Visualization of Barrier Tree Sequences Revisited

Christian Heine[1], Gerik Scheuermann[1], Christoph Flamm[2], Ivo L. Hofacker[2], and Peter F. Stadler[3]

[1] Image and Signal Processing Group, Department of Computer Science, University of Leipzig, {heine,scheuermann}@informatik.uni-leipzig.de
[2] Department of Theoretical Chemistry and Structural Biology, University of Vienna, {xtof,ivo}@tbi.univie.ac.at
[3] Bioinformatics Group, Department of Computer Science, University of Leipzig, studla@bioinf.uni-leipzig.de

Summary. The increasing complexity of models for prediction of the native spatial structure of RNA molecules requires visualization methods that help to analyze and understand the models and their predictions. This paper improves the visualization method for sequences of barrier trees previously published by the authors. The barrier trees of these sequences are rough topological simplifications of changing folding landscapes – energy landscapes in which kinetic folding takes place. The folding landscapes themselves are generated for RNA molecules where the number of nucleotides increases. Successive landscapes are thus correlated and so are the corresponding barrier trees. The landscape sequence is visualized by an animation of a barrier tree that changes with time.

The animation is created by an adaption of the foresight layout with tolerance algorithm for dynamic graph layout problems. Since it is very general, the main ideas for the adaption are presented: construction and layout of a supergraph, and how to build the final animation from its layout. Our previous suggestions for heuristics lead to visually unpleasing results for some datasets and, generally, suffered from a poor usage of available screen space. We will present some new heuristics that improve the readability of the final animation.

1 Introduction

1.1 Biological Background

Ribonucleic acid (RNA) is a linear biopolymer, i.e. a chain of covalently connected units (nucleotides) of which there are four types: adenine (A), guanine (G), cytosine (C), and uracil (U). RNA molecules play an important role in many biological contexts, e.g. protein synthesis. The biological function of an RNA molecule is determined predominantly by its spatial structure which in turn is determined by the sequence of nucleotides. When an RNA molecule is produced in the cell, it folds back to form double helical regions consisting of paired nucleotides. The list of helices or (equivalently) of base pairs is known as the *secondary structure* of the RNA molecule. Since helices stabilize the structure while the intervening unpaired loops are destabilizing, each secondary structure can be assigned a free energy equivalent to the energy released when the molecule folds. To a large extent, the secondary structure already determines the function of RNA.

Various methods have been proposed to explain and predict the structures of RNA molecules. Typically, one considers the structure with the lowest free energy, i.e. the one for which the folding process that starts from the completely unfolded state releases the maximum amount of energy. This structure is the most stable one, and according to the laws of statistical mechanics, the one that is most frequently attained in thermodynamic equilibrium. The folding process itself can, however, take a long time so that the equilibrium state that will be reached after an infinite waiting time may not be biologically relevant. Instead, the folding process may pause in metastable structures from which it is hard to escape due to high energy barriers. The folding process of an RNA molecule can be modeled as a Markov process whose states are the individual secondary structures [CHS96]. Transitions are allowed only between "neighboring configurations", i.e. those that differ by only one base pair [FFHS00], and transition rates are proportional to $\exp(\Delta E/RT)$, where ΔE is the difference in energy, T is the ambient temperature, and R is a constant. In practice, however, the transition matrix is much too large to solve the resulting master equation directly.

A refined model transforms the configuration space into a large graph, whose vertices are secondary structures and whose edges connect neighboring structures. The neighbor graph along with the energy specific to each configuration can be imagined as a discrete energy landscape. A folding or refolding process can then be described by a path in the graph or a walk in the energy landscape. For each such path there exists one structure of maximal free energy, the *maximum* of the path. The *barrier* between two configurations is the smallest maximum of all paths between the two configurations. If a structure refolds, it has to overcome at least this energy barrier. These barriers partition the graph into "basins" that are centered around local energy minima (secondary structures of which all neighbors are less stable). An approximate model is now obtained by considering the basins as effective states of the RNA molecules. Transition rates between basins can be derived from the more detailed model under the assumption that the folding process is nearly equilibrated locally within each basin [WSSF^{+}04].

The relevant information can now be stored in the so-called *barrier tree* T of the landscape. The leaves of T correspond to the local minima of the energy landscape together with their basins of attraction, while inner vertices represent the barriers (also called saddle-points) between the basins. Figure 1 shows an example of a barrier tree for a very simple landscape. This example is just for illustrative purposes; we consider mainly landscapes where individual

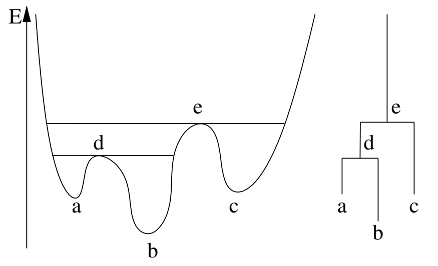

Fig. 1. A very simple landscape and barrier tree
In contrast to normal trees, each vertex of a barrier is drawn at a height that reflects the free energy of the folding configuration it represents. To determine the energy barrier between two local minima, one has to find the barrier tree vertex that has both leaves representing the local minima as descendants and the greatest topological distance to the root of the tree.

points do have a high and varying number of neighbors, making the landscape a high dimensional object. Barrier trees are constructed by successively "flooding" the basins of the landscape. A barrier is found at the point where the lakes of two basins would join. These two joined basins are considered to be one when the "flooding" is continued. See Flamm et al. [FHSW02] for a detailed description.

In reality, however, RNA molecules are not "born" as a whole. Rather, they are "transcribed" nucleotide by nucleotide from their DNA template, so that the molecule is still growing while it already starts to fold [MM04]. The structures that are formed are thus dependent upon the relative rates of folding and transcription. Similar effects are observed when an RNA molecule travels through a narrow pore, where it must unfold on one side and refolding on the other [GBH04]. Again the kinetics of folding is coupled to the speed with which the molecule is pulled through the pore. Instead of single static energy landscape, we thus have to deal with a situation where the energy landscape, and hence the rules of folding, changes with each step of the second dynamical process. Since the latter proceeds in small steps, it only causes moderate changes in the energy landscape. Thus, there is a natural correspondence between a local energy minimum x before and a (unique) local minimum x' after a step of the second dynamics: Structure x is modified to some structure x^* i.e. by appending a single unpaired nucleotide. Then x^* relaxes to the local minimum x' to whose basin it belongs to. Note that multiple local minima can map to the same local minimum in the next step, and that local minima might arise that are not mapped from any local minimum of the previous step.

From the biophysical point of view, the problem is thus to understand the dynamics of folding combined with another process such as transcription or pore traversal. As in the static case, this can be done by approximating the folding energy landscape at each step by its barrier tree. The second dynamics is then represented by transitions between corresponding local minima. While the folding process in the static case is relatively easily interpreted as a movement on the barrier tree, we now have to consider a movement on a series of barrier trees whose vertices are connected in a specific way.

In numerical simulations, one observes, that for some RNAs the fraction of folding trajectories that reach the ground state of a certain fully grown chain depends in a non-trivial way on the relative speed of transcription. Both for very slow and very fast transcription the molecule reaches the ground state quickly, while in an intermediate regime most of the trajectories become trapped in a metastable, very different, secondary structure. In order to understand this phenomenon it is necessary to compare the trajectories in the barrier tree series and to pinpoint the step(s) in which escape from local minima occurs at the same time scales as chain elongation. The same type of questions naturally arise in other settings where the folding energy changes, i.e., whenever the temperature or salt concentration changes.

1.2 Visualization Problem

Without an appropriate visualization tool it is virtually impossible to find the time-steps and transition at which time-scale difference have a drastic effect, as there is little or no *a priori* coherence between the layouts of the individual barrier trees in a series. It is thus very tedious to actually follow a trajectory through a series and to determine the likely transitions. The mapping of local minima, however provides information that, as we shall see, can be utilized to enhance the coherence of adjacent trees in a series.

The barrier trees thus share common information that should be presented accordingly, i.e. it should not attract more attention than the parts that differ. Instead of visualizing a sequence of barrier trees that have some redundancy, one can also say that there is just one barrier tree that changes with time in a way that the barrier trees of the sequence are snapshots of the dynamic tree at certain points of time. In this work, we will thus view this problem as a dynamic graph drawing problem. As an abstraction, we define the problem as follows: *Given a sequence of barrier trees and leaf mappings, where leaves of one tree are mapped on leaves of the following tree, determine the layout of all trees such that in a presentation the mental map is retained.*

1.3 Algorithm Outline

To solve the visualization problem, our algorithm is split in several parts, which we will decribe in sections 3 to 5. Given the barrier trees $T_i = (V_i, E_i, e_i)$, ($e_i : V_i \rightarrow R$ is a function that gives the energy of each vertex) and the leaf mappings $f_i : V_{i-1} \rightarrow V_i$ between them, we first find equivalent vertices. These vertices are then arranged in an order that minimizes an objective function which is mainly determined by the number of visible edge crossings for the whole sequence at presentation time. We use simulated annealing to determine this order. Given this order, we directly derive the layout of the single trees that make up the barrier tree sequence and present them in an animation with transitions that help to communicate the changes.

2 Related Work

Drawing a graph is the process of transforming topological properties of the graph to geometric objects in a graphical representation. This process is mostly determined by the generation of a layout for that graph, that *places* vertices in a vector space and *routes* edges to connect the vertices. The layout of a graph has properties that can be measured with certain cost functions, e.g., area of the layout, number of edge crossings, distribution of vertices and edges, congruency of isomorphic structures, etc. To make visually pleasing drawings, esthetic criteria have been defined. Such criteria often demand maximizing or minimizing one of the cost functions. As not all esthetic criteria can be obeyed simultaneously, a layout algorithm generally makes a trade-off between them. The field of static graph layout creation has been intensively studied in the past decades. There exist good overviews for this topic ([dBETT94, HMM00, Tam99]).

The first attempts toward dynamic graph drawing were very specific. Moen [Moe90] presents an algorithm that shows a part of an ordered tree. Although the tree itself stays the same, the selected subset may change through replacement of subtrees by leaves and vice versa. Cohen et al. [CdBT$^+$92] gives detailed algorithms and data structures for a number of dynamic graph classes. These allow visualizing popular data structures, e.g. AVL- Trees, and adjusting the layout of a graph, if it is being edited or browsed. Both approaches share a common motivation: they reduce the computation time of the layout by reusing information about the previous layout. This has the side effect of making the layout of the changed graph similar to the unchanged, but accumulation of many elementary changes can result in an esthetically unpleasing drawing.

North [Nor95] measures the quality of an algorithm to make good dynamic drawings based on *incremental* or *dynamic stability*, i.e. the property of an algorithm to produce very similar layouts for graphs that differ only slightly. He applies his concepts to the drawing of dynamic directed acyclic graphs. Misue et al. [MELS95] introduce the concept of *mental distance*. It formally describes the difference of two layouts and can be used to measure the perceived stability of a dynamic graph layout. They define the esthetic criterion "preserving the mental map" for any dynamic graph drawing problem, and refine it to three models. In the *orthogonal ordering* the left-to-right, and up-down order of vertices stays the same. *Proximity relations* are preserved, if the relative distances of vertices and edges do not change. The *topology* is preserved, if vertices and groups of vertices of one region stay in that region. The *mental distance* of two layouts is the number of times or the amount by which a rule is broken. Frishman and Tal [FT04] present an algorithm that draws dynamic clustered general graphs using an incremental force directed method. Their algorithm generally preserves the mental map by reusing the earlier layout, but improves the layout slightly, if a static graph drawing esthetic criteria is not met any more. They recently generalized their method to unclustered graphs ([FT07]).

If the layout process cannot be formulated to minimize the mental distance between successive layouts, a local transition or morphing of the layouts has to take place. Friedrich and Eades [FE02] describe a method to make sure that the transition preserves the mental map. To do that, an affine transformation that registers both layouts is determined and performed. Using a force-directed approach, vertices are moved to their final positions while avoiding occlusions and other visual artifacts linear interpolation would bring forth. Fortunately, our algorithm produces layouts that are stable enough not to require these forms of transition.

Erten et al. [EHK$^+$03] describe a method to layout general dynamic graphs using a force-directed method. Vertices of the evolving graph that are equivalent are connected by virtual springs that contract in the force-directed method. As a result, vertices referring to the same instance at different times are positioned closely together. This ensures a good stability of the dynamic layout. We do not use this general approach, because we feel that the final animation should at least resemble the look and feel of barrier trees.

Diehl and Görg [DG02] propose a general scheme to layout dynamic graphs when all graphs of the sequence are known prior to layout creation. This scheme is independent of the class of the graphs and the layout algorithm used. Their *Foresight Layout with Tolerance* algorithm makes a trade-off between static and dynamic graph drawing esthetic criteria based on a tolerance parameter. In a first phase a *supergraph* is constructed that contains all graphs of the sequence as subgraphs. Then the layout of this (static) supergraph is determined and used as a blueprint for the layout of the subgraphs. The layout of the subgraphs can be further improved with respect to static graph drawing esthetic criteria, but its mental distance may not differ by more than the tolerance parameter from the blueprint layout. Presentation of the sequence can be done using morphing geometry information between the single subgraphs. Görg et al. [GBPD04] further improve the scheme with the notion of the *importance* of a vertex or edge. This importance is a measure for the number of times a vertex or edge is present in the graph sequence and is used to improve the visual quality of the layouts.

A similar idea is presented by Gaertler and Wagner [GW05]. Instead of an animation, a $2\frac{1}{2}$D visualization, i.e. a 3D view of a stack of static 2D layouts–each showing the graph at a certain point of time–is generated. Brandes et al. [BDS03] also use $2\frac{1}{2}$D visualization to show a set of similar metabolic pathways. They create the layouts of the acyclic directed graphs representing the pathways using a layout of an union of all graphs, and also determine the optimal ordering of layouts. Both approaches share the notion of the supergraph, local adjustments like in the *Foresight Layout with Tolerance* algorithm are not performed. Dwyer and Schreiber [DS04] also use $2\frac{1}{2}$D to visualize a set of similar phylogenetic trees. Phylogenetic trees are very similar in structure to barrier trees. In contrast to the other two approaches instead of a supergraph only a *minimal leaf ordering* is determined. This neglects the identification of equivalent inner vertices, which becomes necessary, if transitions are to be shown between keyframes. It also requires each inner vertex to have exactly two children, a property which barrier trees do not have in general.

In this work we adapted the *Foresight Layout with Tolerance* algorithm. Since it is very general, we optimized each of the phases to fit our dynamic barrier tree application.

The layouts of the subgraphs that is generated from the supergraph layout can also be used in a $2\frac{1}{2}$D visualization. However, we found this to be inappropriate, because the barrier tree sequences under consideration were highly dynamic. In our datasets we observed that almost any tree at time t has nearly nothing in common with the tree at time $t + 5$. A $2\frac{1}{2}$D visualization would therefore exhibit much visual clutter. Also, the energy of a vertex, and thus its vertical position, can change between subgraphs. In a $2\frac{1}{2}$D visualization one would have to indicate such events with edges between slices, we found it more natural to indicate that in an animation with a movement of the vertex. In general, we think that the animation of transitions between subgraph layouts can be efficiently used to communicate the changes the barrier tree topology to the user.

3 Constructing the Equivalence Classes

The first step in the *Foresight Layout with Tolerance Algorithm* is to construct a supergraph. In the general case, this would be the union of all graphs of the sequence. Unfortunately, this works well only if the supergraph contains all the information of the subgraphs afterward. But barrier trees have additional information per vertex that is used for layout, i.e. their energy. Since a vertex of the supergraph may represent multiple vertices of the tree sequence and each of these vertices may have a different energy, a supergraph vertex may not have a single energy value. We found no useful solution to incorporate this information into the supergraph, so in earlier work [HSF$^+$06] we simply ignored this information and constructed the supergraph nonetheless. While it can be shown that this may lead to suboptimal results, especially during edge crossing minimization, we found that hardly ever a problem for the datasets we considered, and used preprocessing to minimize the errors.

When we considered larger datasets, we observed that using too much preprocessing on them deleted much information, but construction and layout of the supergraph for the unprocessed data gave esthetically unpleasing results. For this work, we decided to take an alternative approach and do not use all of the output of the supergraph construction but put all barrier tree information directly into the layout process. The identification of equivalent barriers, however, is still required and to that end the supergraph algorithm in [HSF$^+$06] can be used. For brevity we will not repeat the rather lengthy algorithm here, instead we refer to its original publication. From the output of the algorithm we ignore the structure, i.e. , we ignore the edges of the supergraph G. The vertices of G are our equivalence classes and the function k, which maps from each barrier tree vertex to a supergraph vertex, becomes a function which maps each barrier tree vertex to its equivalence class.

4 Layout

4.1 Supergraph Layout

We use the barrier trees directly as an input for the supergraph layout. We try to find an order σ of the equivalence classes such that the sum of all edge crossings in all trees is minimized, if the barrier tree vertices were drawn using this order as the horizontal order.

$$\sigma = \underset{O}{argmin} \sum_{i=1}^{N} (\alpha \cdot crossings(T_i, O_{V_i}) + \beta \cdot localorder(T_i, O_{V_i}))$$

where

- $T_i = (V_i, E_i, e_i)$ is the i-th tree in the sequence,
- $G = (V, E)$ is the supergraph of the tree sequence according to [HSF$^+$06],
- V is the set of equivalence classes,
- $k : \bigcup_{i=1}^{N} V_i \to V$ maps each tree vertex to its equivalence class,
- $O \subset V \times V$ is an ordering relation,
- O_{V_i} is that ordering relation restricted to V_i and satisfies $(u, v) \in O_{V_i}$, if and only if $(k(u), k(v)) \in O$ for all $u, v \in V_i$,
- $crossings(T_i, O_{V_i})$ denotes the number of edge crossings if the tree T_i was drawn with the horizontal order of the vertices given by O_{V_i},
- $localorder(T_i, O_{V_i})$ names approximately the number of times a parent vertex is not drawn between its children, and
- α, β constants, which we set to 1 and 5 respectively.

At first we minimzed the above function only considering minimizing the number of edge crossings and used simulated annealing [KGV83] to that end. We were surprised that it is possible to draw the simplest sequence (ATT) with a total of 27 edge crossings for the whole sequence. We were quickly disappointed by the images themselves, as it was apparent that we neglected to encourage the father of two vertices to be drawn between them. Because of that, the use of our orthogonal drawing style resulted in hardly readable images. So we added the second term to our objective function to avoid this particular effect. It accumulates the difference of the number of vertices that are drawn to the left of their parent and the number of vertices that are drawn to the right of their parent for each vertex. If each vertex is always between its two successors, the contribution of this term to the objective function is always zero. We experimentally determined $\alpha = 1$ and $\beta = 5$ to give good final layouts. It roughly means that we rather allow 5 edge crossings than one parent that is not between its children.

There are multiple possibilities to implement the simulated annealing strategy for this particular objective function. We tested several of them, and found the following to behave the best. We start with a random order of equivalence classes and iteratively improve this order. At each iteration, we pick a random equivalence class and insert it at a random position between two other equivalence classes. Then we re-evaluate the objective function for all trees and compare it to the old value. If we improved, we keep the new order, otherwise we only keep it with the probability

$$p = \frac{1}{1 + exp(\Delta C\, T_t^{-1})} \qquad\qquad T_t = \frac{n_t - t}{t}$$

with ΔC being the cost increase and T_t being a temperature which decreases linearly with each iteration t. We stop the process after a fixed number of iterations n_t.

Instead of recalculating the total number of edge crossings, we just calculate ΔC by considering only adjacent and incident edges on all vertices v with $h(v)$ being the equivalence class currently moved. We can do this similarly for the *localorder* term of the objective function. This greatly decreases the time per iteration and makes the process very fast.

In our previous work we computed the layout of the supergraph using the *dot* algorithm by Gansner et al. [GKNV93]. In this algorithm most of the time is spend minimizing edge crossings in a repeated heuristic two layer edge crossing minimization which had a time complexity of $O(N^4)$, where N is the maximum number of vertices on one layer. Although the algorithm seldom runs in that order for real world examples, it takes a very long time to find the minimal number of edge crossings for our barrier tree sequences. Not only because we observed that there was at least one layer where one eighth of all supergraph vertices resided in, but also because the swapping of vertices often did not change the number of edge crossings directly, but a few iterations later might have allowed improvements.

Our new method has much faster iterations because the number of operations per iteration is in the order of

$$O\left(\sum_{i\in\{1,\dots,N\}} deg(T_i)|E_i|\right) = O\left(\sum_{i\in\{1,\dots,N\}} deg(T_i)|V_i|\right)$$

where $deg(T)$ is the degree of T, i.e. the maximum number of incident and adjacent edges on any vertex v of T. So one iteration roughly scales linearly with the total number of vertices of the whole sequence, as the degree of our trees is 2 or 3 in almost all cases, i.e. a very small constant. So one iteration lies in $O(N)$, but we require many more iterations to achieve the same quality improvement of one iteration of the *dot* algorithm.

4.2 Tree Layout

Coordinate assignment of tree vertices is done for each tree separately, respecting the ordering relation generated in the layout phase. This constraint preserves the mental map, specifically the *orthogonal ordering*. Initially the horizontal position of a tree vertex v is directly gained from the number of equivalence classes smaller than $h(v)$ with respect to the global ordering relation O. The vertical position of v directly reflects its energy.

After the vertices have been positioned, edges must be routed. For simplicity each tree edges consist of just one horizontal and one vertical line segment that directly connect the two adjacent vertices. In general, it is not always possible to draw the trees without edge crossings. We sacrificed this property for the preservation of the mental map. Drawing the edges as orthogonal line segments conforms to the style, barrier trees are usually drawn. We also found that a straight line drawing does not necessarily reduce the number of edge crossings and additionally makes tracing the edges harder than in an orthogonal drawing.

Positioning each tree separately allows us to locally improve the layout of the subgraphs. This corresponds to the third phase of the *Foresight Layout with Tolerance* algorithm. It is trivially possible to generate the horizontal position of a Tree vertex v from the number of vertices of the same tree that are smaller than v with respect to O_{V_i}. This would make a better visual impression, if the keyframes were studied by themselves, but it destroys a lot of the mental map, so we do not use this possibility for our animations.

5 Animation

After the layout for each tree has been generated, the single trees could be presented using the generated layout. In practice, there can be quite a number of changes between consecutive trees. Vertices and edges may appear or disappear, and whole subtrees can change the energy of their vertices. We created methods to make the transition smooth and to indicate the type of change. Vertices that experience a change of energy are moved accordingly in the drawing area using linear interpolation of the coordinates. Barriers that appear or disappear are presented using *blending*. Edges are modified based on the changes of their adjacent vertices. Subtrees that are created or merged "grow" out of or into the vertices, where they are created or merged into, again using linear interpolation of their coordinates.

Usually the huge number of changes would require each change to be visualized separately. In our proof-of-concept implementation, all changes are shown simultaneously using the following scheme: Each transition is given a time interval $[t_i, t_i + \Delta t)$. Vertices that change their energy are moved during $[t_i + \frac{3}{8}\Delta t, t_i + \frac{7}{8}\Delta t)$. Subtrees that grow into a vertex because of merging are scaled during $[t_i + \frac{2}{8}\Delta t, t_i + \frac{5}{8}\Delta t)$, subtrees that grow out of a vertex, do so during $[t_i + \frac{5}{8}\Delta t, t_i + \frac{8}{8}\Delta t)$. Fading out of barriers is done during $[t_i + \frac{2}{8}\Delta t, t_i + \frac{6}{8}\Delta t)$ and fading in takes place during $[t_i + \frac{4}{8}\Delta t, t_i + \frac{8}{8}\Delta t)$. The remaining interval $[t_i, t_i + \frac{2}{8}\Delta t)$ is used for a static presentation of tree T_i. The segments overlap intentionally. In the dataset we observed we found that using non-overlapping sections resulted in large parts of the tree simply disappear and appear and destroy the mental map of the user.

6 Highlighting

One common question for a domain expert that analyzes the barrier tree sequence is "which of two given structures is the winner", i.e., which one is more probable to be found in nature. It is typically found when the folding process starts in one part of the energy landscape and, later on, a new part of the energy landscape is created which is separated by a very high barrier from the rest. Regardless of how optimal the local minima of the new part of the landscape are, it is unlikely that the molecule will fold into one of them, because the barrier is too high and the probability that it will be overcome is very low on the timescale for folding reactions.

Using a simple technique, some elements of the animation can be highlighted to emphasize such observations. We split the last tree at the root and look for the leaf of lowest energy in the left subtree and the leaf of lowest energy in the right subtree. The first one is marked blue and the later one red. When the animation is shown, predecessors of these two leaves will be drawn with the appropriate color. The predecessors are given by the leaf mapping, i.e., they are local minima that will refold into the two final configurations during the process. We also found that drawing the path from the root to the actual leaf with the highlight color is visually more attracting that just coloring the leaf and its one adjacent edge.

7 Results

We evaluated our improved algorithm on three datasets. The ATT dataset consists of 20 barrier trees, with at most 25 leaves per tree and a total of 894 vertices in all trees. It represents a small RNA molecule, with sequence length growing from 40 to 74 nucleotides with varying step size. Figure 2 shows the keyframes for this dataset. The LEPTO dataset consists of 47 barrier trees, with a maximum of 50 leaves per tree and a total of 3727 vertices in all trees. The sequence length of the molecule increases from 10 to 56 nucleotides. The largest example, the HOK dataset, consists of 65 trees with a maximum of 100 leaves and a total of 8635 vertices. The sequence length grows from 10 to 74 nucleotides. The inner vertices of all trees of these datasets satisfy $odeg(v) = 2$, i.e., all inner vertices have exactly two children. All datasets present rather short RNA molecules.

We found our new supergraph layout, which effectively only determines an optimal vertex order for the supergraph vertices, to perform very well. We feared that most of the random permutations do not improve the tree layout at all or only slowly. Indeed, we require a greater number of iterations until the images produced were readable. Whereas the old algorithm used 1000 layer sweep iterations the new algorithm uses at least 100.000 iterations to generate readable animations. But because each iteration requires much less time, the new implementation is still faster. On an AMD Opteron with 2.0GHz, the old algorithm with 1000 iterations required approximately 37, 3462, and 16790 seconds for the ATT, LEPTO, and HOK dataset with 381, 1531, and 3793 equivalence classes respectively. The new method using 1.000.000 iterations required approximately 27, 79, and 182 seconds respectively.

The visual output of the two radically different methods is not directly comparable. There is one thing directly observable for all datasets. The new algorithm distributes vertices more evenly across the drawing area. But this is rather a nice side effect of the method and was not originally intended. While the ATT sequence does not improve much visually if compared to the old method, it benefits from the reduced computation time. The visual quality of the LEPTO and HOK sequences improved greatly, because now all edge crossings are accounted for.

8 Conclusion and Future Work

We have shown that it is possible to generate readable layouts for sequences of barrier trees using the *Foresight Layout with Tolerance* algorithm. We also showed, that construction of a supergraph may sometimes lead to suboptimal results if the supergraph does not use all information from the graphs it is constructed for. Our naïve implementation of a combination of supergraph construction and layout clearly outperformed the version where these two were separate, both quality and runtime-complexity wise. The number of iterations needed for a given dataset seems to scale with its size, but we are yet uncertain exactly how. We are currently looking for methods that automatically determine the optimal number of iterations.

The layout of the single trees may be combined with additional information. The simulation of the folding process during the growing of the molecule under various temperatures and growing rates results in distribution functions for local minima. Because the animation of the barrier trees preserves the orthogonal ordering, annotating the barrier tree leaves with the density of the corresponding structure configurations preserves the mental map for the annotations. The change in the densities can be additionally indicated by a flow of liquid along the tree edges. Methods that combine tree layout and additional information are currently investigated.

As a transition from one tree to the next consists of many elementary operations, instead of showing them simultaneously, it might be better to break the leaf mappings in elementary operations and show them in sequence.

The constructed supergraph is a static visualization of the whole sequence, and presentation forms other than an animation, may be investigated. One idea is synthesizing a 2D landscape from all barrier trees, where the folding process is visualized as a walk.

9 Acknowledgements

We would like to thank the anonymous reviewers for their in-depth comments on our work. This work was supported in part by the EMBIO project in FP-6 (`http://www-embio.ch.cam.ac.uk/`).

References

[BDS03] Ulrik Brandes, Tim Dwyer, and Falk Schreiber. Visualizing related metabolic pathways in two and a half dimensions. In Liotta [Lio04], pages 111–122.

[CdBT+92] Robert F. Cohen, Giuseppe di Battista, Roberto Tamassia, Ioannis G. Tollis, and Paola Bertolazzi. A framework for dynamic graph drawing. In *Symposium on Computational Geometry*, pages 261–270, 1992.

[CHS96] Jan Cupal, Ivo L. Hofacker, and Peter F. Stadler. Dynamic programming algorithm for the density of states of RNA secondary structures. In R. Hofstädt, T. Lengauer, M. Löffler, and D. Schomburg, editors, *Computer Science and Biology 96 (Proceedings of the German Conference on Bioinformatics)*, pages 184–186. Universität Leipzig, 1996.

[dBETT94] G. di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: An annotated bibliography. *Computational Geometry: Theory and Applications*, 4(5):235–282, 1994.

[DBL04] *10th IEEE Symposium on Information Visualization (InfoVis 2004), 10-12 October 2004, Austin, TX, USA*. IEEE Computer Society, 2004.

[DG02] Stephan Diehl and Carsten Görg. Graphs, they are changing. In Stephen G. Kobourov and Michael T. Goodrich, editors, *Graph Drawing*, volume 2528 of *Lecture Notes in Computer Science*, pages 23–30. Springer, 2002.

[DS04] Tim Dwyer and Falk Schreiber. Optimal leaf ordering for two and a half dimensional phylogenetic tree visualisation. In Neville Churcher and Clare Churcher, editors, *InVis.au*, volume 35 of *CRPIT*, pages 109–115. Australian Computer Society, 2004.

[EHK+03] Cesim Erten, Philip J. Harding, Stephen G. Kobourov, Kevin Wampler, and Gary V. Yee. Graphael: Graph animations with evolving layouts. In Liotta [Lio04], pages 98–110.

[FE02] Carsten Friedrich and Peter Eades. Graph drawing in motion. *J. Graph Algorithms Appl.*, 6(3):353–370, 2002.

[FFHS00] Christoph Flamm, Walter Fontana, Ivo L. Hofacker, and Peter Schuster. RNA folding at elementary step resolution. *RNA*, 6:325–338, 2000.

[FHSW02] Christoph Flamm, Ivo L. Hofacker, Peter F. Stadler, and Michael T. Wolfinger. Barrier trees of degenerate landscapes. *Z. Phys. Chem.*, 216:1–19, 2002.

[FT04] Yaniv Frishman and Ayellet Tal. Dynamic drawing of clustered graphs. In *INFOVIS* [DBL04], pages 191–198.

[FT07] Yaniv Frishman and Ayellet Tal. Online dynamic graph drawing. In *EUROVIS* [DBL04], pages 191–198.

[GBH04] Ulrich Gerland, Ralf Bundschuh, and Terence Hwa. Translocation of structured polynucleotides through nanopores. *Phys. Biology*, 1(1-2):19–26, 2004.

[GBPD04] Carsten Görg, Peter Birke, Mathias Pohl, and Stephan Diehl. Dynamic graph drawing of sequences of orthogonal and hierarchical graphs. In János Pach, editor, *Graph Drawing*, volume 3383 of *Lecture Notes in Computer Science*, pages 228–238. Springer, 2004.

[GKNV93] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem-Phong Vo. A technique for drawing directed graphs. *IEEE Trans. Software Eng.*, 19(3):214–230, 1993.

[GW05] Marco Gaertler and Dorothea Wagner. A hybrid model for drawing dynamic and evolving graphs. In Patrick Healy and Nikola S. Nikolov, editors, *Graph Drawing*, volume 3843 of *Lecture Notes in Computer Science*, pages 189–200. Springer, 2005.

[HMM00] Ivan Herman, Guy Melançon, and M. Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 06(1):24–43, 2000.

[HSF$^+$06] Christian Heine, Gerik Scheuermann, Christoph Flamm, Ivo L. Hofacker, and Peter F. Stadler. Visualization of barrier tree sequences. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):781–788, 2006.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[Lio04] Giuseppe Liotta, editor. *Graph Drawing, 11th International Symposium, GD 2003, Perugia, Italy, September 21-24, 2003, Revised Papers*, volume 2912 of *Lecture Notes in Computer Science*. Springer, 2004.

[MELS95] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *J. Visual Languages and Computing*, 6(2):183–210, 1995.

[MM04] Irmtraud M. Meyer and Istvan Miklos. Co-transcriptional folding is encoded within RNA genes. *BMC Molecular Biology*, 5(10), 2004.

[Moe90] Sven Moen. Drawing dynamic trees. *IEEE Software*, 7(4):21–28, July 1990.

[Nor95] Stephen C. North. Incremental layout in dynadag. In *Graph Drawing*, pages 409–418, 1995.

[Tam99] Roberto Tamassia. Advances in the theory and practice of graph drawing. *Theoretical Computer Science*, 217(2):235–254, 1999.

[WSSF$^+$04] Michael T. Wolfinger, W. Andreas Svrcek-Seiler, Christoph Flamm, Ivo L. Hofacker, and Peter F. Stadler. Exact folding dynamics of RNA secondary structures. *J. Phys. A: Math. Gen.*, 37:4731–4741, 2004.

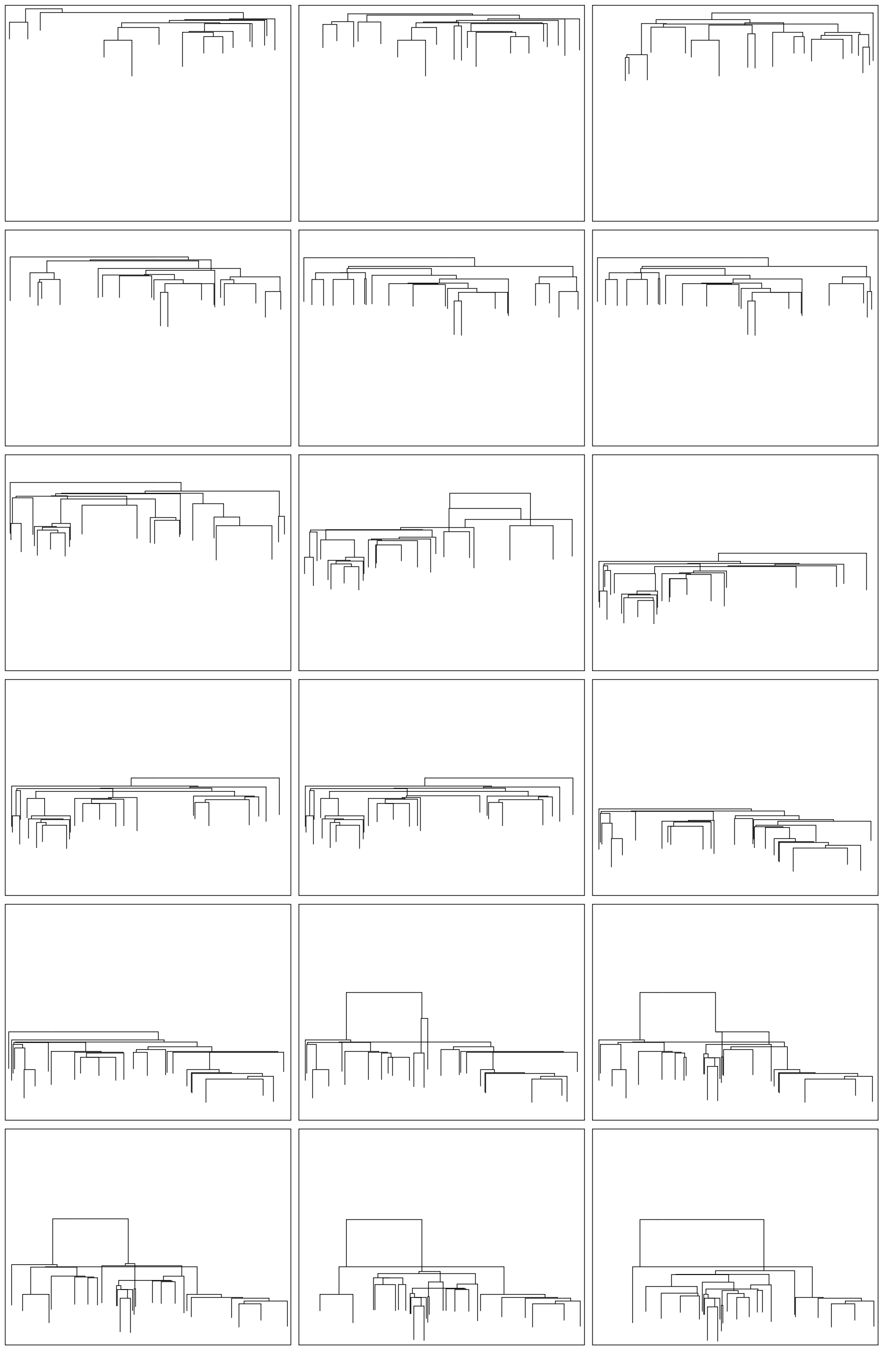
Fig. 2. The first 18 subgraph layouts of the ATT sequence.