Tree decomposition

The exposition is based on the following sources, which are all recommended reading:

1. Kleinberg, Tardos: Algorithm Design, Addison Wesley, 2006
2. Cai et al: Rapid *ab initio* RNA Folding Including Pseudoknots via Graph Tree Decomposition.

Introduction

It is often the case that problems that are generally hard to solve, can be solved quite efficiently if the input has a certain structure. For example finding longest paths in a general graph is NP-hard, whereas finding a longest path in a DAG can be solved by a simple DFS traversal.

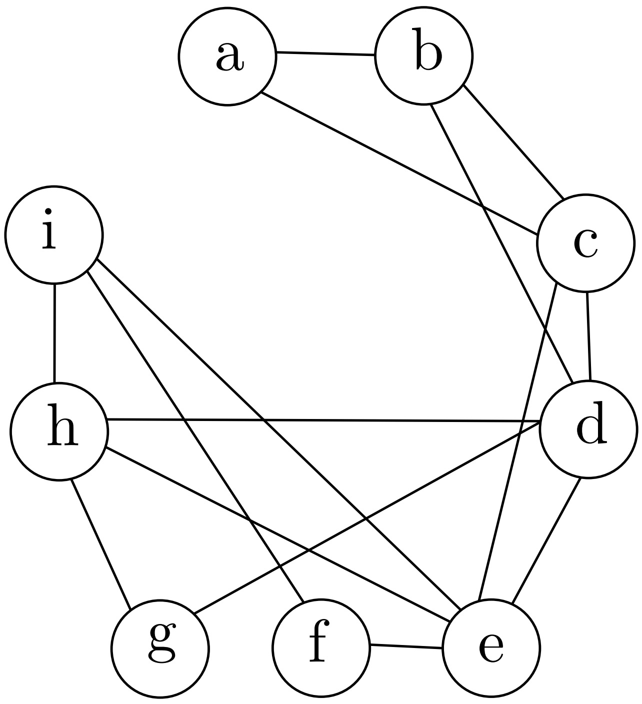
Many algorithms on graphs become easy if the input graph is a tree. For example computing the maximal independent set on a graph can be computed in linear time if the graph is a tree.

Introduction (2)

The question is now whether one can take advantage of those well performing algorithms on input that are *almost* good, that means for example almost a tree.

The notion of being *almost* a tree can be formalized using the concept of *tree width*. If the tree width of a graph is small, then it is *tree-like*. In particular, a tree has tree-width 1. Tree width is defined using the concept of *tree decomposition*.

Definition (3)



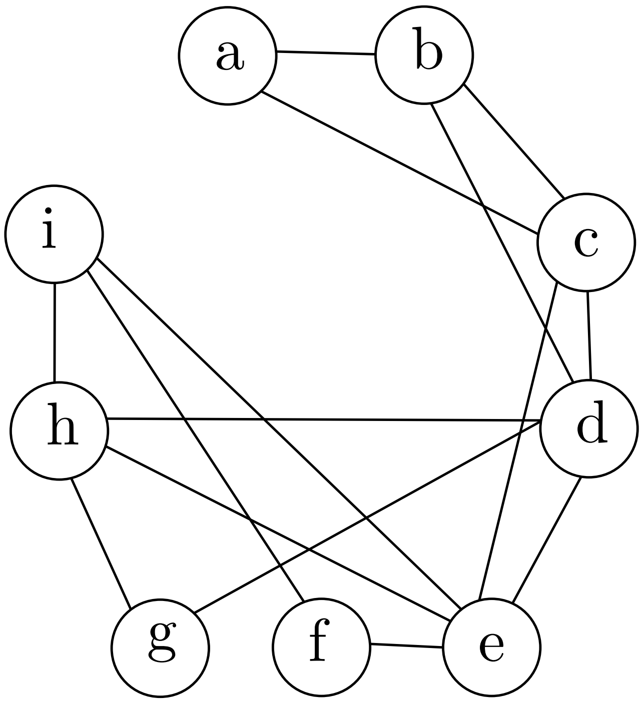
Is this graph tree-like?

Definition

Formally, a *tree decomposition* of $G = (V, E)$ consists of a tree T and a subset $V_t \subseteq V$ associated with each node $t \in T$. We will call the subsets V_t *pieces* of the tree decomposition. T and $\{V_t : t \in T\}$ must satisfy:

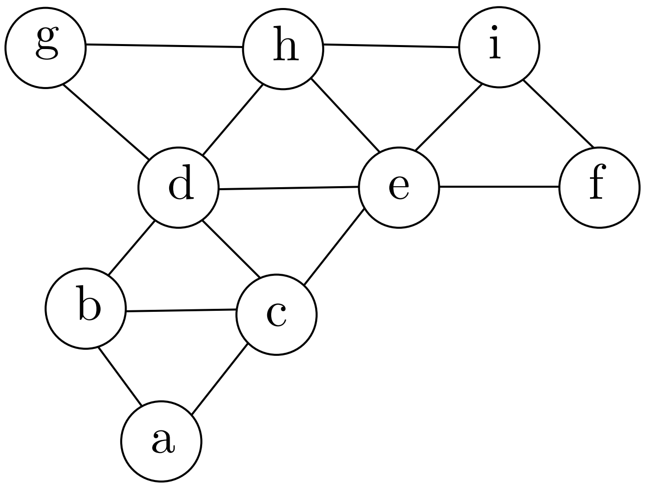
- *(Node coverage)* Every node of G belongs to at least one piece V_t.
- *(Edge coverage)* For every edge e of G, there is some piece V_t containing both ends of e.
- *(Coherence)* Let t_1, t_2 and t_3 be three nodes of T such that t_2 lies on the path from t_1 to t_3. Then, if a node v of G belongs to both V_{t_1} and V_{t_3}, it also belongs to V_{t_2}.

Definition (2)



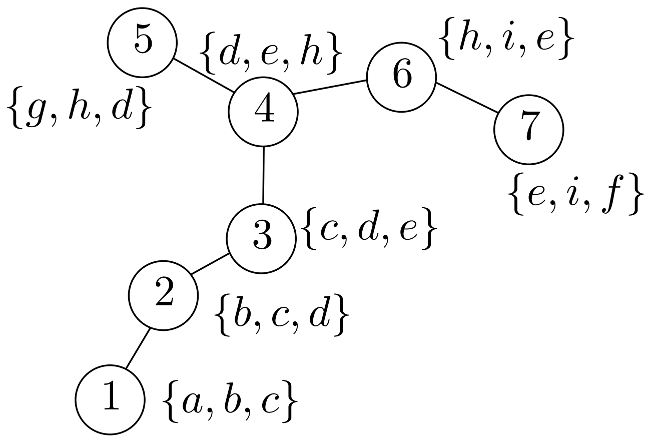
Is this graph tree-like?

Definition (3)



I draw it differently.

Definition (4)



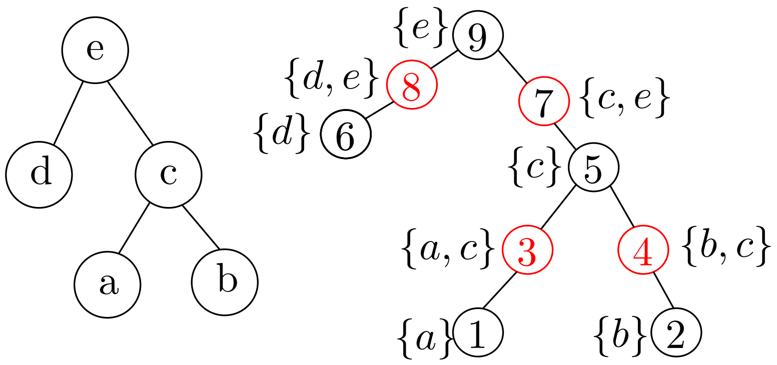
Here is a possible tree decomposition. Check the definition.

Definition (5)

If the input graph G is a tree, we can even give a simple algorithm to compute a tree decomposition $(T, \{V_t : t \in T\})$.

1. Define a node t_v for each $v \in G$.
2. Define a node t_e for each $e \in G$.
3. T has an edge (t_v, t_e) when v is an end of e in G.
4. Define the pieces $V_{t_v} = v$ for all t_v and $V_{t_e} = \{u, v\}$ for all $e = (u, v)$.

Definition (6)



We will later see an algorithm to construct a TD for general graphs.

Properties of tree decompositions

First we will look at some properties of TDs. The three conditions in the definition of TD ensure that the collection of pieces in the TD corresponds to G.

Node coverage and edge coverage ensure the "existence" of all nodes and edges of G.

Properties of tree decompositions (2)

The coherence condition ensures tree-like separation properties (1) if you delete an edge e from a tree, it falls apart into two connected components. 2) if you delete a node from a tree it corresponds to deleting all incident edges.)

The coherence property is designed to guarantee that separations of T correspond to separations of G as well.

Properties of tree decompositions (3)

If T' is a subtree of T we use $G_{T'}$ to denote the subgraph of G induced by the nodes in all the pieces associated with T', that is the set $\cup_{t \in T'} V_t$.

Consider deleting a node $t \in T$.

Theorem 1. *Suppose that $T - t$ has components $T_1, \ldots, T_d$. Then the subgraphs*

$$G_{T_1} - V_t, G_{T_2} - V_t, \ldots, G_{T_d} - V_t$$

have no nodes in common, and there are no edges between them.

Properties of tree decompositions (4)

Proof: Assume there is a node v that belongs to $G_{T_i} - V_t$ and $G_{T_j} - V_t$ for some $j \neq i$. Then, by the node coverage property there must be pieces V_x with $x \in T_i$ and V_y with $y \in T_j$. Since t lies on a $x - y$-path in T, it follows from the coherence property that $v \in V_t$. Hence v belongs to neither $G_{T_i} - V_t$ nor $G_{T_j} - V_t$.

Now assume there is an edge $e = (u, v)$ in G with u in $G_{T_i} - V_t$ and v in $G_{T_j} - V_t$ for some $i \neq j$. If there were such an edge, then by the edge coverage property there must be a piece V_x containing both u and v. Obviously x cannot be in both T_i and T_j.

Assume $x \notin T_i$. But u is in $G_{T_i} - V_t$ and hence in some piece V_y for $y \in T_i$. Then u belongs to both V_y and V_x and since t lies on a $x - y$-path in T it follows by coherence that $u \in V_t$ which again leads to a contradiction.

Properties of tree decompositions (5)

In a similar fashion you can prove the following theorem in which we assume the deletion of an edge $e = (x, y)$ from T.

Theorem 2. *Let X and Y be the two components of T after the deletion of the edge (x, y). Then deleting the set $V_x \cap V_y$ from V disconnects G into the two subgraphs $G_X - (V_x \cap V_y)$ and $G_Y - (V_x \cap V_y)$. More precisely, these two subgraphs do not share any nodes, and there is no edge with one end in each of them.*

Proof: (exercises).

Properties of tree decompositions (6)

Tree decompositions are useful in that the separation properties of T carry over to G. A this point one might think that the key question is: Which graphs have tree decompositions?

However, this is not the point, because every graph has a TD (Why?).

The crucial point is rather whether there exists a TD in which all the pieces are *small*. This is really what we try to carry over from trees, by requiring that the deletion of a very small set of nodes breaks the graph into disconnected subgraphs.

Properties of tree decompositions (7)

So we now formally define:

Definition 3. The *width* of a TD $(T, \{V_t\})$ is

$$\mathit{width}(T, \{V_t\}) = \max_t \mid V_t \mid -1.$$

The *tree-width* of a graph G is the minimum width of any tree decomposition of G.

Note that all tree decompositions must have pieces with at least two nodes, and hence the minimum tree-width is 1. (The -1 in the definition is somewhat arbitrary and ensures that a tree has tree-width 1).

Properties of tree decompositions (8)

Indeed the following holds:

Theorem 4. *A connected graph G has tree-width* 1 *if and only if it is a tree.*

Proof: exercise. Hint: Use the previous theorem.

Another useful observation is, that we can often reduce the number of pieces without changing the width of a TD. If we see an edge $(x, y) \in T$ such that $V_x \subseteq V_y$, then we can contract the edge (x, y) and obtain a smaller TD with the same width (check our construction for the TD of a tree, it can be reduced in size).

Construction of a TD

Finally we turn to an algorithm that constructs a tree decomposition. The problem of determining the tree-width of a graph is unfortunately NP-hard.

However, we are only interested in graphs were the tree-width is a small constant. For this case we will give an algorithm with the following guarantee:

Given a graph G of tree-width less than w, it will produce a TD of G of width less than $4w$ in time $O(f(w) \cdot mn)$, where m and n are the number of edges and nodes of G and $f(\cdot)$ depends only on w.

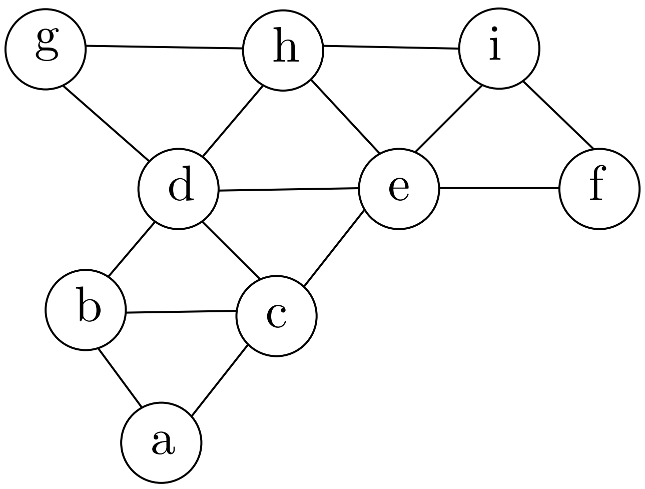
Construction of a TD (2)

First we define a local structure in a graph that is an obstacle for having a TD with small width. Our algorithm will then either construct the desired TD, or it will detect the local substructure which implies that there is no TD of small width.

Given two sets $Y, Z \subseteq V$ of the same size, we say they are *separable* if some strictly smaller set can completely disconnect them. Specifically there is a set $S \subset V$ and there is no path from $Y - S$ to $Z - S$ in $G - S$.

A set X of nodes in G is *w-linked* if $\mid X \mid \geq w$ and X does not contain separable subsets Y and Z, such that $\mid Y \mid = \mid Z \mid \leq w$.

Definition (3)



In this graph the sets $X = \{g, d, h, e\}$ and $Y = \{h, e, i, f\}$ are separable with $S = \{h, e\}$. The set $X = \{g, d, h, e, i, f\}$ is $2 - linked$ but not $3 - linked$.

Construction of a TD (4)

The following fact will prove useful:

Theorem 5. *Let $G = (V, E)$ have m edges, let X be a set of k nodes in G, and let $w \leq k$ be a given parameter. Then we can determine whether X is $w - linked$ in time $O(f(k) \cdot m)$, where $f(\cdot)$ depends only on k. Moreover, if X is not $w - linked$, we can return a proof of this in form of sets $Y, Z \subseteq X$ and $S \subset V$ such that $\mid S \mid < \mid Y \mid = \mid Z \mid \leq w$ and there is no path from $Y - S$ to $Z - S$ in $G - S$.*

Construction of a TD (5)

Proof: Enumerate each pair of subsets Y, Z. Since X has at most 2^k subsets there are at most 4^k such pairs.

We can determine whether Y and Z are separable checking whether there are $l = \mid Y \mid = \mid Z \mid$ node-disjoint paths each with one end in Y and one end in Z. This can be done by computing a flow with unit capacities in time $O(lm)$.

One should imagine a w-linked set as being self-entwinded. It has no small parts that can be easily split off from each other.

Construction of a TD (6)

We will use later the following theorem:

Theorem 6. *If G contains a $(w+1)$-linked set of size at least $3w$, then G has tree-width at least w.*

Proof: Suppose, by way of contradiction, that G has a $(w+1)$-linked set X of size at least $3w$, and it also has a nonredundant TD $(T; \{V_t\})$ of width less than w. The idea of the proof is to find a piece V_t that is "centered" with respect to X, so that when some part of V_t is deleted from G, one small subset of X is separated from another.

(rest: exercise)

Construction of a TD (7)

We give now a greedy algorithm for constructing a tree decomposition of low width. The algorithm will not precisely determine that tree-width of the input graph $G=(V,E)$; rather, given w, either it will produce a TD of width less than $4w$, or it will discover a $(w+1)$-linked set of size at least $3w$.

In the latter case, this constitutes proof that the treewidth of G is at least w by applying the above theorem.

We start by assuming that there is now $(w+1)$-linked set of size at least $3w$. Then our algorithm will produce a tree decomposition.

Construction of a TD (8)

We will continously and greedily grow a set $U \subset V$ in each step of our algorithm and maintain the property that we have a *partial tree decomposition*. With partial TD we mean, that if $U \subseteq V$ denotes the set of nodes of G that belong to at least one of the pieces already constructed, then our current tree T together with all the pieces constructed should form a TD on the subgraph of G induced by U.

Now we have to make sure that all the pieces have size at most $4w$ in order to achieve our goal of having a treewidth of at most $4w$.

Construction of a TD (9)

If C is a connected component of $G-U$, we say that $u \in U$ is a *neighbor* of C if there is some node $v \in C$ with an edge to u. The key behind the algorithm is not simply to maintain a partial TD of width less than $4w$, but also to make sure that following invariant (*) is enforced the whole time:

At any stage of the execution of the algorithm, each component C of $G-U$ has at most $3w$ neighbors, and there is a single piece V_t that contains all of them.

Now we describe how to add a new node and a new piece so that we still have a partial TD, the invariant (*) is fulfilled, and U has grown strictly larger. (The code is without the enclosing loop).

Construction of a TD (10)

Construction of a TD (11)

Lets compute a TD for our initial example graph of treewidth 2 and lets see what the greedy algorithm comes up with.

```
(1)  greedyTDcore($G$,w);
(2)  choose arbitrarily $v \in G$;
(3)  $U = \{v\}$;  insert $t$ into $T$;  $V_t = \{v\}$;
(5)  let $C_1, C_2, \dots, C_k$ be the connected comp. of $G - U$;
(6)  choose arbitrarily a $C_j$;
(7)  $X_j = \{u \in U \mid u$ is a neighbor of a $v \in C_j\}$;
(8)  $V_t =$ the piece in $T$ that contains all of $X_j$;
(9)  if $\mid X_j \mid < 3w$
(10)   then
(11)       choose a $v \in C_j$ that neighbors $X_j$;
(12)       $U = U \cup \{v\}$;  insert $s$ into $T$;  $V_s = X \cup \{v\}$;
(13)       add edge in $T$ from $t$ to $s$;
(14)   else // $\mid X_j \mid$ is exactly $3w$
(15)       if $X_j$ is a $(w+1)$-linked set
(16)         then output $G$ has tree width at least $w$;
(17)         else proceed using proof of not being $(w+1)$-linked;
(18)       fi
(19) fi
```

Construction of a TD (12)

Now suppose, the graph does not fall apart and X has exactly $3w$ nodes. In this case it is less clear how to proceed. If we add an additional node of a connected component C to X and end up with a component that has all nodes of X as neighbors, then the invariant (*) is violated.

There is no simple way around this. G might actually not have a low tree width. Hence it makes sense to ask whether X is a $(w+1)$-linked set. Using the previous theorem we can determine this in time $O(f(w) \cdot m)$ since $\mid X \mid = 3w$.

If X is a $(w+1)$-linked set then we are done, since G has then tree-width of at least w, which is an acceptable output of our algorithm.

Construction of a TD (13)

If X is not $(w+1)$-linked, then we end up with $Y, Z \subseteq X$ and $S \subseteq V$ such that $\mid S \mid < \mid Y \mid = \mid Z \mid \leq w+1$ and there is no path from $Y - S$ to $Z - S$ in $G - S$. The sets Y, Z and S will now provide us with a means to extend out partial TD.

Let S' consist of the nodes in S that lie in $Y \cup Z \cup C$. We observe that $S' \cap C$ is not empty. (Y and Z each have edges into C, and so if $S' \cap C$ were empty, there would be a path from $Y - S$ to $Z - S$ in $G - S$ that started in Y, jumped into C, traveled through C and jumped back into Z.) Also $\mid S' \mid \leq \mid S \mid \leq w$.

Construction of a TD (14)

We define a new piece $V_s = X \cup S'$, making s a leaf of t. All the edges from S' into U have their ends in X and $\mid X \cup S' \mid \leq 3w + w = 4w$, so we still have a partial TD. Moreover, the set U has grown, since $S' \cap X \neq \emptyset$. So we are done when we can show that our invariant (*) still holds.

Our partial TD now covers $U \cup S'$; and where we previously had a component C of $G - U$, we now may have several components $C' \subseteq C$ of $G - (U \cup S')$. Each of these components C' has all its neighbors in $X \cup S'$. So we must make sure that there are at most $3w$ such neighbors.

Construction of a TD (15)

Consider one such component C'. We claim that all of its neighbors in $X \cup S'$ actually belong to one of the two subsets $(X - Z) \cup S'$ or $(X - Y) \cup S'$, and each of these sets has size at most $| X | \leq 3w$.

For, if this did not hold, then C' would have a neighbor in both $Y - S$ and $Z - S$ contradicting the construction of the sets. Hence it holds and hence (*) still holds after the addition of the new set.

Using TD in Bioinformatics: An example

Cai and colleagues employ TD in a number of bioinformatics problems. To show you its use in a real world example we present their paper:

Rapid *ab initio* RNA Folding Including Pseudoknots via Graph Tree Decomposition.

We start by giving an introduction to the problem, give the formal definition in terms of graphs (the problem can be reduced to the weighted independent set problem), and give an algorithm to compute an optimal solution.

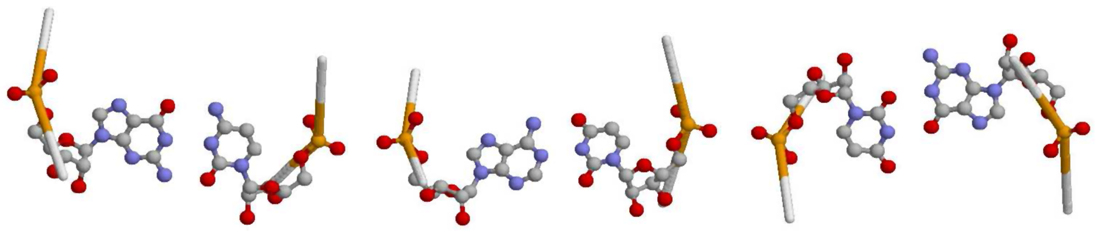
Using TD in Bioinformatics: An example (2)

The standard algorithms for computing an RNA folding with minimum free energy are by Zuker or use McGaskill's partition function (see script "Algorithmic Bioinformatics").

Basically they compte a loop decomposition where the individual loops are weighted by an energy term. However, these algorithms do not assume the presence of pseudoknots.

RNA secondary structure

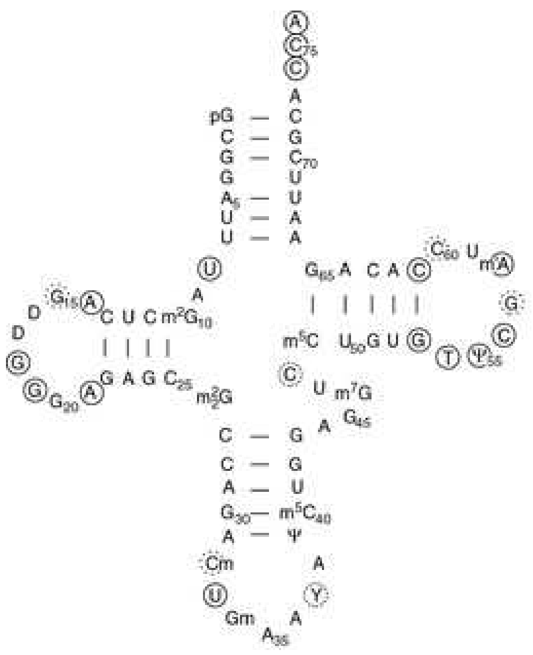
Unlike DNA, RNA is single stranded. However, complementary bases $\mathtt{C}-\mathtt{G}$ and $\mathtt{A}-\mathtt{U}$ form stable *base pairs* with each other using hydrogen bonds. These are called *Watson-Crick* pairs. Additionally, one sometimes considers the weaker $\mathtt{U}-\mathtt{G}$ *wobble pairs*. These are all called *canonical base pairs*. Wobble base pairs have one hydrogen bond less than $G-C$ base pairs.

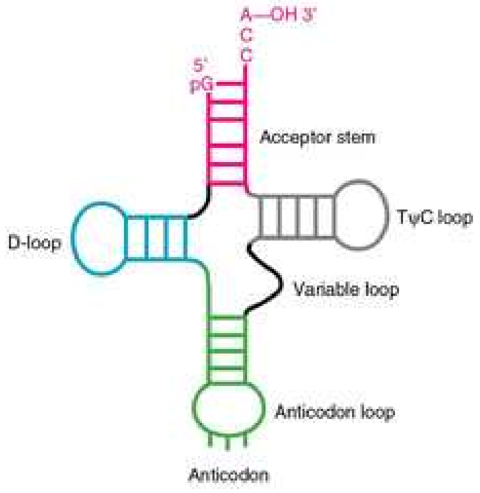


(source: Lyngsø)

RNA secondary structure (2)

When base pairs are formed between different parts of a RNA molecule, then these pairs are said to define the *secondary structure* of the RNA molecule. Here is the secondary structure of a tRNA:



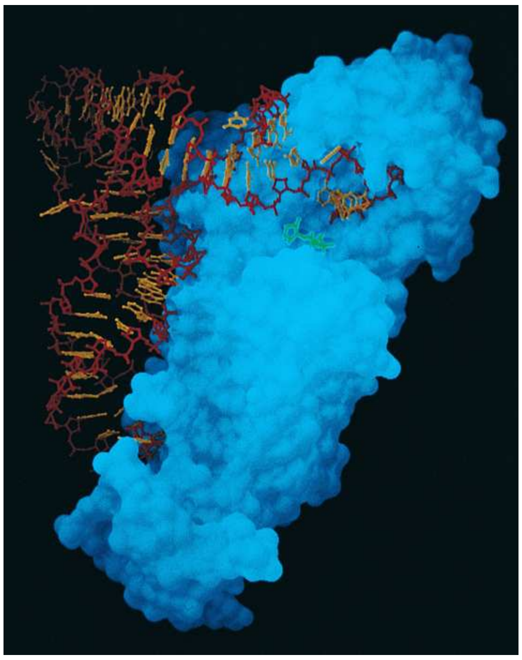


This particular tRNA is from yeast and is for the amino acid phenylalanine.

(source: http://www.blc.arizona.edu/marty/411/Modules/ribtRNA.html)

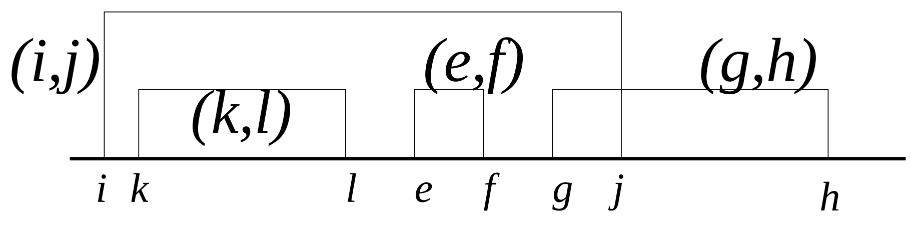
RNA secondary structure (3)

The three dimensional structure is much less packed than that of a protein:



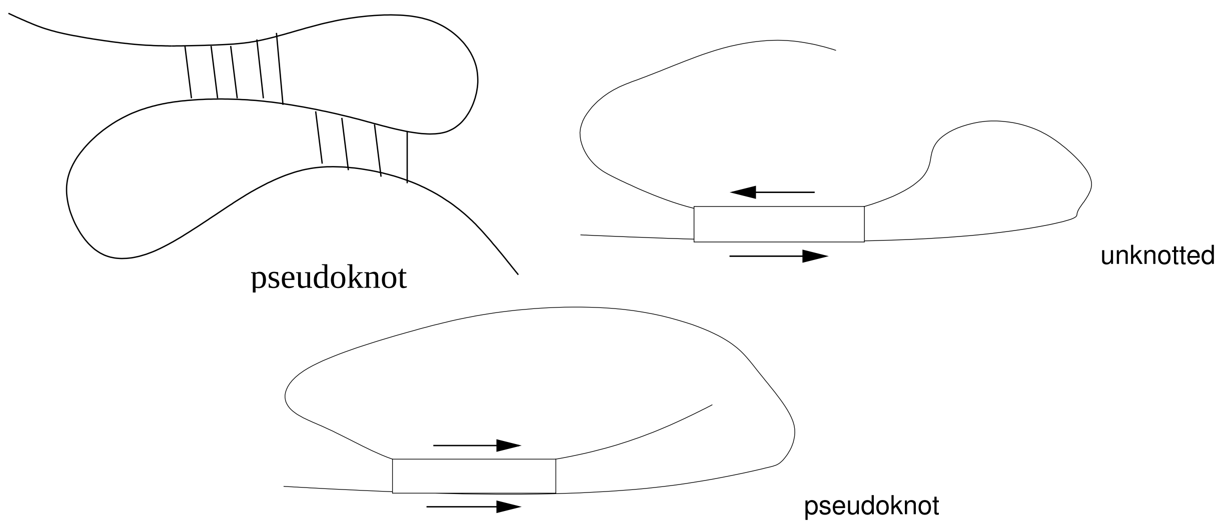
Nested structures and pseudoknots

Usually only *nested* secondary structures are considered, as the more complicated non-nested structures imply computationally harder problems.



Here, the interactions (i,j) and (g,h) are not nested.

Interactions that are not nested give rise to a *pseudoknot* configuration in which segments of sequence are bonded in the “same direction”, or have a three dimensional contact.



Nested structures and pseudoknots (2)

Practical approaches to cope with the computational challenge posed by pseudoknots are either to restrict the class of pseudoknots under consideration or to employ heuristics in the algorithms.

Optimal algorithms for restricted pseudoknot classes are usually thermodynamics-based extensions of Zuker’s algorithm for the prediction of pseudoknot-free structures. In such algorithms, the predicted optimal structure of a single RNA sequence is the one with the global minimum free energy based on a set of experimentally determined parameters.

Nested structures and pseudoknots (3)

Among these algorithms, PKNOTS can handle the widest classes of pseudoknots. However, its time complexity $O(n^6)$ makes it infeasible to fold RNA sequences of a moderate length. The computational efficiency may be improved at the cost of further restricting the structure of pseudoknots, but still with a time complexity $O(n^5)$ or $O(n^4)$.

Another drawback of most such algorithms is that they produce only the optimal solution, while suboptimal ones that may reveal the true structure are often ignored.

Nested structures and pseudoknots (4)

On the other hand, computationally efficient heuristic methods have also been explored to allow unrestricted pseudoknot structures.

Iterated loop matching (ILM) is one such method. It finds the most stable stem, adds it to the candidate secondary structure and then masks off the bases forming the stem and iterates on the remaining sequence segments until no other stable stem can be found. One structure is reported at the end.

Nested structures and pseudoknots (5)

Another algorithm, HotKnots, does the prediction in a slightly different way. It keeps multiple candidate structures rather than only one and builds each of them in a similar but more elaborate way.

These methods can usually be fast, yet they often do not provide an optimality guarantee for the predicted structure or a quality measure on the predicted structure with respect to the optimal structure. Other heuristic methods based on genetic algorithms and Monte Carlo simulation usually do not address the optimality issue either.

Problem definition

A *(canonical) base pair* is either a Watson-Crick pair (A-U or C-G) or a *wobble* pair G-U. A *stem* is a set of stacked nucleotide base pairs on an RNA sequence s. In general a stem S can be associated with four positions (i^l, j^l, i^r, j^r), where $i^l < j^l < i^r < j^r$, on the sequence s such that

1. $(s[i^l], s[j^r])$ and $(s[j^l], s[i^r])$ are two canonical base pairs; and
2. for any two base pairs $(s[x], s[y])$, $(s[z], s[w])$ in the stem S, either $i^l \leq x < z \leq j^l$ and $i^r \leq w < y \leq j^r$, or $i^l \leq z < x \leq j^l$ and $i^r \leq y < w \leq j^r$.

Problem definition (2)

Region $s[i^l..j^l]$ is the *left* region of the stem and $s[i^r..j^r]$ is the *right* region of the stem. Stem S is *stable* if the formation of its base pairs allows the thermodynamic energy $\Delta(S)$ of the stem to be below a predefined threshold parameter $E < 0$.

The following table shows all the stable stems in Ec_Pk4 with $E = -5 kcal/mol$, the fourth pseudoknot in E.coli tmRNA, and their corresponding free energy values.

Problem definition (3)

ID	L	R	En
1	1..10	27..36	−12.4
2	5..10	47..52	−7.8
3	1..7	34..40	−6.8
4	9..12	22..25	−6.2
5	10..13	27..30	−5.5
6	11..14	19..22	−5.4
7	12..14	23..25	−5.4
8	23..25	35..37	−5.4
9	15..18	30..33	−5.3
10	27..29	34..36	−5.3

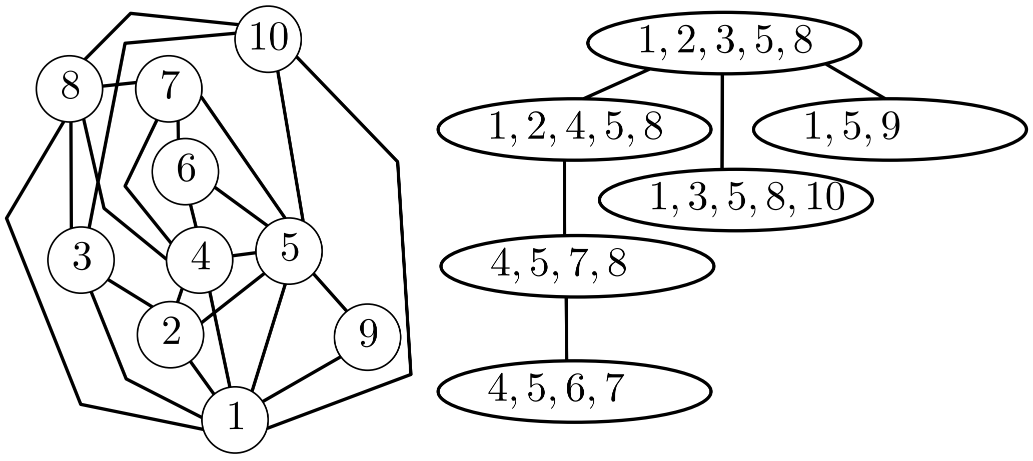
Problem definition (4)

A *stem graph* $G = (V, E)$ can be defined for the RNA sequence s, where each vertex in V uniquely represents a stable stem on s, and E contains an edge between two vertices if and only if the corresponding two stems (a, b, c, d) and (x, y, z, w) *conflict* in their positions, i.e., one or both of the regions $s[a..b]$ and $s[c..d]$ overlap with at least one of the regions $s[x..y]$ and $s[z..w]$.

Problem definition (5)

The following Figure shows the stem graph for Ec_Pk4 constructed according to the stable stems given in the above table together with a TD. The stem graph is a weighted graph, with a weight on every vertex. Usually, the weight of a vertex can simply be the absolute value of the thermodynamic energy $\Delta(S)$ of the stem S corresponding to the vertex.

Problem definition (6)



Problem definition (7)

The problem of predicting the optimal structure of the RNA then corresponds to finding a collection of non-conflicting stems from its stem graph which achieves the maximum total weight.

This is exactly the same as the graph theoretic problem: finding the maximum WIS in the stem graph. Note the weight for an IS representing a secondary structure is based on the total energies of the stems only (similar models were previously adopted by both primitive methods and more elaborate ones).

Identifying stable stems

Stable stems are defined according to a set of parameters. In particular, a stem contains at least P base pairs; the loop length in between the left and right region of the stem is at least L; the thermodynamic energy is at most E.

Bulges within a stem are allowed, for which the stem essentially becomes a set of substems separated by the bulges. In addition, parameter T limits the minimum substem length, and parameter B limits the maximum bulge length.

Identifying stable stems

The thermodynamic energy $\Delta(S)$ of stem S is calculated by taking into account both the stacking energies and the destabilizing energies caused by bulges.

The stable stem pool can be extended by introducing maximal substems that can resolve the conflicts and meet the requirements defined by the above parameters for each pair of overlapping stems in the pool.

An algorithm based on TD (2)

Maximum WIS can be used via TD (see chapter 10.4 of Kleinberg, Tardos). We will give this algorithm now instead of the one described by Cai.

To understand the algorithm we first have to see how computing a WIS works if the input is a tree. This can be easily done by dynamic programming. We construct subproblems by rooting the tree at an arbitrary node r. We start then at the leaves and work our way up the tree.

An algorithm based on TD (3)

We observe that we need to distinguish two cases in order to obtain a maximum WIS S for the tree T_u. Either we include node u in S or we do not.

If we include u we cannot include any of its children. If we do not include u we have the freedom to include or omit these children. This suggests that we use two tables in our DP.

The subproblem $opt_{in}(u)$ will denote the maximum weight of an IS of T_u including u and $opt_{out}(u)$ will denote the maximum weight of an IS of T_u that does not include u.

An algorithm based on TD (4)

It should be clear that following code solves the problem:

```
(1)  WIS($T$);
(2)  root the tree at a node $r \in T$;
(3)  for  all nodes $u$ of $T$ in post-order
(4)     do if  $u$ is a leaf
(5)           then $M_{out}[u] = 0$; $M_{in}[u] = w_u$;
(6)           else
(7)                 $M_{out}[u] = \sum_{v \in children(u)} \max(M_{out}[v], M_{in}[v])$;
(8)                 $M_{in}[u] = w_u + \sum_{v \in children(u)} M_{out}[v]$;
(9)     fi
(10) od
```

An algorithm based on TD (5)

Now we will see how we make use of our TD to compute a maximum WIS on a graph and therefore can solve the RNA folding problem.

The algorithm for computing the WIS on a TD $(T, \{V_t\})$ is very similar to the one one that works on a tree. We root the tree T and built an independent set by considering the pieces V_t from the leaves upward. At an internal node t of T we confront the following basic question:

The optimal WIS intersects the piece V_t in some subset U, but we do not know which set U it is. So we enumerate all possibilities for U. That is in all 2^{w+1} possibilities to consider.

An algorithm based on TD (6)

Lets be more precise. We root T at the node r. For any node t let T_t denote the subtree rooted at t. Recall that G_{T_t} denotes the subgraph of G induced by the nodes in all pieces associated with nodes in T_t. As notation we also use G_t. For a subset U of V, we use $w(U)$ to denote the total weight of nodes in U.

We define a set of subproblems for each subtree T_t, one corresponding to each possible subset U of V_t that represents the intersection of the optimal solution with V_t. Thus for each independent set $U \subseteq V_t$, we write $f_t(U)$ to denote the maximum weight of an independent set S in G_t, subject to the requirement that $S \cap V_t = U$.

An algorithm based on TD (7)

The quantity $f_t(U)$ is undefined if U is not an IS, since in this case we know that U cannot represent the intersection of the optimal solution with V_t.

There are at most 2^{w+1} subproblems associated with each node $t \in T$. We can assume that we are working with a TD of at most n pieces, and hence there are a total of at most $2^{w+1}n$ subproblems overall.

Clearly we can determine the maximum weight of an IS in G by looking at the solutions of all the subproblems associated with the root r. We simply take the maximum over all independents sets $U \subseteq V_r$ of $f_r(U)$.

An algorithm based on TD (8)

Now we show how to build up the solutions to all subproblems via a recurrence. When t is a leaf $f_t(U)$ is equal to $w(U)$ for each independent set $U \subseteq V_t$.

Now suppose t has d children $t_1, ..., t_d$, and we have already determined the values $f_{t_i}(W)$ for each child t_i and each IS $W \subseteq V_{t_i}$. How do we determine the value of $f_t(U)$ for an IS $U \subseteq V_t$?

An algorithm based on TD (9)

Let S be the maximum weight IS in G_t subject to the requirement that $S \cap V_t = U$, that is, $w(S) = f_t(U)$.

The key is to understand how S looks when intersected with each of the subgraphs G_{t_i}. Let S_i denote the intersection of S with the nodes of G_{t_i}.

Then the following holds:

Theorem 7. *S_i is a maximum weight IS of G_{t_i} subject to the constraint that $S_i \cap V_t = U \cap V_{t_i}$.*

An algorithm based on TD (10)

Proof: Suppose there were an IS S_i' of G_i with the property that $S_i' \cap V_t = U \cap V_{t_i}$ and $w(S_i') > w(S_i)$. Then consider the set $S' = (S - S_i) \cup S_i'$. Clearly $w(S') > w(S)$. Also it is easy to check that $S' \cap V_t = U$.

We claim that S' is an IS in G. This will contradict our choice of S as the maximum weight IS in G_t subject to $S \cap V_t = U$. For suppose S' is not independent, and let $e = (u, v)$ be an edge with both ends in S'. It cannot be that u and v both belong to S, or that they both belong to S_i', since those are both IS. Thus we must have $u \in S - S_i'$ and $v \in S_i' - S$, from which follows that u is not a node of G_{t_i} while $v \in G_{t_i} - (V_t \cap V_{t_i})$. But then by the edge separation theorem there cannot be an edge joining u and v.

An algorithm based on TD (11)

The above theorem is exactly what we need to design our recurrence for the subproblems. It says, that the information needed to compute $f_t(U)$ is implicit in the values already computed in the subtrees.

Specifically, for each child t_i, we need simply determine the value of the maximum weight IS S_i of G_{t_i}, subject to the constraint that $S_i \cap V_t = U \cap V_{t_i}$. This constraint does not completely determine what $S_i \cap V_t$ should be, rather, it says that it can be any IS $U_i \subseteq V_{t_i}$ such that $U_i \cap V_t = U \cap V_{t_i}$. Thus the weight of the optimal S_i is equal to

$$\max\{f_{t_i}(U_i) : U_i \cap V_t = U \cap V_{t_i} \text{ and } U_i \subseteq V_{t_i} \text{ is ind.}\}$$

An algorithm based on TD (12)

Finally the value of $f_t(U)$ is simply $w(U)$ plus these maxima added over the d children, except that to avoid overcounting of nodes in U, we exclude them from the contribution of the children. Thus we have:

$$f_t(U) = w(U) + \sum_{i=1}^{d} \max\{f_{t_i}(U_i) - w(U_i \cap U) :$$
$$U_i \cap V_t = U \cap V_{t_i} \text{ and } U_i \subseteq V_{t_i} \text{ is ind. }\}$$

On the next slide we see the pseudocode.

```
(1) WIS($(T, \{V_t\})$);
(2) modify the TD to make it nonredundant;
(3) root the tree at a node $r \in T$;
(4) for all nodes $t$ of $T$ in post-order
(5)     do if $t$ is a leaf
(6)           then for each IS $U$ of $V_t$ do
(7)                     $f_t(U) = w(u)$;
(8)                 od
(9)           else for each IS $U$ of $V_t$ do
(10)                    $f_t(U) =$ determined as in above theorem;
(11)                od
(12)    fi
(13) od
(14) return $\max\{f_r(U) : U \subseteq V_r$ is ind.$\}$;
```

An algorithm based on TD (13)

An algorithm based on TD (14)

For each of the d children t_i, and for each IS U_i in V_{t_i}, we spend time $O(w)$ checking if $U_t \cap V_t = U \cap V_{t_i}$, to determine whether it should be considered. This is a total time of $O(2^{w+1}wd)$ for $f_t(U)$. Since there are at most 2^{w+1} sets U associated with t, the total time spend on node t is $O(4^{w+1}wd)$.

Finally, if we sum this over all nodes t to get the total running time. Observe that the total number of children is n. Hence the total running time is $O(4^{w+1}wn)$.

An example

Consider the example from the lecture.

An example

In conclusion, the implementation of the above algorithm has outperforms in quality some programs and is comparable to the best one (PKNOTS). However, it is much faster than PKNOTs.