Visualization of Barrier Tree Sequences

Christian Heine[*]
Image and Signal Processing Group
Department of Computer Science
University of Leipzig

Gerik Scheuermann[†]
Image and Signal Processing Group
Department of Computer Science
University of Leipzig

Christoph Flamm[‡]
Bioinformatics Group
Department of Computer Science
University of Leipzig

Ivo L. Hofacker[§]
Department of Theoretical
Chemistry and Structural Biology
University of Vienna

Peter F. Stadler[¶]
Bioinformatics Group
Department of Computer Science
University of Leipzig

Abstract

The increasing complexity of models explaining the spatial structure of RNA molecules require visualization methods that help to analyze the validity of these models. In this article, we consider the visualization of so called folding landscapes in the case of a growing RNA molecule. These folding landscapes may be thought of discrete energy or fitness landscapes and tend to be huge and high dimensional. They are reduced to a so called barrier tree that reflects the essential properties of the landscape such as local minima and saddle points between them. For each sequence length of the growing RNA chain, there is a folding landscape. We visualize the sequence of folding landscapes by an animation of the corresponding barrier trees. To do that, we adopt the foresight layout with tolerance algorithm for general dynamic graph layout problems. Since it is very general, we give detailed account to each phase: constructing a supergraph for the trees, layout of that supergraph using a modified dot algorithm, and presentation techniques for the final animation.

CR Categories: G.2.2 [Discrete Mathematics]: Graph Theory—Trees

Keywords: graph drawing, dynamic graph, RNA folding, energy landscape, fitness landscape, barrier tree

1 Introduction

1.1 Biological Background

Ribonucleic acid (RNA) is a biopolymer, i.e. a chain of covalently bound nucleotides. There are four different types of nucleotides or bases found in RNA: adenine, guanine, cytosine, and uracil. RNA molecules play an important role in many biological contexts, e.g. protein synthesis. The biological function of a RNA molecule is determined by its spatial structure rather than the actual sequence of nucleotides. This structure is build up in a process called folding by building up hydrogen bonds between nucleotides. Not all combinations of nucleotides are legal or energetically equal. The folding reaction releases some energy. For simplicity sake, often only the secondary structure is considered. This *secondary structure* is just a list of base pairs that describe, which nucleotides are connected by hydrogen bonds. Secondary structures are the backbone of the spatial structure and determine most of its properties, e.g. the energy, yet are mathematically simple enough to easily operate with. While the number of possible spatial structures grows exponentially with the length of the base sequence, only few of these structures are found in nature. It is assumed that a RNA molecule is able to fulfill its purpose only with a certain structure, the native state.

Various methods have been proposed to explain and predict the structures of RNA molecules. A very simple attempt to predict the structure is to find the secondary structure with maximum number of legal base pairs. This method is not biologically motivated and does not predict the correct structure. A better attempt is to look for the structure with the lowest free energy, i.e. the structure where the most energy gets released during the folding from the completely unfolded state. Since most physical processes are motivated by energy minimization, the structure with the lowest free energy tends to be the most stable, and has thus the highest probability of occurrence. When all other structures along with their free energy are taken into consideration too, one can estimate the transition rates between the configurations using only the differences of the free energies and compute the probability of occurrence by solving a system of differential equations [3]. A problem for this method is the size of the transition matrix caused by the huge number of possible configurations. This thermodynamic approach assumes that each transition can be done directly, it neglects properties of the corresponding refolding reaction. A refined version of the thermodynamic approach is given by Flamm [5] and assumes transitions only to take place between neighboring configurations. Two configurations are neighbors, if their structural differences are smaller than a given threshold. For instance, two configurations can be seen as neighbors, if their secondary structures differ by only one base pair. The whole configuration space can be transformed into a large graph with edges between neighboring structures. If the structural dissimilarities between two configurations are small, the difference in energy will be small as well. 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 maximum free energy called the *maximum*. 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. The transition probabilities between two neighboring configurations can be calculated directly like in the thermodynamic approach but the transition probabilities for non- neighboring configurations has to consider all paths between the two configurations and all elementary steps on these paths. All this information can be stored in a so called barrier tree, leaves representing local minima and inner vertices representing the barriers between them. Fig. 1 shows an example of a barrier tree for a very simple landscape. Barrier trees are constructed by successively flooding the valleys of the landscape. A barrier is found at the point where the lakes of two valleys join. A detailed description of generating barrier trees from landscapes is given by Flamm *et al.* [6].

The thermodynamic approach with respect to the kinetic properties of the folding still does not fully represent reality. RNA

[*]e-mail: heine@informatik.uni-leipzig.de
[†]e-mail: scheuermann@informatik.uni-leipzig.de
[‡]e-mail: xtof@bioinf.uni-leipzig.de
[§]e-mail: ivo@tbi.univie.ac.at
[¶]e-mail: peter.stadler@bioinf.uni-leipzig.de

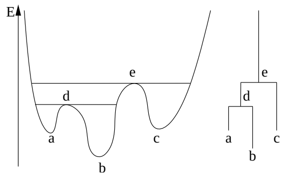


Figure 1: a very simple landscape and barrier tree
In addition to normal trees each vertex of a barrier tree also has an energy value attached, that corresponds to the energy of the folding configuration it represents. To determine the barrier between two local minima, one just has to find the vertex in the barrier tree that has both leaves representing the local minima as transitive children and the greatest topological distance to the root of the tree.

molecules are often short lived and does not only fold after, but also during synthesis [12]. Similar effects take place, when an RNA molecule travels through a narrow pore, unfolding on one side and refolding on the other [9]. It is believed that these effects have a significant impact on the native state by selecting one of the many local minima of the folding landscape. These processes are thought of as a growing chain of nucleotides that folds while new nucleotides are added. Parameters for this process are the temperature of the system and the growing rate of the nucleotide chain. The only option to determine the final state is to fully simulate the process, as different parameters can lead to totally different solutions. For each chain length there is a folding landscape in which the current molecule folds until the next nucleotide is added.

1.2 Visualization Problem

Visualizing all of the folding landscapes may help to determine the final state independent of the simulation and the simulation parameters and help to understand the full process. The folding landscape for the new chain does have a strong similarity to the previous landscape and it makes sense to determine folding configurations in the new landscape that are most similar to configurations of the previous landscape based on structural similarities. To identify the successor of a structure that is a local minima, one would start at the structurally most similar configuration of the next landscape and travel along the gradient of this energy landscape to a new local minima. Multiple local minima may have the same successor and some local minima of the new landscape have no predecessor. These changes to the landscape can be adopted for the corresponding barrier trees, e.g. merging of local minima can be thought of as a replacement of a subtree by a leaf, creation of local minima as adding leaves or subtrees. Due to the size of the landscapes, the series of folding landscapes should be visualized by a series of barrier trees. These barrier trees share some common information that should be presented accordingly, i.e. they 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 on barrier tree that changes with time in a way that the barrier trees of the sequence are snapshots of it 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 an vector space and *routes* edges to connect the vertices. The layout of a graph has properties that can be measured with certain cost functions, like area of the layout, number of edge crossings, distribution of vertices and edges, and congruency of isomorphic structures. Esthetic criteria have been established that try to make visually pleasing drawings. Such criteria often include 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 researched in the past decades. There exist good overviews for this topic ([1, 11, 17]).

Moen [14] was one of the first to consider the dynamic tree drawing problem. However, his motivation was to present a tree that does not change except for hiding and unhiding subtrees. The change can be seen as a replacement of a subtree by a leaf and vice versa. Moen's algorithm efficiently computes the layout of the tree after such a change from layout information before the change. Cohen *et al.* [2] give detailed algorithms and data structures for a number of graph classes that are suited for graph editing systems and visualization of operations on data structures (e.g. AVL- Trees). These works mainly considered changes to graphs that result from interaction with a user in the case of graph editing and browsing and were motivated by reusing large parts of the layout in order to not having to recompute them. Reusing layout information also has a strong impact on the perceived quality of the presentation of a dynamic graph. North [15] tried to measure 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 applied his concepts to the drawing of dynamic directed acyclic graphs. Misue *et al.* [13] introduced the formal concept of *mental distance* that tries to formally described the difference of two layouts and thus the perceived stability layout of a dynamic graph. As all graph drawing is subject to certain esthetic criteria, he added the criterion “preserving the mental map” to the esthetic criteria of dynamic graph drawing. He refined this criterion: 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 does 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 one rule is broken. Frishman and Tal [7] presented an algorithm that draws dynamic clustered general graphs using an incremental force directed method. Their algorithm generally preserves the mental map, but improves the layout slightly, if a static graph drawing esthetic criteria is not met anymore. Diehl and Görg [4] propose a general scheme to layout dynamic graphs if all graphs of the sequence are known prior to layout creation. This scheme seems mostly 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 blueprint for the layout of the subgraphs. The layout of the subgraphs is further improved to meet 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 and Diehl [10] further improve their 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.

In this work we adapted this scheme. 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 an directed acyclic graph (DAG). For the layout of this supergraph we implemented and modified the *dot* algorithm by Gansner *et al.* [8].

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. A path in a graph G is a list of vertices of G, where each two successive vertices of the list are connected by an edge of G, and no vertex occurs more than one time in the list. The only exception to this is, that the first vertex is the same as the last. Such a path is called a circle. A path or circle is called directed, if all vertex connecting edges point in the same direction. 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 is a directed path from u to v in G. $odeg_G(v)$ denotes the number of edges of G that point away from 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. 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 $path_G(v,w)$ and $odeg_G(v) = 0$. 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 to a sequence of trees with leaf mappings is: given a sequence of rooted trees $T_0, \ldots, T_n$ with

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

and

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

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:

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

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

2. G conforms to the leaf mapping:

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

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

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

3.2 Motivations

The first step of the *Foresight Layout with Tolerance* algorithm [4] 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 equal. 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 of the previous tree is non trivial. This identification should not be motivated by graph theoretic minimization, but to reflect properties of the corresponding landscapes. A barrier tree not only stores barriers between local minima, it also gives a rough and abstract view on the topology of a landscape. These topological properties of the landscape can and will be used to identify inner vertices shared between barrier trees. 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 identify further inner vertices as equal, and quickly identify isomorphic structures between the barrier trees that still reflect 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 equal. This is avoided by identifying equal inner vertices based on the set of local minima that can be reached from the corresponding barrier just by descending in the landscape. In Fig. 2a, vertex e and j are considered to be equal, because the set of leaves that can be reached from them is equal under the leaf mapping. Vertices d and i are not equal because the set of leaves that can be reached from them, $\{a,b\}$ and $\{g,h\}$ respectively, are not equal under the leaf mapping. Such cases are very common and border mostly on the change of the height of barriers between successive trees. The supergraph in that case is 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 equal to any vertex of tree $t+1$, but equal 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 $t+1$ 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 equal 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 equal ($\{a,b\}$ vs. $\{d\}$). The solution is to temporarily add the mirror vertices a' and b' as children to d and modify the leaf mapping. Now c and d will be considered equal. This method also works, 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 equal to both vertices. In that case the vertex that is farthest from the root (c) of the tree 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 $t+1$. The removal of transitive edges has little to no effect on the quality of the

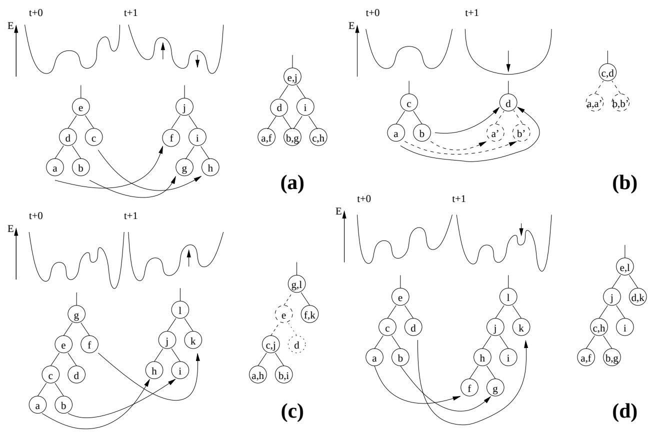


Figure 2: examples of elementary landscape and barrier tree changes
Each figure shows how the energy landscape changes, illustrates the barrier trees (only the topology is shown) and the leaf mappings and show how the supergraph should look like for the following cases: barrier swap (a), leaf merging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

final presentation but reduces the size of the supergraph and greatly improves the performance when the layout of the supergraph is determined. In Fig. 2d a leaf is added to the tree. This is the direct opposite 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 $t + 0$.

These operations are considered elementary and are the only operations we observed in our datasets. However, it is expected that multiple elementary operations happen between successive trees of the sequence. While creation, deletion, and merging can not happen to the same leaves of a tree simultaneously, these operations and the modification they imply on the supergraph do not affect each other. Also creation, deletion, and merging happen at or near the leaves, while *barrier swaps* can only modify 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:

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

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.

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:

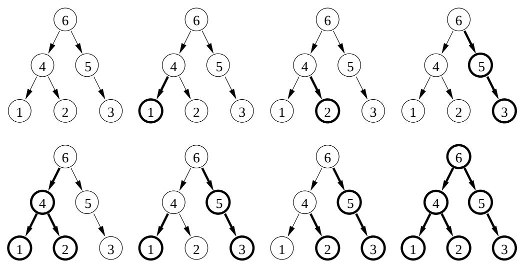


Figure 3: Example for the mark$_G$ function
The result is illustrated by vertices with thick lines.
top left to right: $\mathrm{mark}_G(\emptyset)$, $\mathrm{mark}_G(\{1\})$, $\mathrm{mark}_G(\{2\})$, $\mathrm{mark}_G(\{3\})$.
bottom left to right: $\mathrm{mark}_G(\{1,2\})$, $\mathrm{mark}_G(\{1,3\})$,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

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 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. $l_i(v)$ contain the most similar vertices.

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

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

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

The inserted edges are:

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

Transitive edges may be removed:

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

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 includes some code fragments that illustrate implementation details for the operations needed for the supergraph construction.

3.4 Example

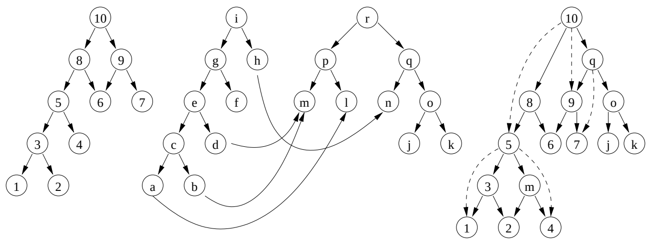


Figure 4: Example construction of the supergraph of two trees left to right: the supergraph G_i, the tree T_i, the tree T_{i+1} and the supergraph G_{i+1}. Arrows between T_i and T_{i+1} indicate the leaf mapping. The dashed lines in G_{i+1} indicate edges that can be replaced by a path. The exact description, 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 on 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:

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

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:

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

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 Postprocessing

Unfortunately, the use of the supergraph as a data structure to find similar leaf sets, does not find the smallest supergraph in the general case. Some edges are inserted to ensure correct results for the *match* and *mark* functions, but could be removed after the supergraph of the full sequence is generated, without violating the condition that each tree should be contained in the supergraph. It is difficult to track these redundant edges, because an edge may be deemed superfluous when inserted, but may become necessary, when further trees are merged with the supergraph. Such 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 trees it occurs in. This information will be used in the following layout phase. Usually tree edges are replaced by paths in the supergraph. In that case 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 path with the highest number of edges is annotated. After annotating there will be many edges which do not belong to any tree. These can be safely removed. 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 contains each tree. 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.

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 of the tree sequence will be a directed acyclic graph (DAG). Different strategies have been proposed to layout a DAG. Sugiyama *et al.* [16] 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. Gansner *et al.* [8] gives detailed heuristics for each of the phases. In this work we adopted his heuristics to lay out the supergraph.

The main esthetic goal for DAG layout is the removal of edge crossings. In practice, the supergraph can contain a large number of edge crossings, but most of them do not matter for us, because the crossing edges may never be shown simultaneously. The postprocessing phase of the supergraph construction not only decreases the number of edges and thus the possibility of an edge crossing, by annotating each edge with the trees it occurs in, we can weight the importance of an edge crossing. In the *dot* [8] algorithm, graphs are laid out respecting edge weights. Edges with a high weight are kept short and crossing free. The weight of an edge is generated from the number of trees it occurs in. For the phase of crossing removal, instead of this weight, the weight of an edge is replaced by the weight of a 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.* [8] 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 directly. 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 process similar to simulated annealing. A temperature, initially 1, is used as the probability of performing a switch that does not improve the number of edge crossings directly. Improvements are always and deterioration never accepted. Each global iteration the system cools down, the temperature decreases exponentially. The process terminates, if no more switches are performed. 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 neglected. 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 be associated with 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 has to be done for each tree separately, respecting the order generated in the *ordering* phase. This constraint preserves the *mental map*, more specifically the *orthogonal ordering*. Positioning each tree separately allows us to locally improve the layout of the subgraphs. This is also the third phase of the *Foresight Layout with Tolerance* algorithm.

The original formulation of the algorithm generates one coordinate from the number of the layer, that a vertex resides in. Because the edges of a barrier tree point from vertices with higher energy to vertices of lower energy, and the construction of the supergraph and the assignment to layers preserve this orientation, the coordinate, that is best suited to be replaced with the coordinate generated from the energy of an vertex, is the one, that would have been generated from the layer number. The other coordinate, usually the horizontal one, is generated by shifting vertices inside the layers without switching their order and minimizing the global edge length. The original formulation of this phase of the layout algorithm ensures that no two vertices of the same layer will have the same horizontal position. But if the vertical coordinate is generated from the energy of the vertices, it is possible for two vertices from different layers to overlap, if they have been assigned to the same horizontal position and have the same energy. Since we do not have the freedom of changing the energy/ vertical position of a vertex, one way to avoid this, is to assign an unique horizontal position to each vertex independent of what layer it may belong to. The simplest way to do this, is to perform the positioning of vertices for the whole supergraph as in the original formulation of the layout, sorting the vertices according to their horizontal position, and using their order in the sorted list as their new horizontal position. That way no two vertices overlap, but this results in a bad coverage of the drawing area and does not allow further local improvements. A different strategy is to force all sinks of the supergraph to be in the same layer and performing the *positioning* for the whole supergraph. This ensures that at least the sinks of the supergraph will have an unique horizontal position. As most leaves of the barrier trees are sinks of the supergraph, and they are the most interesting vertices, this results in better drawing due to a more efficient usage of the drawing area. However this part of the algorithm still leaves room for improvements as there is still no local improvement of the layout in the horizontal coordinate.

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 this 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 *orthogonal ordering*, 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 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 in a similar manner, based on the changes of their adjacent vertices. Subtrees that are created or merged “grow” out or into the vertices, where they are created or merged into, again using linear interpolation of the coordinates. Usually the 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 this 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 two segments overlap intentionally. Because vertices are not drawn explicitly, but are implied by the point, where edges join, this preserves the mental map better. 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. The last page of this article shows the keyframes for this dataset. The full animation is part of the additional material to this paper. 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 grows 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. A different perspective is estimating the minimum supergraph. This supergraph would be created if there were no *barrier swaps* in the barrier tree sequence, only addition, deletion and merging. The minimum supergraph has the same number of sinks as the generated supergraph, because each sink was once a leaf in the tree and the equivalence of leaves can be trivially determined based on the leaf mappings. The supergraph must be connected, i.e. may not be split and it can be shown, that the property $odeg_{T_i}(v) = 2$ for each inner vertex of the barrier tree results in the property $odeg_G(v) = 2$ for each inner vertex of the minimum supergraph. If N is the number of sinks in G, then G must have at least $2N-1$ vertices. The number of leaves in the supergraph for the *att*, *lepto* and *hok* datasets are 110, 332 and 1035 respectively. The leaves already make up a large part of the supergraph and it can be seen, that the supergraphs are no bigger than three times the minimal solution (*att*: 1.79x, *lepto*: 2.82x, *hok*: 2.22x). The cause of this are mainly *barrier swaps*.

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. If there are no edge crossings in the supergraph, there will not be edge crossings in the layout of the subgraphs. 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 in the datasets. The animation suffers from long edges that are close together and make it difficult to track edges.

Various 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. Barriers are not that important, and 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*, thus the supergraph is made up of 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 down to 2419 and 5863 respectively and the number of supergraph vertices down to 853 and 2493 respectively. This means that 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, so 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.

Another 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 from the viewpoint of folding landscapes. 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. The minimal supergraph did not get significantly smaller but the generated supergraphs are only twice as big as the minimal one. 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 improves the supergraph layout. There are a lot less edge crossings and only a few long edges. This directly results in a better layout of the barrier trees.

7 Conclusion and Future Work

It is possible to generate a readable dynamic layout for sequence 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 dynamic 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 also preserves the mental map for the annotations. The change in the densities can be additionally indicated by a flow of liquid along the lines of the tree edges. Such methods, that combine tree layout and additional information, are currently investigated.

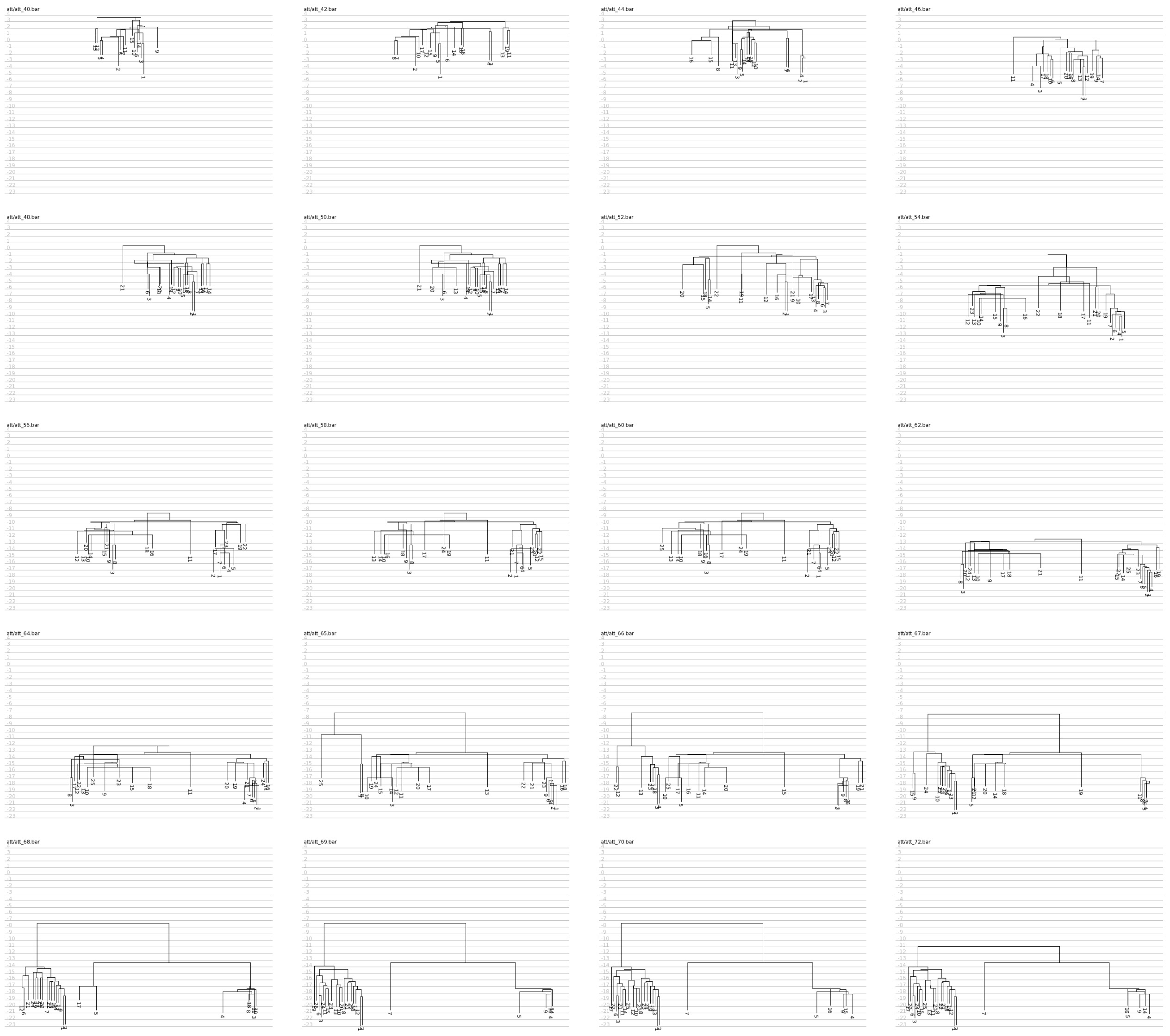
The current methods to generate the animation leave room for further improvements. Different strategies for edge removal during the postprocessing of the supergraph construction can result in an improved layout, because fewer edges generally result in fewer edge crossings. Rather than overgenerating 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 leading to resolution issues. 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. The supergraph is already such a static presentation. Other ideas include the generation of a synthetic energy landscape from all barrier trees, where the folding process is visualized as a walk in this landscape.

References

[1] 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.

[2] 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.



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

[4] Stephan Diehl and Carsten Görg. Graphs, they are changing - dynamic graph drawing for a sequence of graphs. In M. T. Goodrich and S. G Kobourov, editors, *Proceedings of 10th International Symposium on Graph Drawing*, number 2528 in Lecture Notes in Computer Science, LNCS, pages 23–31, 2002.

[5] Christoph Flamm. *Kinetic Folding of RNA*. PhD thesis, University of Vienna, Austria, 1998.

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

[7] Yaniv Frishman and Ayellet Tal. Dynamic drawing of clustered graphs. In *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS'04)*, pages 191–198, 2004.

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

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

[10] Carsten Görg, Peter Birke, Mathias Pohl, and Stephan Diehl. Dynamic graph drawing of sequences of orthogonal and hierarchical graphs. In *Proceedings of 12th International Symposium on Graph Drawing*, 2004.

[11] Ivan Herman, Guy Melançon, and M. Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

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

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

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

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

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

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