Lines: 347

Visualizing Evolving Trees

Kathryn Gray, Mingwei Li, Reyan Ahmed, and Stephen Kobourov

Department of Computer Science, University of Arizona

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, however, 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 four prior approaches (two 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 [27], bibliometric networks [47], software engineering [13], and pandemic modeling [6]; see the survey by Beck *et al.* [9]. 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 [35] and the Mathematics Genealogy Graph [34]. 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, 14, 15, 37], however, most of them have limitations that can impact their usability in this domain. Many represent nodes only as points ignoring labels [14, 15], 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 to drawing hierarchical graphs [26, 39], which does not capture the underlying graph structure well. Force-directed algorithms tend to better capture the underlying graph structure [30], 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.
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 [38]. 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 [36]. 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 [41], and since we work with trees, a layout without edge crossings is possible and desirable. We use two trees extracted from Tree of Life [35] and the Mathematics Genealogy [34] 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 four earlier methods, showing the two proposed methods perform well overall; see Fig. 1.

2 Related Work

Dynamic graph drawing has a long history [10, 45]. It can be divided into 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 [17–19], connecting the same node in consecutive time-slices and optimizing them simultaneously [21–23], providing animation [4], and showing small multiples type visualization [3]. *DynNoSlice* by Simonetto *et al.* [42] 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.* [14,15] and Workman *et al.* [49] 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.

DynaDAG is an online graph drawing method for drawing dynamic directed acyclic graph as hierarchies [39].This method moves nodes between adjacent ranks based on the median sort. It was not specifically developed for trees and may introduce crossings; see Fig. 2a. Other approaches for online dynamic graph

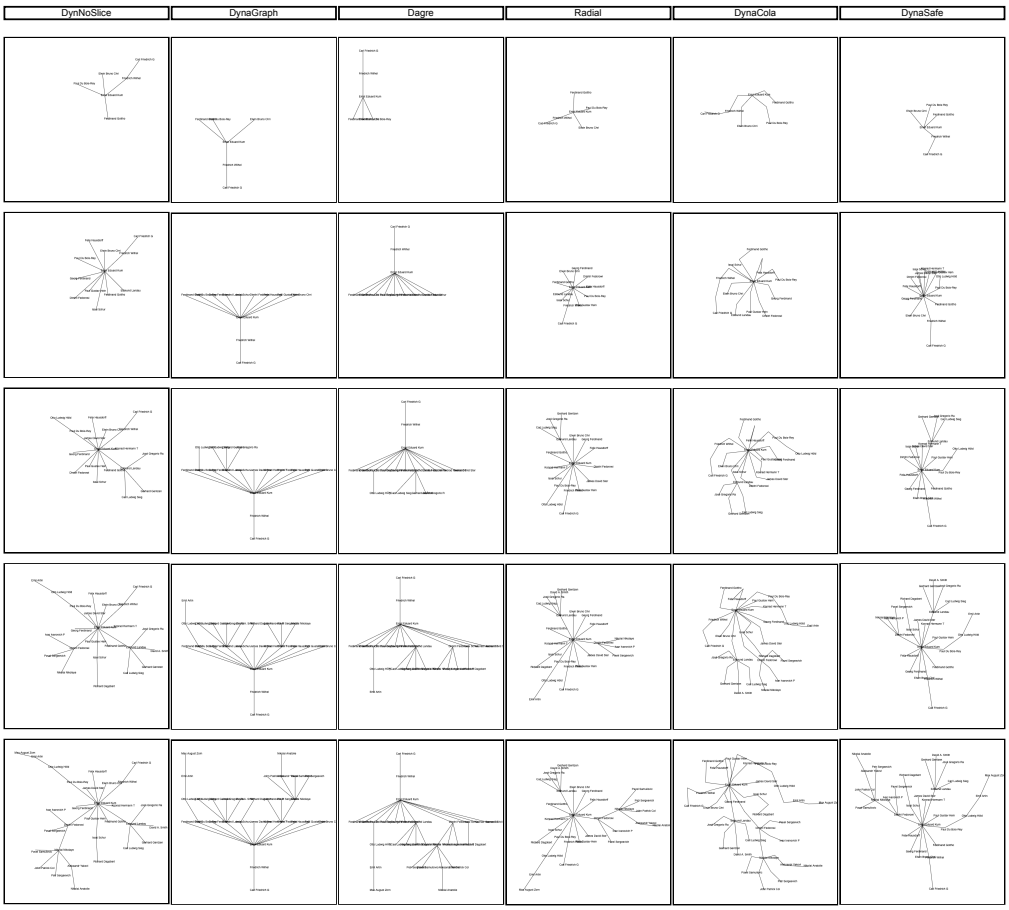


Fig. 1. The layouts obtained by DynNoSlice, DynaGraph, Dagre, Radial, DynaCola and DynaSafe of the same evolving math genealogy tree; each row adds six new nodes.

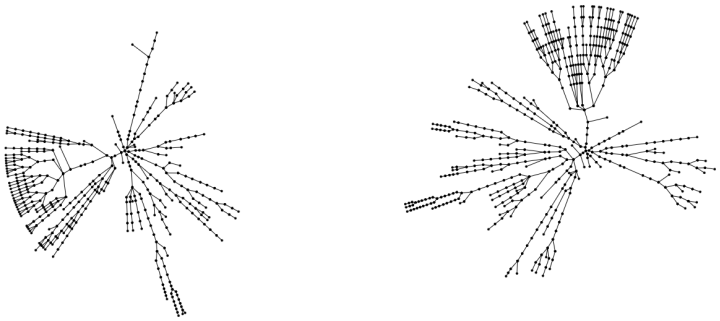
drawing have maintained the horizontal and vertical position of nodes [36], used node aging methods [28], and adapted multilevel approaches [16] (using FM^3 [29]). Online approaches have also been implemented on the GPU [24]. However, these methods do not guarantee crossing free layouts for trees and do not take into account desired edge lengths; see Fig. 2a.

Dagre is a multi-phase algorithm for drawing directed graphs based on [26]. 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) [31]. However, since Dagre draws graphs in a hierarchical structure, the edge lengths may vary arbitrarily; see Fig. 2b.

The *radial* layout implemented in yFiles [48] displays each biconnected component in a circular fashion. Radial layouts were introduced by Kar *et al.* [32]

(a) **(b)**

Fig. 2. (a) A DynaDAG layout of a tree with 100 nodes that introduces some edge crossings. (b) A Dagre layout of a tree with 300 nodes, with large edge length variability. Both trees are extracted from the math genealogy dataset.



(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, a lot of movement was needed to accommodate it.

for static graphs. Dougrusoz *et al.* [20] described an interactive tool for dynamic graph visualization based on the radial layout. Six and Tollis [44] adapted the radial idea to circular drawings of static biconnected graphs and experimentally showed that their layout has fewer edge crossings. Kaufmann *et al.* [33] extended this model to handle dynamic graphs, providing the basis of the yFiles *radial* implementation. Pavlo *et al.* [40] 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 [46] 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 [17–19,42] 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. 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 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. 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 [11].

- 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 [7]. 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).

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.

Algorithm 1: DynaCola(Current tree layout Γ_T, new edge e, $nIters$, Number of subdivision nodes n_s)

```
Result: A crossing-free layout, maintaining the desired properties
Let u, v = e;
Let u ∈ 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 to T;
for each node w of T do
    Δw = 0;
for i ∈ {1, 2, ··· , nIters} do
    Compute the collision circle radius;
    for each edge e' of T do
        Let u', v' = e';
        Δu', Δv' += f_E(e');
    for each node w of T do
        Δw += F_R(w);
        Δw += F_C(w);
        Δw += F_G(w);
for each node w of T do
    X_w += Δw;
```

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. 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 [11]

- 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 4 earlier methods: DynNoSlice, DynaGraph, Dagre, and Radial. 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 [35]. 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

Algorithm 2: DynaSafe(Current tree layout Γ_T, new edge e, $nIters$)

```
Result: A crossing-free layout, maintaining the desired properties
Let u, v = e;
Let u ∈ 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
    Δw = 0;
for i ∈ {1, 2, ··· , nIters} do
    for each edge e' of T do
        Let u', v' = e';
        Δu', Δv' += f_E(e');
    for each node w of T do
        Δw += F_R(w);
        Δw += F_G(w);
    for each node pair w, x do
        Δw, Δx += F_S(w, x);
for each node w of T do
    Reduce Δw by p% for at most q times;
    Update X_w by Δw if no crossing occurs;
```

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 [34]. 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:

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

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 [8]. 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.

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

This formula produces a non-negative number; the ideal value for this measure is 0 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 [43], but since DynNoSlice does not use time slices, it is closer to the measure found in [12]:

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

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 [12, 25, 43]:

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

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 four previous algorithms: DynNoSlice, DynaGraph, Dagre, and Radial. We note that while Dagre and Radial are not specifically designed for dynamic graphs, they can be modified very naturally for this purpose. We can simply pass in the trees at every timestep and redraw.

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 obtained in the layout. We then normalize these values for a fair comparison.

The performance of DynNoSlice depends heavily on two parameters, τ and δ 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 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 [48] implementation. The other algorithms are also used with their default settings.

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 5 and indicate that DynaCola does best. DynNoSlice is 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 6 and indicate that DynNoSlice outperforms the rest of the algorithms. DynaCola is second best and DynaSafe is third. Here, Dagre has the worst performance.

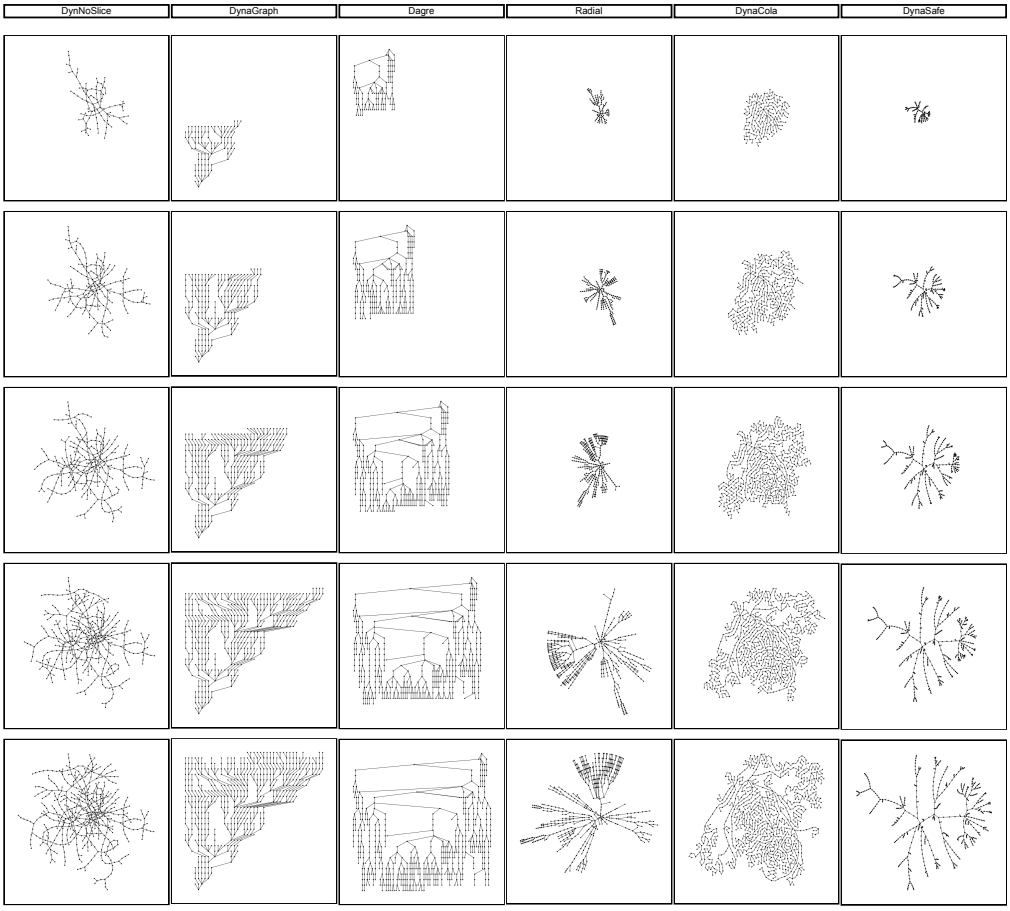


Fig. 4. The layouts of the six methods for the tree of life dataset.

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

Fig. 6. Compactness of math genealogy tree (MG) and the tree of life (TOL).

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

Fig. 5. Desired edge lengths of math genealogy tree (MG) and tree of life (TOL).

Stability: The quantitative results are shown in Table 7 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.

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

Fig. 7. Stability of math genealogy tree (MG) and tree of life (TOL).

Stress: The quantitative results are shown in Table 8. In general, DynaSafe does much better on this measure than any of the rest of them. 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.

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

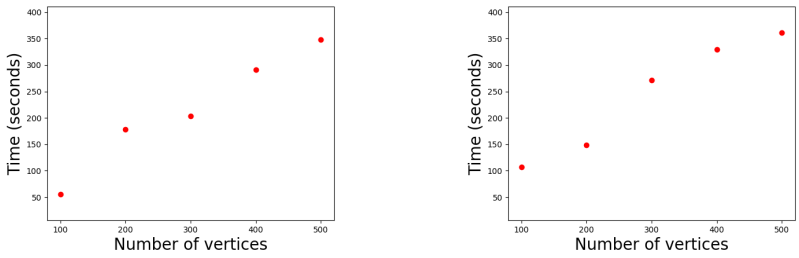
Fig. 8. Stress scores of math genealogy tree (MG) and tree of life (TOL).

Edge Crossings: The quantitative results are shown in Table 9. There are four winners here – the four algorithms that prevent any edge crossings: DynaCola, DynaSafe, Radial, and Dagre. DynNoSlice and Dynagraph do introduce some crossings.

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

Fig. 9. Edge crossings of math genealogy tree (MG) and tree of life (TOL).

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. 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. 10.



(a) DynaCola **(b)** DynaSafe

Fig. 10. Running time w.r.t. number of nodes in the math genealogy tree.

5 Discussion, Limitations, Conclusions and Future Work

While there are many algorithms and tools for drawing static trees, there are only few that can handle dynamic trees well. Among those, there are even fewer that take edge labels into account while also preventing edge crossings. With this in mind, we described two methods for visualizing evolving trees. Fully functional prototypes and videos showing them in action are available online: https://ryngray.github.io/dynamic-trees/index.html. Source code and all experimental data can be found on github: https://github.com/abureyanahmed/evolving_tree.

The two new methods, DynaCola and DynaSafe perform well overall. However, we only used two 500-node trees in the evaluation. We used 5 metrics to quantitatively evaluate the new methods against 4 earlier methods, but it is likely that other earlier methods can be adapted to this setting and might perform better than the ones we used. Further, we did not evaluate how the algorithms behave as the size of the trees grows. Our algorithms have not been optimized for speed and will likely struggle with larger instances.

The two new methods are designed for evolving trees, where growth is the only type of change. A natural question is whether these algorithms can be generalized to the more difficult problem of online dynamic tree visualization.

Acknowledgements

We thank the authors of DynNoSlice, Dynagraph, as well as Felice de Luca and Iqbal Hossain for their assistance with running and tuning algorithms. We also thank yFiles whose radial layout implementation we use in the evaluation, as well Michael Kaufmann and Michael Pfahler for help with the system.

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. 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)
7. Barnes, J., Hut, P.: A hierarchical o(n log n) force-calculation algorithm. Nature **324**(6096), 446–449 (1986)
8. Barth, L., Kobourov, S.G., Pupyrev, S.: Experimental comparison of semantic word clouds. In: Experimental Algorithms. pp. 247–258. Springer (2014)
9. 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)
10. 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)
11. Bostock, M., Ogievetsky, V., Heer, J.: D^3 data-driven documents. IEEE Transactions on Visualization and Computer Graphics **17**(12), 2301–2309 (2011)
12. 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)
13. 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)
14. 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)
15. 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)
16. 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)
17. Diehl, S., Görg, C.: Graphs, they are changing. 10th International Symposium on Graph Drawing (GD) pp. 23–31 (2002)
18. Diehl, S., Görg, C., Kerren, A.: Foresighted graph layout. Technical Report, University of Saarland (2000)
19. Diehl, S., Görg, C., Kerren, A.: Preserving the mental map using foresighted layout. Proceedings of Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym) (2001)

20. 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)
21. 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)
22. 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)
23. 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)
24. Frishman, Y., Tal, A.: Online dynamic graph drawing. IEEE Transactions on Visualization and Computer Graphics **14**(4), 727–740 (2008)
25. 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)
26. 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)
27. 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)
28. 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)
29. 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)
30. Hu, Y., Koren, Y.: Extending the spring-electrical model to overcome warping effects. In: 2009 IEEE Pacific Visualization Symposium. pp. 129–136. IEEE (2009)
31. 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)
32. Kar, G., Madden, B., Gilbert, R.: Heuristic layout algorithms for network management presentation services. IEEE Network **2**(6), 29–36 (1988)
33. Kaufmann, M., Wiese, R.: Maintaining the mental map for circular drawings. In: 10th International Symposium on Graph Drawing (GD). pp. 12–22. Springer (2002)
34. Keller, M.T.: Math genealogy project, https://genealogy.math.ndsu.nodak.edu/
35. Maddison, D., Schulz, K., Lenards, A., Maddison, W.: Tree of life web project, http://tolweb.org/tree/
36. Misue, K., Eades, P., Lai, W., Sugiyama, K.: Layout adjustment and the mental map. Journal of Visual Languages & Computing **6**(2), 183–210 (1995)
37. Moen, S.: Drawing dynamic trees. IEEE Software **7**(4), 21–28 (1990)
38. Nguyen, Q.H.: INKA: an ink-based model of graph visualization. CoRR **abs/1801.07008** (2018)
39. North, S.C.: Incremental layout in dynadag. In: 3rd International Symposium on Graph Drawing (GD). pp. 409–418. Springer (1995)
40. Pavlo, A., Homan, C., Schull, J.: A parent-centered radial layout algorithm for interactive graph visualization and animation. arXiv preprint cs/0606007 (2006)

41. Purchase, H.: Which aesthetic has the greatest effect on human understanding? In: 5th International Symposium on Graph Drawing (GD). pp. 248–261. Springer (1997)
42. Simonetto, P., Archambault, D., Kobourov, S.: Event-based dynamic graph visualisation. IEEE Transactions on Visualization and Computer Graphics **26**(7), 2373–2386 (2018)
43. Simonetto, P., Archambault, D., Kobourov, S.G.: Drawing dynamic graphs without timeslices. CoRR **abs/1709.00372** (2017)
44. 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)
45. Skambath, M., Tantau, T.: Offline drawing of dynamic trees: Algorithmics and document integration. CoRR **abs/1608.08385** (2016)
46. 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)
47. Van Eck, N.J., Waltman, L.: Visualizing bibliometric networks. In: Measuring Scholarly Impact, pp. 285–320. Springer (2014)
48. Wiese, R., Eiglsperger, M., Kaufmann, M.: yfiles visualization and automatic layout of graphs. In: Graph Drawing Software, pp. 173–191. Springer (2004)
49. 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

Fig. **??** shows a small evolving tree with labels, as obtained by the six algorithms: DynNoSlice, DynaGraph, Dagre, Radial, DynaCola and DynaSafe. Fig. 11 shows the same layouts without the labels, allowing us to see the structure a bit better.

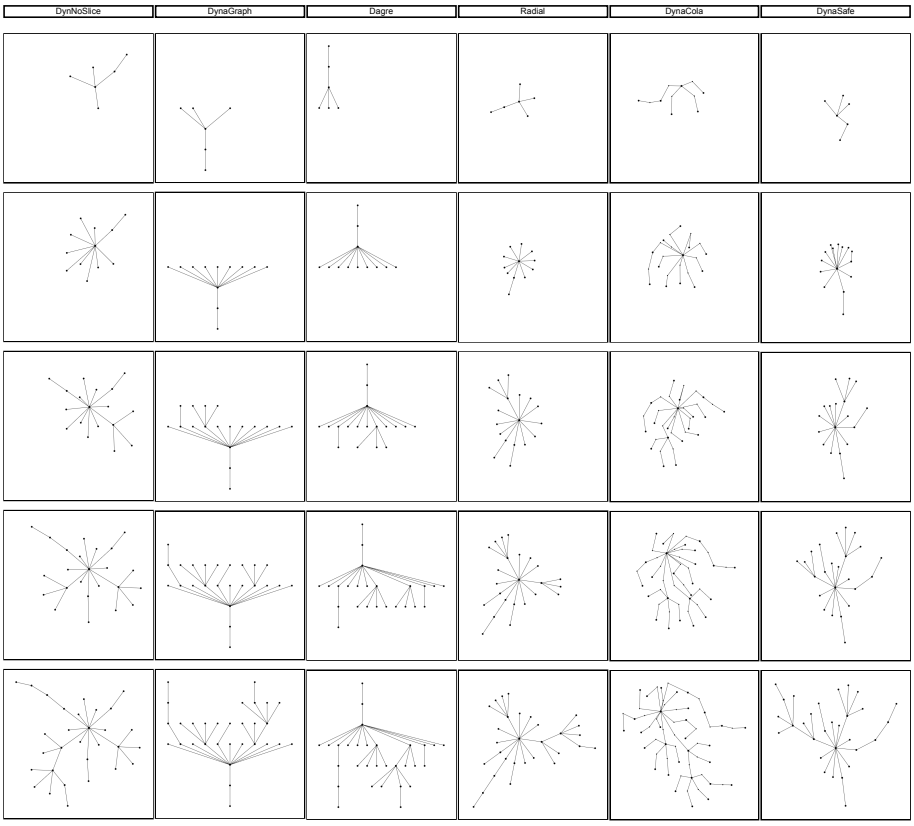


Fig. 11. The layouts of different algorithms of the same evolving trees at different steps.

The graph below shows a comparison of the different algorithms on the math genealogy graph. This shows the timesteps with 100,200,300,400, and 500 nodes.

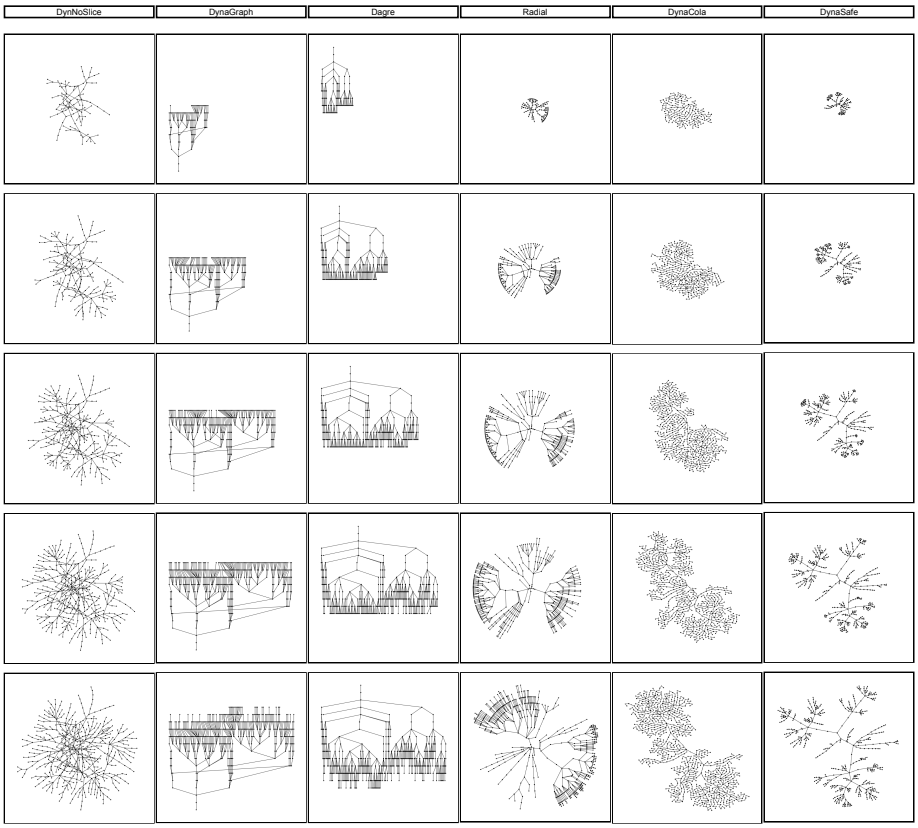


Fig. 12. The layouts of different algorithms of the same evolving math genealogy trees at different steps.