2004)
27. Gansner, E.R., Koutsofios, E., North, S.C., Vo, K.P.: A technique for drawing directed graphs. IEEE Transactions on Software Engineering **19**(3), 214–230 (1993)
28. Gilbert, F., Simonetto, P., Zaidi, F., Jourdan, F., Bourqui, R.: Communities and hierarchical structures in dynamic social networks: analysis and visualization. Social Network Analysis and Mining **1**(2), 83–95 (2011)
29. Gorochowski, T.E., di Bernardo, M., Grierson, C.S.: Using aging to visually uncover evolutionary processes on networks. IEEE Transactions on Visualization and Computer Graphics **18**(8), 1343–1352 (2011)
30. Hachul, S., Jünger, M.: Drawing large graphs with a potential-field-based multilevel algorithm. In: 12th International Symposium on Graph Drawing (GD). pp. 285–295. Springer (2004)
31. Hu, Y., Koren, Y.: Extending the spring-electrical model to overcome warping effects. In: 2009 IEEE Pacific Visualization Symposium. pp. 129–136. IEEE (2009)
32. Jünger, M., Mutzel, P.: 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. In: Graph Algorithms and Applications I, pp. 3–27. World Scientific (2002)
33. Kar, G., Madden, B., Gilbert, R.: Heuristic layout algorithms for network management presentation services. IEEE Network **2**(6), 29–36 (1988)
34. Kaufmann, M., Wiese, R.: Maintaining the mental map for circular drawings. In: 10th International Symposium on Graph Drawing (GD). pp. 12–22. Springer (2002)
35. Keller, M.T.: Math genealogy project, https://genealogy.math.ndsu.nodak.edu/
36. Maddison, D., Schulz, K., Lenards, A., Maddison, W.: Tree of life web project, http://tolweb.org/tree/
37. Misue, K., Eades, P., Lai, W., Sugiyama, K.: Layout adjustment and the mental map. Journal of Visual Languages & Computing **6**(2), 183–210 (1995)
38. Moen, S.: Drawing dynamic trees. IEEE Software **7**(4), 21–28 (1990)
39. Nguyen, Q.H.: INKA: an ink-based model of graph visualization. CoRR **abs/1801.07008** (2018)
40. North, S.C.: Incremental layout in dynadag. In: 3rd International Symposium on Graph Drawing (GD). pp. 409–418. Springer (1995)
41. Pavlo, A., Homan, C., Schull, J.: A parent-centered radial layout algorithm for interactive graph visualization and animation. arXiv preprint cs/0606007 (2006)
42. Purchase, H.: Which aesthetic has the greatest effect on human understanding? In: 5th International Symposium on Graph Drawing (GD). pp. 248–261. Springer (1997)
43. Simonetto, P., Archambault, D., Auber, D., Bourqui, R.: Impred: An improved force-directed algorithm that prevents nodes from crossing edges. In: Computer Graphics Forum. vol. 30, pp. 1071–1080. Wiley Online Library (2011)
44. Simonetto, P., Archambault, D., Kobourov, S.: Event-based dynamic graph visualisation. IEEE Transactions on Visualization and Computer Graphics **26**(7), 2373–2386 (2018)

45. Six, J.M., Tollis, I.G.: A framework for circular drawings of networks. In: 7th International Symposium on Graph Drawing (GD). pp. 107–116. Springer (1999)
46. Skambath, M., Tantau, T.: Offline drawing of dynamic trees: Algorithmics and document integration. CoRR **abs/1608.08385** (2016)
47. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems, Man, and Cybernetics **11**(2), 109–125 (1981)
48. Van Eck, N.J., Waltman, L.: Visualizing bibliometric networks. In: Measuring Scholarly Impact, pp. 285–320. Springer (2014)
49. Wiese, R., Eiglsperger, M., Kaufmann, M.: yfiles visualization and automatic layout of graphs. In: Graph Drawing Software, pp. 173–191. Springer (2004)
50. Workman, D., Bernard, M., Pothoven, S.: An incremental editor for dynamic hierarchical drawing of trees. In: International Conference on Computational Science. pp. 986–995. Springer (2004)

# Appendix

**Algorithm 1:** DynaCola(Current tree $T$, Current layout $\Gamma_T$, New edge $e$, Number of iterations $nIters$, Number of subdivision nodes $n_s$)

**Result:** A crossing free layout, maintaining the desired properties
Let $u, v = e$;
Let $u \in T$;
Find a crossing free position for $v$ by random sampling;
Reduce the edge length of $e$ if necessary;
Add $n_s$ subdivision nodes for $e$;
Add the new edges and dummy nodes to $T$;
**for** *each node* $w$ *of* $T$ **do**
  $\Delta w = 0$;
**for** $i \in \{1, 2, \cdots, nIters\}$ **do**
  Compute the collision circle radius;
  **for** *each edge* $e'$ *of* $T$ **do**
    Let $u', v' = e'$;
    $\Delta u', \Delta v' += f_E(e')$;
  **for** *each node* $w$ *of* $T$ **do**
    $\Delta w += F_R(w)$;
    $\Delta w += F_C(w)$;
    $\Delta w += F_G(w)$;
  **for** *each node* $w$ *of* $T$ **do**
    $X_w += \Delta w$;

Fig. 1 shows a small evolving tree with labels, as obtained by the seven algorithms: DynNoSlice, DynaGraph, Dagre, Radial, ImPrEd, DynaCola, and DynaSafe. Fig. 6 shows the same layouts without the labels, allowing us to see the structure a bit better.

**Algorithm 2:** DynaSafe(Current tree $T$, Current layout $\Gamma_T$, New edge $e$, Number of iterations $nIters$)

**Result:** A crossing free layout, maintaining the desired properties
Let $(u, v) = e$;
Let $u \in T$;
Find a crossing free position for $v$ by random sampling;
Reduce the edge length of $e$ if necessary;
Add the new edge to $T$;
**for** *each node* $w$ *of* $T$ **do**
  $\Delta w = 0$;
**for** $i \in \{1, 2, \cdots, nIters\}$ **do**
  **for** *each edge* $e'$ *of* $T$ **do**
    Let $(u', v') = e'$;
    $\Delta u', \Delta v' += f_E(e')$;
  **for** *each node* $w$ *of* $T$ **do**
    $\Delta w += F_R(w)$;
    $\Delta w += F_G(w)$;
  **for** *each node pair* $w, x$ **do**
    $\Delta w, \Delta x += F_S(w, x)$;
  **for** *each node* $w$ *of* $T$ **do**
    Reduce $\Delta w$ by $p\%$ for at most $q$ times;
    Update $X_w$ by $\Delta w$ if no crossing occurs;

Fig. 4 shows the layouts of the tree of life, obtained by DynNoSlice, DynaGraph, Dagre, Radial, ImPrEd, DynaCola and DynaSafe. Fig. 7 shows the layouts of math genealogy tree, obtained by DynNoSlice, DynaGraph, Dagre, Radial, ImPrEd, DynaCola and DynaSafe.

Tab. 6 shows the running times of DynNoSlice, DynaGraph, Dagre, Radial, ImPrEd, DynaCola and DynaSafe.

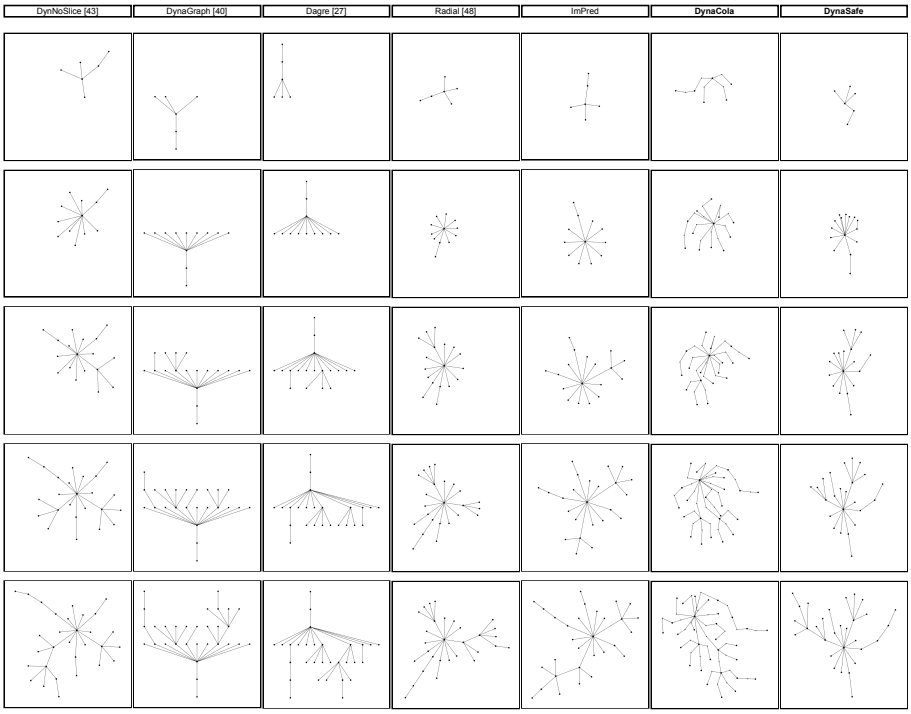



**Fig. 6.** The unlabeled layouts obtained by DynNoSlice, DynaGraph, Dagre, Radial, ImPrEd, DynaCola, and DynaSafe of the same evolving math genealogy tree; each row adds six new nodes.

| Graph | DynNoSlice | DynaGraph | Dagre | Radial | ImPred | DynaCola | DynaSafe |
|---|---|---|---|---|---|---|---|
| 100 MG | 536.86 | 23.01 | 31.39 | 4.01 | 206.65 | 53.27 | 103.61 |
| 200 MG | 1253.38 | 36.02 | 51.04 | 11.42 | 1151.15 | 176.92 | 149.28 |
| 300 MG | 3376.19 | 43.19 | 78.15 | 17.39 | 5451.75 | 201.82 | 267.71 |
| 400 MG | 9382.27 | 57.23 | 96.14 | 27.20 | 9460.37 | 286.27 | 326.16 |
| 500 MG | 24804.39 | 72.77 | 117.87 | 34.93 | 15497.94 | 368.48 | 398.15 |
| 100 TOL | 647.33 | 9.12 | 13.78 | 3.71 | 191.52 | 27.81 | 31.39 |
| 200 TOL | 1297.64 | 13.04 | 19.26 | 9.52 | 1120.00 | 43.01 | 59.74 |
| 300 TOL | 3529.79 | 19.37 | 28.03 | 15.38 | 4281.03 | 67.13 | 87.10 |
| 400 TOL | 9104.15 | 25.28 | 41.03 | 21.91 | 11937.18 | 79.41 | 119.47 |
| 500 TOL | 22212.30 | 36.28 | 52.49 | 28.03 | 22206.56 | 99.30 | 153.89 |

**Table 6.** Running time of different algorithms in seconds.

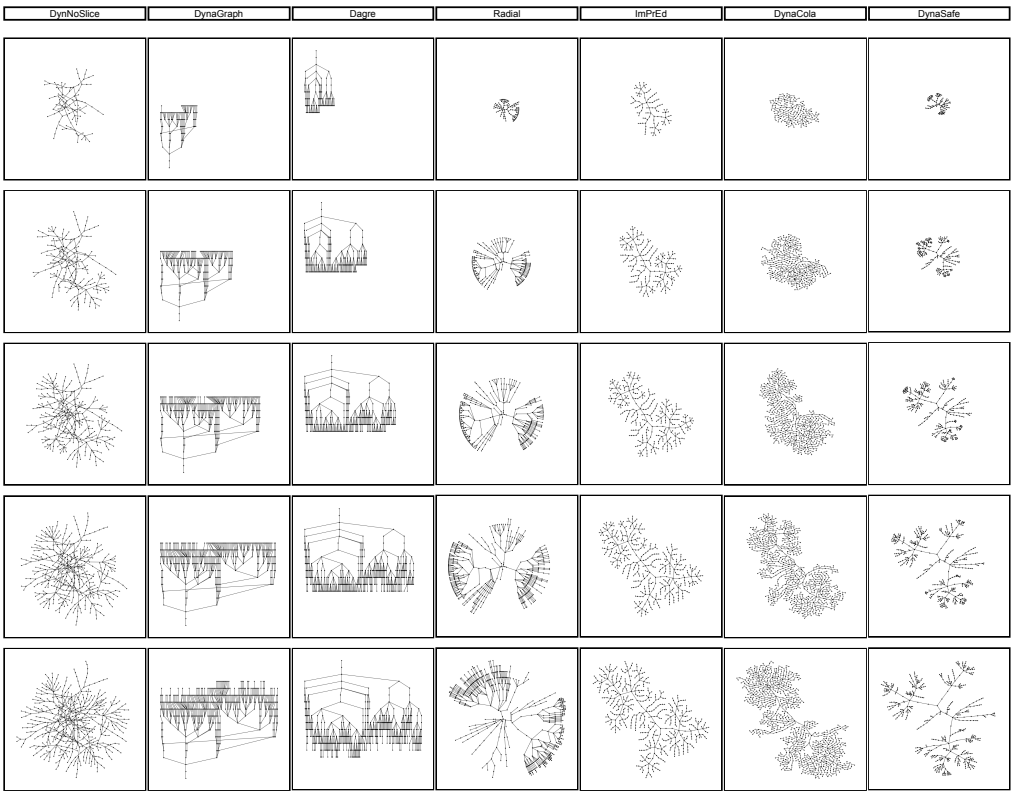



**Fig. 7.** The layouts of different algorithms of the same evolving math genealogy trees at different steps.