Visualization of Barrier Tree Sequences

Christian Heine, Gerik Scheuermann, Christoph Flamm, Ivo L. Hofacker, and Peter F. Stadler

Abstract—Dynamical models that explain the formation of spatial structures of RNA molecules have reached a complexity that requires novel visualization methods that help to analyze the validity of these models. Here, we focus on the visualization of so-called folding landscapes of a growing RNA molecule. Folding landscapes describe the energy of a molecule as a function of its spatial configuration; thus they are huge and high dimensional. Their most salient features, however, are encapsulated by their so-called barrier tree that reflects the local minima and their connecting saddle points. For each length of the growing RNA chain there exists a folding landscape. We visualize the sequence of folding landscapes by an animation of the corresponding barrier trees. To generate the animation, we adapt the foresight layout with tolerance algorithm for general dynamic graph layout problems. Since it is very general, we give a detailed description of each phase: constructing a supergraph for the trees, layout of that supergraph using a modified DOT algorithm, and presentation techniques for the final animation.

Index Terms—Graph drawing, dynamic graph, RNA folding, energy landscape, fitness landscape, barrier tree

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 [3]. Transitions are allowed only between "neighboring configurations", i.e. those that differ by only one base pair [9], 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 [25].

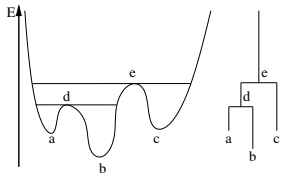

Fig. 1. A very simple landscape and barrier tree
In contrast to normal trees, each vertex of a barrier is drawn at a height that reflects the free energy of the folding configuration it represents. To determine the energy barrier between two local minima, one has to find the barrier tree vertex that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

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. Fig. 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 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. The two joined basins are considered

- *Christian Heine is with Image and Signal Processing Group, Department of Computer Science, University of Leipzig, E-mail: heine@informatik.uni-leipzig.de.*
- *Gerik Scheuermann is with Image and Signal Processing Group, Department of Computer Science, University of Leipzig, E-mail: scheuermann@informatik.uni-leipzig.de.*
- *Christoph Flamm is with Bioinformatics Group, Department of Computer Science, University of Leipzig, E-mail: xtof@bioinf.uni-leipzig.de.*
- *Ivo L. Hofacker is with Department of Theoretical Chemistry and Structural Biology, University of Vienna, E-mail: ivo@bioinf.uni-leipzig.de.*
- *Peter F. Stadler is with Bioinformatics Group, Department of Computer Science, University of Leipzig, E-mail: studla@bioinf.uni-leipzig.de.*

Manuscript received 31 March 2006; accepted 1 August 2006; posted online 6 November 2006.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

to be one, and the "flooding" is continued. See Flamm *et al.* [10] 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 [19]. 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 [15]. 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 of 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.*

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 ([4, 17, 24]).

The first attempts towards dynamic graph drawing were very specific. Moen [21] presented 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.* [2] gave detailed algorithms and data structures for a number of dynamic graph classes. These allow visualizing data structures like AVL- Trees and adjusting the layout of a graph, if it is being edited or browsed. Both approaches share a 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 [22] 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.* [20] 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 "preservation 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 [12] 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 anymore.

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 [11] 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.* [7] 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 [5] 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 is done using morphing geometry information between the single subgraphs. Görg *et al.* [16] 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 [13]. 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.* [1] 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 [6] 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 we consider an important part of our trees.

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 supergraph we construct from the barrier tree sequence will be a directed acyclic graph (DAG). For the layout of this supergraph we implemented and modified the DOT algorithm by Gansner *et al.* [14].

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 the tree at time t does not have much 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 Supergraph

Definitions In the following, $G=(V,E)$ denotes a directed graph, V the vertices and $E \subseteq V \times V$ the edges of G. In the edge $e=(u,v)$ the vertex v is called the head and u is called the tail of e. A directed path in a graph G is a list of edges of G, where the head of each edge in this list is the tail of the edge that follows in the list. If the tail of the first edge equals the head of the last edge, the directed path is called a directed circle. A directed acyclic graph (DAG) is a directed graph that does not contain directed circles. $path_G(u,v)$ shall be true, if and only if there exists a directed path in G starting at u and ending at v. $odeg_G(v)$ denotes the number of edges of G, whose tail is v. $T_i=(V_i,E_i)$ is a rooted tree and also a directed acyclic graph, where all edges are oriented to point away from the root toward the leaves. Note that each leaf v satisfies $odeg_{T_i}(v)=0$. L_i denotes the set of leaves of the tree T_i and F_i an arbitrary subset of L_i. $L_G(v)$ is the set of all vertices w that satisfy both $path_G(v,w)$ and $odeg_G(w)=0$. In a tree, these vertices are leaves, in a directed acyclic graph, they are sinks. Thus $L_G(v)$ assigns the set of leaves/ sinks that can be reached from v to each vertex v. 2^M denotes the set of all subsets of M.

3.1 Problem Definition

The problem of the supergraph of a sequence of trees with leaf mappings is: given a sequence of rooted trees $T_0,\ldots,T_n$ with

$$\forall 0 \le i,j \le n : \left(i \neq j \rightarrow V_i \cap V_j = \emptyset\right)$$

and

$$\forall 0 \le i \le n : \forall v \in V_i : \mathrm{odeg}_{T_i}(v) \neq 1$$

and a sequence of leaf mappings $f_1,\ldots,f_n$ with $f_i : F_{i-1} \rightarrow L_i$, find the smallest graph $G=(V,E)$ and a global mapping of tree vertices on supergraph vertices $k=\bigcup_{i=0}^{n} k_i$, $k_i : V_i \rightarrow V$, k_i injective, such that

1. G contains all trees:

   $$\forall 0 \le i \le n : (k_i(V_i) \subseteq G \wedge \forall (u,v) \in E_i : \mathrm{path}_G(k_i(u),k_i(v)))$$

   and each path from u to v does not visit vertices from $k_i(V_i)$ except u and v.

2. G conforms to the leaf mapping:

   $$\forall 1 \le i \le n : \forall u,v \in V_{i-1} : (f_i(u) \neq f_i(v) \rightarrow k_i(f_i(u)) \neq k_i(f_i(v)))$$

3. G conforms to the topological properties of all trees:

   $$\forall 0 \le i \le n : \forall u,v \in V_i : \neg\,\mathrm{path}_{T_i}(u,v) \rightarrow \neg\,\mathrm{path}_G(k_i(u),k_i(v))$$

3.2 Motivations

The first step of the *Foresight Layout with Tolerance* algorithm [5] is to construct a supergraph of all the graphs in a sequence. The supergraph is the smallest graph that contains all graphs of the sequence as subgraphs. To accomplish this, it is necessary to know which vertices of the graphs should be considered equivalent. Leaf mappings between successive trees are used as a base for this process, however, this can only be applied directly to some of the leaves of the trees. The identification of equivalent inner vertices and leaves that result from merging leaves in the previous tree is non trivial. We did not motivate this identification by graph theoretic minimization, but decided that the supergraph should reflect properties of the corresponding landscapes. This has the advantage, that the supergraph may be used as an alternative and static representation of the barrier tree sequence.

A barrier tree not only stores energy barriers between local minima, it also gives a rough and abstract view on the topology of a landscape. The shape of the barrier tree illustrates the order of the unification of basins. This unification order will be used to identify equivalent inner vertices. If, for instance, an inner vertex u has two leaves as its children that are mapped to two different leaves of the following tree having the same parent v, the inner vertex u and the parent v can be seen as topologically equivalent. If the leaf mapping is extended by this new information, further parts of the trees can be processed to further identify inner vertices as equivalent, and to quickly identify isomorphic structures between the barrier trees that conform to the leaf mapping. This takes only the topology of the barrier tree into account. The energy information about each vertex is neglected.

This procedure ends abruptly, as soon as there is the slightest topological difference in a barrier tree. In practice, this strict behavior results in a large number of vertices that are not considered to be equivalent. This can be avoided by identifying equivalent inner vertices based on the set of local minima that can be reached from the corresponding barrier by descending in the landscape. In Fig. 2a, vertex e and j are considered to be equivalent, because the sets of leaves that can be reached from them are equal considering the leaf mapping. Vertices d and i are not considered to be equivalent because the set of leaves that can be reached from them, $\{a,b\}$ and $\{g,h\}$ respectively, are not equal considering the leaf mapping. Such cases are very common and are generated mostly, when the the height of barriers between successive trees change. The supergraph is in that case no longer a tree, but a directed acyclic graph (DAG). This is unavoidable, but the supergraph will always be at most a DAG.

Imagine, that the barrier swap from Fig. 2a is reverted at time $t+2$. The tree at time $t+2$ conveys exactly the same information as the tree at time $t+0$. It contains an inner vertex that is not equivalent to any vertex of tree $t+1$, but equivalent to vertex d. This vertex should not be inserted in the supergraph, as it does not represent "new" information. But this fact cannot be concluded by looking at tree $t+$

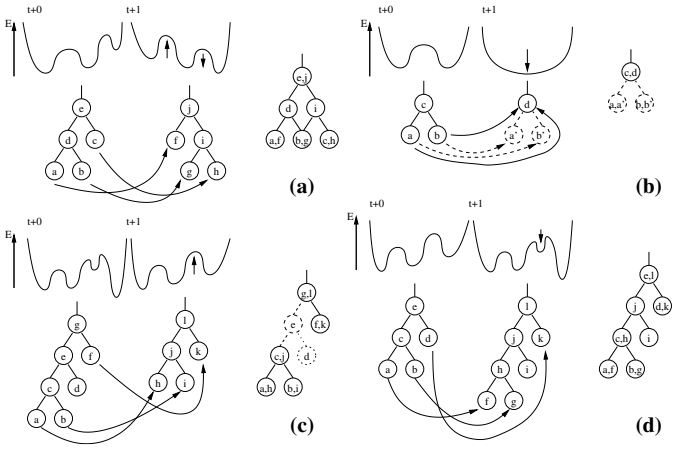

Fig. 2. Examples of elementary landscape and barrier tree changes
Each figure shows, how the energy landscape changes, illustrates the barrier trees (only the topology is shown) and the leaf mappings and shows, how the supergraph should look like in the cases: barrier swap (a), leaf merging (b), leaf vanishing (c), leaf creation (d).

1 alone. Considering all past trees can get quite complicated, it is much easier to just look into the supergraph for the past trees. The supergraph can and will be used as a data structure to quickly identify equivalent inner vertices of the barrier trees. It is efficient to construct the supergraph iteratively. To determine the supergraph for the trees T_0 to T_{n+1}, we use the supergraph of the trees T_0 to T_n for identification of equal vertices and add any new information we gain from tree T_{n+1}.

Fig. 2b shows another common case of change in the energy landscape. Often barriers disappear, and local minima get merged. Obviously our "set of leaves" approach fails in that case, the vertices c and d would not be considered equivalent ($\{a,b\}$ vs. $\{d\}$). The solution is to temporarily add the mirror vertices a' and b' as children to d and modify the leaf mapping. This methodology is a must, if more than two leaves merge or the merging leaves do not share the same parent. Merged leaves must be marked as inactive in the supergraph, so they will not be considered for the "set of leaves" of other inner vertices.

In Fig. 2c a leaf vanishes, i.e. it is not part of the leaf mapping. This may happen, because the number of leaves is usually reduced to the most relevant ones, and a relevant leaf may have a non relevant successor. In such a case the leaf (d) is marked as inactive and is not considered for the set of leaves. This leaves us with the problem, that the vertices c and e of tree $t+0$ have the same set of leaves ($\{a,b\}$), and thus vertex j is considered equivalent to both vertices. In that case, the vertex farthest from the root (c) is selected. What becomes apparent now is, that the tree $t+1$ is not really a subgraph of the supergraph, because it lacks an edge from g,i to c,j. The supergraph is still an expansion of tree $t+1$.

In Fig. 2d a leaf is added to the tree. This is the inverse of the previous case. The edge from e to c is replaced by a path (i,j,h) and the new leaf is added at the appropriate location. Again the supergraph is an expansion of tree $t+0$. The removal of transitive edges has little to no effect on the quality of the final presentation, but reduces the size of the supergraph and greatly improves the performance, when the layout of the supergraph is determined.

These four operations are considered elementary and are the only operations we observed in our datasets. However, it is expected that multiple elementary operations take place between successive trees of the sequence. Because creation, deletion, and merging cannot happen to the same leaf of a tree simultaneously, these operations and the supergraph modifications they imply do not affect each other. Also creation, deletion, and merging happen at or near the leaves, while *barrier swaps* only add inner vertices. So these operations also do not affect each other and can be done separately.

3.3 Construction

For each directed graph $G=(V,E)$ define the function $mark_G$ as:

$$\begin{aligned} \mathrm{mark}_G : 2^V &\rightarrow 2^V \\ M &\mapsto \left\{ v | L_G(v) \subseteq \bigcup_{u\in M} L_G(u) \right\} \end{aligned}$$

The operation of this function may be described as this: Starting from the vertices of M, all incoming edges are marked. If all outgoing edges of a vertex get marked in that process, that vertex is added to M and the process continues. The process ends, if no more vertices can be added to M. Fig. 3 illustrates this. Obviously $M \subseteq \mathrm{mark}_G(M)$ and $M=\emptyset$, if and only if $\mathrm{mark}_G(M)=\emptyset$. Unlike the example, M does not have to contain leaves/ sinks only.

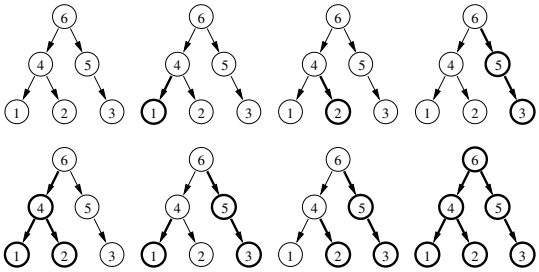

Fig. 3. Example for the mark_G function
The result is illustrated by vertices with thick circles.
top left to right: $\mathrm{mark}_G(\emptyset)$, $\mathrm{mark}_G(\{1\})$, $\mathrm{mark}_G(\{2\})$, $\mathrm{mark}_G(\{3\})$.
bottom left to right: $\mathrm{mark}_G(\{1,2\})$, $\mathrm{mark}_G(\{1,3\})$, $\mathrm{mark}_G(\{2,3\})$, $\mathrm{mark}_G(\{1,2,3\})$.

The function $match_G$ reduces a mark to the topmost layer:

$$\begin{aligned} \mathrm{match}_G : 2^V &\rightarrow 2^V \\ M &\mapsto \mathrm{mark}_G(M) \cap \{ v | \forall (u,v) \in E : u \notin \mathrm{mark}_G(M) \} \end{aligned}$$

For the example in Figure 3: $match_G(\emptyset)=\emptyset$, $match_G(\{1\})=\{1\}$, $match_G(\{3\})=\{5\}$, $match_G(\{1,2\})=\{4\}$, $match_G(\{2,3\})=\{2,5\}$, $match_G(\{1,2,3\})=\{6\}$.

Construct G iteratively: $G_0=T_0$, $\forall v \in V_0 : k_0(v)=v$. Construct $G_i=(V_i',E_i')$ and $k_i : V_i \rightarrow V_i'$ from $G_{i-1}=(V_{i-1}',E_{i-1}')$, $k_{i-1} : V_{i-1} \rightarrow V_{i-1}'$, $T_i=(V_i,E_i)$ and f_i as follows:

Determine the active part of the Supergraph G_{i-1}, this is much easier than tracking inactive (deleted or merged) parts of the supergraph:

$$G_i' = (A_i, K_i)$$

$$A_i = \left\{ v | \, v \in V_{i-1}' \wedge \exists l \in L_i : \mathrm{path}_{G_{i-1}}(v, k_{i-1}(l)) \right\}$$

$$K_i = E_{i-1}' \cap A_i \times A_i$$

For each vertex of the tree T_i determine the set of leaves of T_i that can be reached from that vertex:

$$\begin{aligned} M_i : V_i &\rightarrow 2^{L_i} \\ u &\mapsto \left\{ v | \, v \in L_i \wedge \mathrm{path}_{T_i}(u,v) \right\} \end{aligned}$$

For each vertex of the tree determine its *leaf set*, i.e. the set of vertices of the active part of the supergraph, that map on a leaf in M_i because of the leaf mapping:

$$\begin{aligned} B_i : V_i &\rightarrow 2^{V_i} \\ v &\mapsto \{ k_{i-1}(w) | \, f_i(w) \in M_i(v) \} \end{aligned}$$

Using the $match_G$ function find vertices of the active part of the supergraph with the most similar set of leaves:

$$l_i(v) = \mathrm{match}_{G_i'}(B_i(v))$$

Determine all children of a tree vertex that have an empty *leaf set*. These children are vertices that are created in the current barrier tree. Note that, if all children of a tree vertex have an empty *leaf set*, that vertex will also have an empty *leaf set* also and is thus a newly created inner vertex of the barrier tree.

$$n_i(v) = \{\, w \,|\, (v,w) \in E_i \wedge B_i(w) = \emptyset \}$$

Barrier tree vertices can now be categorized:

- $fresh(v)$, iff $l_i(v) = \emptyset$. v is a new vertex in the current barrier tree.
- $matching(v)$, iff $|l_i(v)| = 1 \wedge n_i(v) = \emptyset$. In that case an equivalent vertex has been found in the supergraph. This vertex is the one element of $l_i(v)$ and no child of v is *fresh*.
- $matchfresh(v)$, iff $|l_i(v)| = 1 \wedge n_i(v) \neq \emptyset$. An equivalent vertex has been found in the supergraph. At least one child of v is *fresh*.
- $recomb(v)$, iff $|l_i(v)| > 1$. An equivalent vertex could not be found. $l_i(v)$ contains the most similar vertices.

Each vertex of the tree must be inserted in the supergraph, unless an equivalent vertex had been found.

$$V_i' = V_{i-1}' \cup \{\, v \,|\, v \in V_i \wedge \neg \mathrm{matching}(v) \}$$

$$k_i(v) = \begin{cases} u & l_i(v) = \{u\} \wedge n_i(v) = \emptyset \\ v & l_i(v) = \{u\} \wedge n_i(v) \neq \emptyset \end{cases}$$

The inserted edges are:

$$\begin{aligned} E_i'' = E_{i-1}' \quad & \cup \quad \{\, (u,v) \,|\, v \in V_i \wedge (u,w) \in E_{i-1}' \wedge \mathrm{matchfresh}(v) \} \\ & \cup \quad \{\, (v,w) \,|\, v \in V_i \wedge l_i(v) = \{w\} \wedge n_i(v) \neq \emptyset \} \\ & \cup \quad \{\, (k_i(v),w) \,|\, v \in V_i \wedge w \in l(v) \wedge \neg \mathrm{matching}(v) \} \\ & \cup \quad \{\, (k_i(u),k_i(v)) \,|\, (u,v) \in E_i \} \end{aligned}$$

Transitive edges may be removed:

$$E_i' = \left\{ (u,v) \,|\, (u,v) \in E_i'' \wedge \neg \exists \mathrm{path}_{(V_i', E_i'')}(u,w) \neq (u,w) \right\}$$

The final supergraph G is equal to the supergraph G_n, i.e. the supergraph after inserting each tree of the sequence. Additional material to this article may be found on the accompanying DVD. It includes some algorithms that illustrate implementation details for the operations needed for this supergraph construction.

3.4 Example

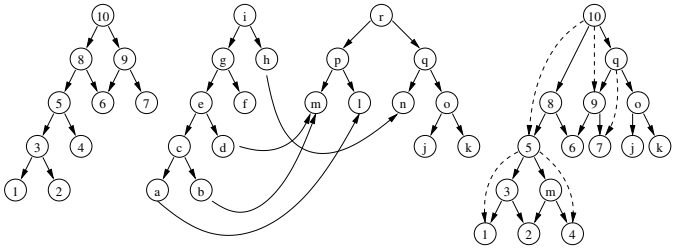

Fig. 4. Example construction of the supergraph of two trees
left to right: the supergraph G_i, the tree T_i, the tree T_{i+1}, and the supergraph G_{i+1}. Arrows between T_i and T_{i+1} indicate the leaf mapping. The dashed lines in G_{i+1} indicate edges that can be replaced by a path. The exact description, how the trees are embedded in the supergraph and how the supergraph is modified in this iteration, are found in the main text.

Fig. 4 shows a nontrivial example for one iteration of the supergraph construction process. It has been chosen to show all four elementary operations that can modify barrier trees. k_i, mapping the vertices of T_i to vertices of G_i is:

$$k_i = \{(a,1),(b,2),(c,3),(d,4),(e,5),(f,6),(g,8),(h,7),(i,10)\}$$

T_i is thus very similar to G_i, only the edge (i,h) of the tree is represented by the path $(10,9,7)$ in G_i. The vertex f does not occur in the leaf mapping, i.e. , it is deleted. The active part of G_i is thus: $A_{i+1} = \{1,2,3,4,5,7,8,9,10\}$. Because of the leaf mapping the *leaf sets* of the vertices of T_{i+1} are:

$$\begin{aligned} B_{i+1} \quad = \quad & \{(j,\emptyset),(k,\emptyset),(l,\{1\}),(m,\{2,4\}),(n,\{7\})\} \\ \cup \quad & \{(o,\emptyset),(p,\{1,2,4\}),(q,\{7\}),(r,\{1,2,4,7\})\} \end{aligned}$$

After $mark_{(A_{i+1},K_{i+1})}$ and $match_{(A_{i+1},K_{i+1})}$ have been determined, l_{i+1} and n_{i+1} result to:

$$\begin{aligned} l_{i+1} \quad = \quad & \{(j,\emptyset),(k,\emptyset),(l,\{1\}),(m,\{2,4\}),(n,\{9\})\} \\ \cup \quad & \{(o,\emptyset),(p,\{5\}),(q,\{9\}),(r,\{10\})\} \\ n_{i+1} \quad = \quad & \{(j,\emptyset),(k,\emptyset),(l,\emptyset),(m,\emptyset)\} \\ \cup \quad & \{(n,\emptyset),(o,\{j,k\}),(p,\emptyset),(q,\{o\}),(r,\emptyset)\} \end{aligned}$$

The vertices of T_{i+1} are categorized as follows:
$fresh(j)$, $fresh(k)$, $match(l)$, $recomb(m)$, $match(n)$,
$fresh(o)$, $matching(p)$, $matchfresh(q)$, $matching(r)$.
Therefore the following vertices have to be added to the supergraph, and k_{i+1} results to:

$$V_{i+1} = V_i \cup \{j,k,m,o,q\}$$

$$k_{i+1} = \{(j,j),(k,k),(l,1),(m,m),(n,9),(o,o),(p,5),(q,q),(r,10)\}$$

Insertion of the edges is left as an exercise to the reader. Some transitive edges may be removed.

3.5 Post-processing

Unfortunately, the use of the supergraph as a data structure to find similar leaf sets often requires the insertion of edges that are not needed for the final solution. Some edges are inserted to ensure correct results for the *match* and *mark* functions, but are not required for the supergraph to be an expansion of all trees. Removal of these edges decreases both the possibility of edge crossings in and the running time of the layout process.

These edges are identified as a side product in a post-processing phase. In this phase each edge of the supergraph is annotated with the set of all trees it occurs in. The motivation for this will be explained in the next section. Usually a tree edge corresponds to a path in the supergraph. Therefore, each edge of the path is annotated with the tree. Quite frequently, there are multiple possible paths for one tree edge. In such cases only the edges of the longest path are annotated. After annotation, there will be many edges which do not belong to any tree. These can be removed safely. Choosing the longest path is a simple and quick heuristic that favors edges with a high probability of reuse. In practice this removes 5–20 percent of all edges of the supergraph plus any transitive edge. However, a proper problem definition for this phase would be: find the largest set of edges that can be removed without violating the constraint, that the supergraph is an expansion to each tree.

4 Layout

4.1 Supergraph Layout

The second step of the *Foresight Layout with Tolerance* algorithm creates the layout of the supergraph. In general, the supergraph will be a DAG. Sugiyama *et al.* [23] proposed to split the task in three phases. In the first phase, the *ranking*, vertices are grouped in successive layers, such that the edges are oriented in one direction, usually from top to bottom. In the second phase, the *ordering*, an order of vertices in each layer is determined that minimizes edge crossings. In the final phase, the *positioning*, coordinates are assigned to each vertex, preserving

the order inside the layers, but minimizing the overall edge length by shifting vertices inside the layers. In this work we used the heuristics presented Gansner *et al.* [14] — also known as the DOT algorithm — to lay out the supergraph and made slight modification.

The main esthetic goal for DAG layout is the removal of edge crossings. In practice, the supergraph contains a large number of edge crossings that do not matter, because these crossing edges are never be shown simultaneously. The annotation of the postprocessing phase allows us to weight the importance of an edge crossing. In the DOT algorithm, graphs are laid out respecting edge weights. Edges with a high weight are kept short and crossing free. in. In the *ordering* phase, the weight of an edge is replaced by the weight of an edge crossing. This weight is generated from the number of trees, that both edges actually cross.

In the original algorithm, crossing reduction is done by repeatedly iterating over and through all layers, switching the order of two successive vertices, if that locally improves the number of edge crossings. Sometimes, such a switch does neither improve nor deteriorate the number of edge crossings, but the switch may lead to further improvements. Gansner *et al.* [14] suggest performing such switches only every other global iteration. In our case, many crossing weights will be zero and switches seldom improve the number of edge crossings immediately. As a result, the original formulation leads to long running times of the algorithm and changes periodically from one extreme to the other.

In our implementation, we perform the switch randomly in a simulated annealing process. Improvement is always and deterioration never accepted. A temperature, initially 1, is used as the probability of performing a switch that does not change the number of weighted edge crossings. Each global iteration the system cools down, the temperature decreases exponentially. The process terminates, if no more switches are performed. In practice this resulted in a higher number of edge crossings, but reduced the running time drastically.

The routing of edges is not relevant for the layout of the supergraph. The edges will be routed only in the subgraphs.

4.2 Tree Layout

Until now, the energy of a vertex has been ignored. 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. Because we want one of the coordinates to indicate the energy, it is not possible to do the third phase of the DAG- layout, the *positioning*, for the whole supergraph. Coordinate assignment is done for each tree separately, respecting the order generated in the *ordering* phase. This constraint preserves the *mental map*, specifically the *orthogonal ordering*. 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.

The DOT algorithm may give two vertices the same horizontal position, if they are in different layers. If we assign these vertices their energy value as the vertical position it is possible, that they overlap. By assigning each vertex of the supergraph an unique horizontal position we can avoid this. However, if the layout is created in that way, the vertices of the subgraphs can be very unevenly distributed. We found it visually more appealing, if only sinks of the supergraph have an unique horizontal position. This can be achieved trivially, if all sinks are positioned in the same layer prior to the *ordering* phase.

After the vertices have been positioned, edges must be routed. For simplicity each tree edges consist just of one horizontal and one vertical line segment that directly connect the two adjacent vertices. Edges are routed independently of the supergraph, where an edge might have been replaced by a path. Because of the problem definition, vertices on that path would not be a part of the current tree and thus layout information of these vertices is not valid for this tree. In general, it is not always possible to draw the subgraphs 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 drawn usually. We also found, that a straight line drawing does not reduce the number of edge crossings and makes tracing the edges even harder than an orthogonal drawing.

5 Animation

Now that 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 the 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 segments overlap intentionally, as we observed many changes to interior vertices. The remaining interval $[t_i, t_i + \frac{2}{8}\Delta t)$ is used for a static presentation of tree T_i.

6 Results

To create and evaluate our algorithm we had three datasets at our disposal. 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. This example was used to design and test the algorithm. Fig. 5 shows the keyframes for this dataset, as well as some explanations from experts. The full animation can be found on the DVD proceedings. 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.

One way to determine the quality of the algorithm is to look at properties of the supergraph. The number of vertices in the supergraph of the ATT, LEPTO, and HOK datasets are 392, 1874 and 4594 respectively. This means that only about half of the vertices of the trees were identified as redundant. This results from a property of the sequences that we have not yet mentioned. In each new tree of a sequence, leaves get deleted, merged, and added. The average number of leaves that are added is 5.00, 7.20 and 16.16 respectively. That means that up to 20 percent of each tree changes on average. It can be shown, however, that a graph-theoretic minimum supergraph would not be smaller than approximately half to one third of the size of our supergraphs.

More critical to the perceived quality of the layout is the number of edges. If this number is near the number of vertices, the supergraph is very similar to a tree and can thus be drawn with few edge crossings and (horizontally) short edges. Horizontally long edges in the supergraph layout are undesirable, because each edge is shown at least once. The amount of edges divided by the amount of vertices for the three datasets are 1.52, 1.69, and 1.61 respectively. Although these numbers seem to be close, the LEPTO and HOK datasets have a significantly larger number of edge crossings and long edges than the ATT dataset. This is because the edges are unevenly distributed among the layers of the supergraph layout. The animation suffers from long edges that are close together and are notoriously difficult to track.

Two preprocessing methods have been tested to determine, if a subset of the data still results in bad layouts. Surely, we do not want to reduce the number of trees, since we want to visualize the whole process.There are a number of barriers that are connected by an edge in the barrier tree, and whose energy differs only slightly. Such barriers are merged in a preprocessing step. This process reduces the probability of *barrier swaps* and the supergraph will have less vertices. In the LEPTO and HOK datasets, the merging of barriers that differ by 0.5 or less (which is approximately two percent of the overall energy range) reduced the total number of tree vertices to 2419 and 5863 respectively and the number of supergraph vertices to 853 and 2493 respectively. This means, that now nearly two third of the vertices were identified as redundant. Unfortunately this method does not reduce the number of edges as much as the number of vertices, thus the supergraph suffers from a huge number of edge crossings and long horizontal edges. After applying this method, the supergraph span less layers and the edges got distributed more equal over the layers. In the final animation long edges are still visible, but they are no longer close together, so it is easier to track them.

The second method is the reduction of leaves in the barrier trees. Local minima with a low energy are generally more stable and have a high probability of being present in the next barrier tree. They are also more interesting than local minima of higher energy. For each leaf that is removed, the one barrier connecting it to the rest of the tree is removed as well. By reducing the number of leaves in the LEPTO and HOK datasets to a maximum of 31 and 66 leaves per tree, the total number of tree vertices was reduced to 2409 and 5875 respectively. The number of supergraph vertices was reduced to 732 and 2528, so again almost two third of the tree vertices have been identified as redundant. This preprocessing method removed substantially more edges than vertices, and in the LEPTO and HOK datasets the number of edges divided by the number of vertices decreased to 1.50 and 1.44 respectively, which greatly improved the supergraph layout. There were a lot less edge crossings and only a few long edges. This directly resulted in a better layout of the barrier trees.

7 Conclusion and Future Work

We showed that it is possible to generate readable layouts for sequences of barrier trees using the *Foresight Layout with Tolerance* algorithm. For larger datasets, preprocessing may have to be applied to the sequence. While reducing barriers decreases the height of the supergraph layout, a reduction of leaves decreases the width and greatly improves the perceived quality of the layout.

From the viewpoint of folding landscapes, often only a small number of leaves are of interest. These leaves and their history can be highlighted using colors. 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.

The current methods to generate the animation leave room for further improvements. Different strategies for edge removal during the postprocessing of the supergraph construction can result in an improved layout, because fewer edges generally result in fewer edge crossings. Rather than overgenerating the edges of the supergraph and reducing it afterwards, a more constructive method could be proposed. In this article we did not pay much attention to local improvement of the subgraph layout. Especially in larger datasets this would be beneficial, because each subgraph only uses a small part of the drawing area and requires high resolution. A local improvement strategy based on a force-directed strategy is currently being implemented.

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.

References

[1] U. Brandes, T. Dwyer, and F. Schreiber. Visualizing related metabolic pathways in two and a half dimensions. In Liotta [18], pages 111–122.

[2] R. F. Cohen, G. di Battista, R. Tamassia, I. G. Tollis, and P. Bertolazzi. A framework for dynamic graph drawing. In *Symposium on Computational Geometry*, pages 261–270, 1992.

[3] J. Cupal, I. L. Hofacker, and P. 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.

[4] 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.

[5] S. Diehl and C. Görg. Graphs, they are changing. In S. G. Kobourov and M. T. Goodrich, editors, *Graph Drawing*, volume 2528 of *Lecture Notes in Computer Science*, pages 23–30. Springer, 2002.

[6] T. Dwyer and F. Schreiber. Optimal leaf ordering for two and a half dimensional phylogenetic tree visualisation. In N. Churcher and C. Churcher, editors, *InVis.au*, volume 35 of *CRPIT*, pages 109–115. Australian Computer Society, 2004.

[7] C. Erten, P. J. Harding, S. G. Kobourov, K. Wampler, and G. V. Yee. Graphael: Graph animations with evolving layouts. In Liotta [18], pages 98–110.

[8] G. Fayat, J.-F. Mayaux, C. Sacerdot, M. Fromant, M. Springer, M. Grunberg-Manago, and S. Blanquet. *Escherichia coli* phenylalanyl-tRNA synthetase operon region : Evidence for an attenuation mechanism. Identification of the gene for the ribosomal protein L20. *J. Mol. Biol.*, 171(3):239–352, 1983.

[9] C. Flamm, W. Fontana, I. L. Hofacker, and P. Schuster. RNA folding at elementary step resolution. *RNA*, 6:325–338, 2000.

[10] C. Flamm, I. L. Hofacker, P. F. Stadler, and M. T. Wolfinger. Barrier trees of degenerate landscapes. *Z. Phys. Chem.*, 216:1–19, 2002.

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

[12] Y. Frishman and A. Tal. Dynamic drawing of clustered graphs. In *INFOVIS*, pages 191–198. IEEE Computer Society, 2004.

[13] M. Gaertler and D. Wagner. A hybrid model for drawing dynamic and evolving graphs. In P. Healy and N. S. Nikolov, editors, *Graph Drawing*, volume 3843 of *Lecture Notes in Computer Science*, pages 189–200. Springer, 2005.

[14] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Trans. Software Eng.*, 19(3):214–230, 1993.

[15] U. Gerland, R. Bundschuh, and T. Hwa. Translocation of structured polynucleotides through nanopores. *Phys. Biology*, 1(1-2):19–26, 2004.

[16] C. Görg, P. Birke, M. Pohl, and S. Diehl. Dynamic graph drawing of sequences of orthogonal and hierarchical graphs. In J. Pach, editor, *Graph Drawing*, volume 3383 of *Lecture Notes in Computer Science*, pages 228–238. Springer, 2004.

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

[18] G. 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.

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

[20] 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.

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

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

[23] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical systems. *IEEE Trans. Systems, Man, and Cybernetics*, 11:109–125, 1981.

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

[25] M. T. Wolfinger, W. A. Svrcek-Seiler, C. Flamm, I. L. Hofacker, and P. F. Stadler. Exact folding dynamics of RNA secondary structures. *J. Phys. A: Math. Gen.*, 37:4731–4741, 2004.

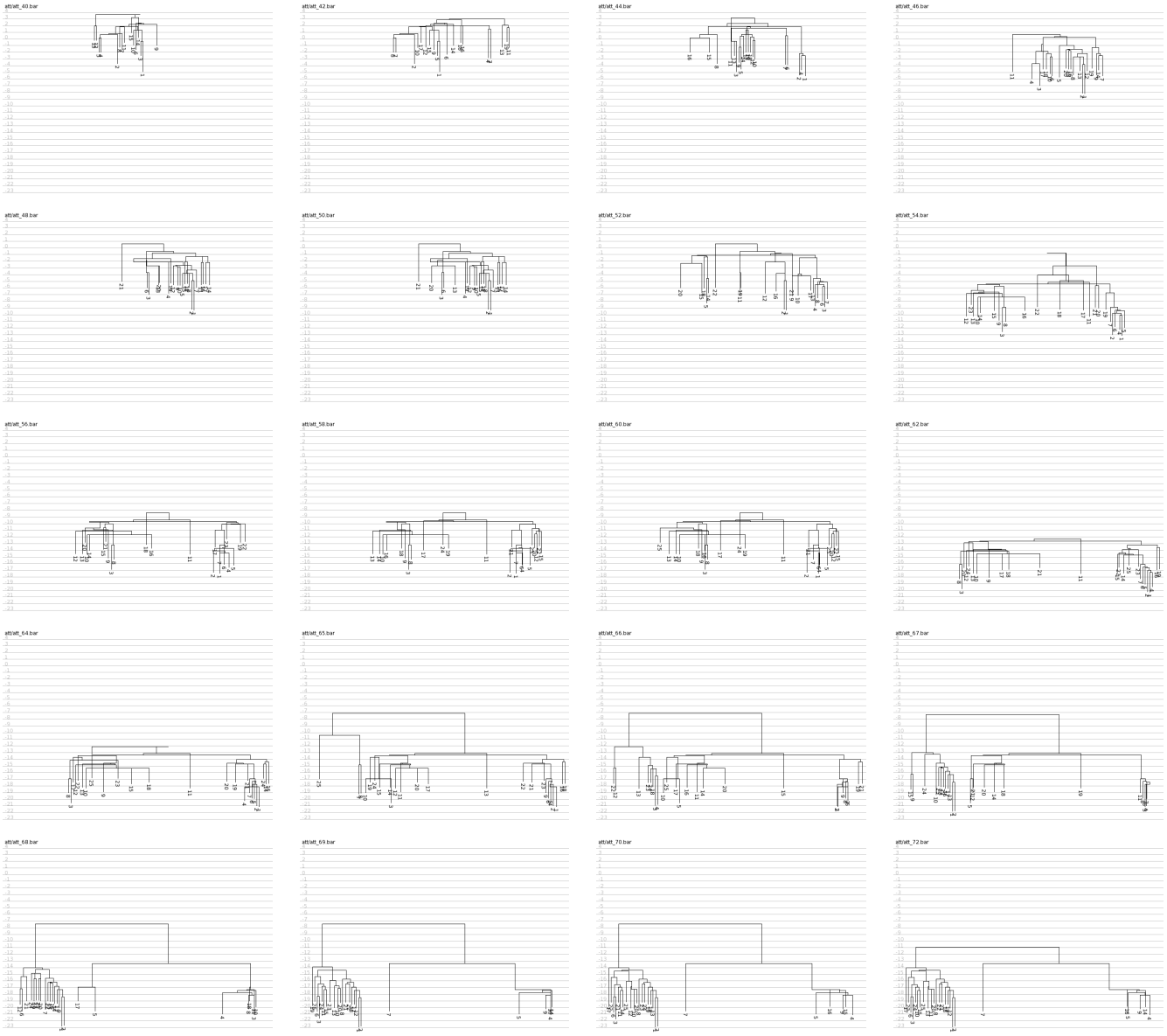

Fig. 5. The 20 subgraph layouts of the ATT sequence.

Terminator/anti-terminators are small RNA structure elements occurring in bacterial messenger RNAs (mRNAs) that modulate the translation of mRNA into a protein. They are switch-like elements that can form alternative structures with drastically different physiological function. In one state, the mRNA is translated and protein is produced, while the alternative RNA conformation suppresses this process. The speed of transcription determined, which of these two states in reached. The barrier-tree sequence of the growing RNA element can be used to understand the molecular mechanism for this behavior.

The above panels show the leader sequence of the pheS-pheP operon of *E. coli* [8]. In the first stages, i.e., when only the 5' (left) part of the molecule has been transcribed, the folding landscape is dominated by a single conformation (visible as the lowest-energy subtree in panels 5 and 6. As the molecule grows, an alternative basin of attraction (left subtree in rows 3 to 5) appears. In the first stages, this class of conformation is less stable than the r.h.s. sub-tree, which is initially populated as it corresponds to the stable conformations in the first folding stages. In the full-size element, however, a different conformation class is thermodynamically favored, which appears as the l.h.s. subtree in the last 7 panels. The important observation is that this class of conformations is reachable only by transversing a sizable energy barrier; hence the transition into this conformation is slow. The speed of translation thus determines, whether the growing chain has sufficient time to switch into the optimal subtree (approximately in panels 13-15), or whether it remains trapped in the r.h.s. subtree, as the barrier height (and hence the necessary transition time) increases with the chain length (panels 17-19).