Basic Research in Computer Science

BRICS DS-00-5 R. B. Lyngsø: Computational Biology

Computational Biology

Rune B. Lyngsø

BRICS Dissertation Series **DS-00-5**

ISSN 1396-7002 **March 2000**

See back inner page for a list of recent BRICS Dissertation Series publications. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `DS/00/5/`

Computational Biology

Rune Bang Lyngsø

Ph.D. Dissertation

Department of Computer Science
University of Aarhus
Denmark

Computational Biology

A Dissertation
Presented to the Faculty of Science
of the University of Aarhus
as Second Mandatory Assignment
in Partial Fulfilment of the Requirements for the
Ph.D. Degree

by
Rune Bang Lyngsø
September 13, 2001

Outline of Dissertation

> Back where I come from, we have universities of great learning, where men go to become great thinkers. And when they come out, they think deep thoughts and with no more brains than you have. But they have one thing that you haven't got: a diploma.
>
> —Wizard of Oz, *Wizard of Oz*

During four years of arduous service, a Ph. D. student is expected to familiarise himself[1] with his field of research, *and*, hopefully, contribute to this field. This is reflected by the division of this dissertation into two parts. Part I is a (partial) overview of the field of computational biology as I conceive it, an overview that is aimed at presenting the context for my contributions to the field of computational biology. These contributions are presented in part II as five independent articles.

Part I is constituted of chapters 1 through 3. Chapter 1 gives a brief introduction to some important concepts from computer science and molecular biology used throughout the dissertation. The remainder of part I is split into two chapters. The first of these, chapter 2, focuses on problems arising when we abstract biomolecules – DNA, RNA and proteins – to be merely sequences over a finite alphabet. In the second, chapter 3, I focus on the structural problems introduced when looking at biomolecules as real, three dimensional chemical molecules.

Part II is constituted of chapters 4 through 8, each consisting of an independent article. Prior to each article there is a short description of the publication status of the results presented in the article, and references to implementations where applicable.

[1] From the Merriam-Webster dictionary [4]: *he [pronoun] – used in a generic sense or when the sex of the person is unspecified.*

Acknowledgements

> Non nobis Domine non nobis,
> Sed Nomini tuo da gloriam
> Sed Nomini tuo da gloriam.
>
> —William Byrd

Writing an acknowledgement section is not an easy task, as too many people to all be listed here have assisted in numerous ways and to various degrees in bringing this dissertation about. So to hedge this list of acknowledgements, let me start by thanking all those people who have helped making life such a breeze during my time at the University of Aarhus and Washington University.

In any list of acknowledgements I could have decided on, except the empty one, one person was bound to be mentioned. So it seems fair to thank Christian Nørgaard Storm Pedersen first of all, for inspiring cooperation and for the numerous, at times heated, discussions about the field of computational biology, research and life in general.

When choosing the area of computational biology as my field of study, I was aware of the problem, that I would not be able to find a advisor at the Computer Science Department who had computational biology as his primary area of research. With this said, I am grateful to my advisor Ole Micheli Caprani for introducing me to the area of computational biology which has turned out to be quite entertaining. For keeping me busy reading about the area in which I was supposed to do work, in those first years of despair, where it seemed to me that all the interesting and significant results had already been picked up. For, if not guiding me in my specific research, then being a great mentor in how to be a scientist. And for his patience.

Furthermore, having Jotun John Hein at the neighbouring Department for Ecology and Genetics has been a great asset. Not only has he been an invaluable source of ideas and suggestions himself, but he has also managed to gather a number of intelligent and inspiring people from various areas of computational biology around him.

Apart from gratitude for dispensing of his vast knowledge of RNA's and their structures, Michael Zuker has earned my great admiration for having the courage to welcome this Great Dane to join his Lab for a year. That he contributed in making my tour of duty an outstanding experience, as well socially as scientifically, also deserves mentioning. One complaint though: Why should it take almost half a year before introducing me to the *social hour* across the street?

Finally I would like to thank Regnar Bang Lyngsø for proofreading this dissertation – though I know it does not relieve me from the full responsibility for the spelling errors, misconceptions and other blunders the reader will encounter on the following pages – and Charlotte Skov for digging up reference [122]. And Scapa Distillery.

Contents

Part I

Overview

Chapter 1

Introduction

> What's in a name? That which we call a rose by any other name would smell as sweet;
>
> —William Shakespeare, *Romeo and Juliet*

The analytical and differential engines of Charles Babbage were mainly intended for generating tables of functions and performing basic mathematical computations. Since then the power of automated computing has increased tremendously. With the advent of affordable computational power numerous fields have discovered the advantages of employing computers' capacities in storage, computation and simulation, thus spawning new interdisciplinary areas with computer science as one of the disciplines.

One such area is *computational biology*. This area is concerned with the use of computers for biological problems, most prominently problems in evolutionary and molecular biology. This area is also referred to as bioinformatics and these two terms are often used interchangeably. However, it does seem that some consensus is forming for using computational biology when the focus is on developing good algorithms for mathematical models with biological relevance, and bioinformatics when the focus is on constructing and using computational tools for biology. With this distinction the work presented in this dissertation clearly falls in the category of computational biology.

1.1 Computational Concepts

When developing an algorithm the primary objective is of course for the algorithm to actually solve the problem it is intended to solve. Another important objective is to limit the resources, usually the time and space, used by the algorithm. A useful way to describe this is by the *O notation* expressing the *asymptotic behaviour* of resource usage. This allows us to ignore unimportant details complicating the analysis but still gives a good foundation for estimating the requirements of the algorithm for new instances of the problem.

Let $f : \mathbf{N} \to \mathbf{R}$ be a function expressing the resource usage of an algorithm, such that $f(n)$ is the maximal resource usage for any problem instance of size n (this measure of size might e.g. be the number of characters if the input is a sequence). Formally $f(n)$ is $\mathrm{O}(g(n))$ where $g : \mathbf{N} \to \mathbf{R}$ (often written

$f(n) = \mathrm{O}(g(n)))$ if $\exists n_0, c > 0\ \forall n \geq n_0 : f(n) \leq cg(n)$, that is, g gives an upper bound on the growth of f. In itself the O notation does not give us any information on the actual resource requirements for a specific instance of the problem. But coupled with a good estimate on c, possibly in part obtained by some representative examples, it does allow rough estimates for problem instances of any size.

Actually it does not make sense to specify the resource requirements for an algorithm without having fixed a *computational model*. If our model only allows us to add numbers and not multiply them it would take time $\mathrm{O}(\log n)$ to compute n^2 whereas we can compute the same result in time $\mathrm{O}(1)$ in a model with unit-cost multiplication. Throughout this dissertation we will not go into details with the assumed computational models; the assumptions can be inferred from the resource bounds arguments and are all of a 'unit-cost standard computer' type, that is, a model resembling a standard computer (conditional branching, various addressing modes, arithmetic operations with 'sufficient' precision etc.) with single operations taking constant time. One argument for this choice is, that in an interdisciplinary field like computational biology the focus is on computationally solving problems more than on determining the intrinsic complexity of problems.

Of special interest are those algorithms that run in *polynomial* time, that is, the time requirement is bounded by some polynomial in the size of the instance. One can, arguably cf. [113, pp. 6–8], claim that algorithms running in polynomial time are efficient, while algorithms requiring more than polynomial time are inefficient. Furthermore, there is a polynomial correspondence between most realistic computational models, that is, we can simulate one model in another model in time bounded by a polynomial of the time required in the model being simulated. As the composition of two polynomials is again a polynomial, polynomial time is rather robust to the choice of computational model.

The problems for which a polynomial time algorithm is known (or more generally exists) can thus be solved efficiently. This class of problems is denoted by **P** (short for polynomial time) and is often referred to as the *tractable* problems. Another important class for studying efficient solutions is the class of problems that can be solved in *nondeterministic polynomial* time denoted by **NP**. Informally **NP** is the class of problems for which we can verify (but not necessarily find) a solution in polynomial time. Some confusion exists about this being the class of intractable problems but as $\mathbf{P} \subseteq \mathbf{NP}$ this is obviously not the case. The confusion, supported by **NP** being misconceived as short for non-polynomial time, is probably due to the fact, that proving a problem **NP** *complete* is taken as a strong indication that the problem is in fact intractable. The **NP** complete problems are in some sense as hard as they come in **NP** – if there is any problem in **NP** that cannot be solved in polynomial time then no **NP** complete problem can be solved in polynomial time. As many bright people have attempted to find polynomial time algorithms for numerous **NP** complete problems but no one has succeeded, **NP** complete is usually considered equivalent to 'probably intractable'.

With all this said, it should be remembered that what constitutes a good and useful algorithm very much depends on the particular case. Much effort and

computational power has been devoted to find suboptimal solutions to problems arising in telecommunications, airline planning etc. as even small improvements yield significant savings; in *Hitch Hiker's Guide to the Galaxy* [5, chapter 25] a race of hyperintelligent pan-dimensional beings builds Deep Thought, a humongous computer that runs for millions of years to find the answer to the great question of life, the universe and everything. On the other hand polynomial time algorithms finding optimal solutions within a model might be of little interest. In section 3.1.2 we present an algorithm with time complexity $O(n^5)$ for predicting a class of RNA secondary structures containing pseudoknots. Being ready to wait a year for the result, we should thus be able to handle quite large sequences. That kind of patience in this situation is however unrealistic as the result will only be a rough estimate of a real-world secondary structure, a structure that might be of limited significance. Whether the resource requirements are acceptable thus to a high degree depends on the perceived quality of the result of the computation.

1.2 Biological Concepts

Having an efficient algorithm solving a problem is of little use if we do not have interesting instances of the problem needed to be solved. A key component in shaping the field of computational biology has thus been the development of efficient techniques to peek into the fundamental building blocks of life. Where Darwin and Mendel were restricted to just observe the phenotypes, i.e. colour, shape etc., we now have the means to explore the genotype, i.e. the hereditary characteristics, of an organism as well. Apart from opening for a new level of understanding of biological processes, this can also be very helpful when comparing species. As convergent evolution is highly unlikely at the level of DNA sequences, this provides us with a rich source of information that might help settling old disputes. Or introducing new ones [122].

But before we can bring computers to bear on all this new information, we need to bring it into the computer. In other words, we need to design models about the biological realities that are comprehensible to computers. The specific model of course depends on the questions we are asking, but a few elements – the sequence nature of key biomolecules and thermodynamics – are of so general use that we will give a brief treatment of them here.

1.2.1 Biological Sequences

In their genetic material all living organisms carry a blueprint of the molecules they need for the complex task of living. This genetic material is (usually) stored in the form of DNA – short for **d**eoxyribo**n**ucleic **a**cid – sequences. The DNA is not actually sequences but rather long, chain-like molecules consisting of *nucleotides* linked together by phosphate ester bonds. A nucleotide consists of phosphoric acid, a pentose sugar (2-deoxyribose for DNA) and an *amine base*, cf. figure 1.1. The amine base is one of guanine, cytosine, adenine and thymine, cf. figure 1.2. A DNA molecule is thus a uniform backbone of 2-deoxyriboses linked together by phosphates with side chains of amine bases. Hence, a DNA

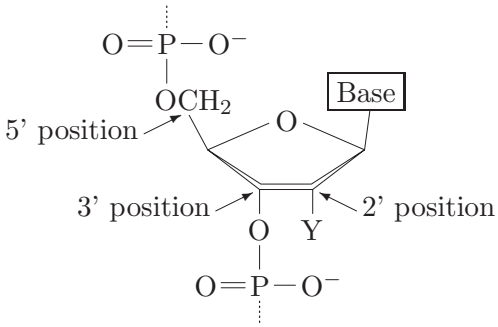


Figure 1.1: A generic nucleotide with the phosphates linking to the previous and next nucleotides. If Y is a hydrogen atom the sugar is 2-deoxyribose, and if Y is is an OH-group it is ribose.

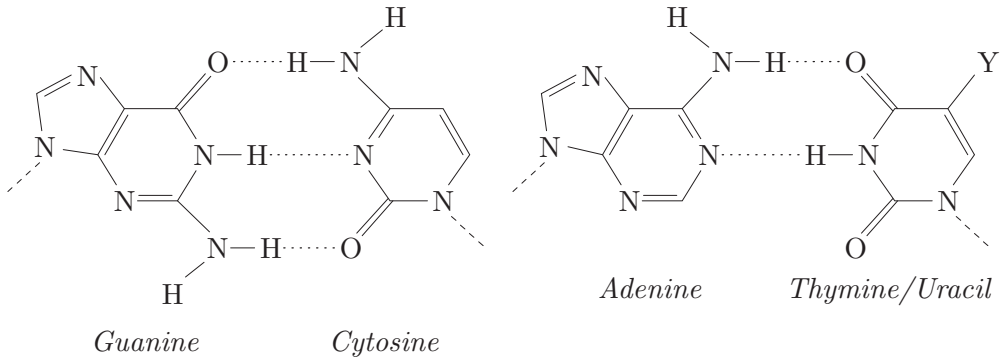
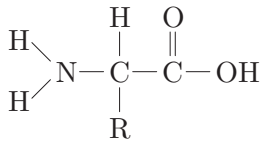


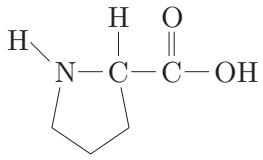
Figure 1.2: The five bases of nucleic acid sequences. When Y is a hydrogen atom, the right-hand base is uracil, and when Y is a CH_3 group, it is thymine. The hydrogen bonds in the base pairings are shown with dotted lines. The connections to the backbone are shown with dashed lines.

molecule can be specified uniquely by listing the sequence of amine base side chains, a listing conventionally starting from the 5' end. This motivates the abstract view of DNA as sequences over a four letter alphabet.

In organisms the DNA is actually not stored as just a single sequence of nucleotides, but as two *complementary sequences* of nucleotides wound around each other in a helix. Two sequences are complementary if one is the other read backwards with guanine and cytosine interchanged and adenine and thymine interchanged. As illustrated in figure 1.2, the four types of amine bases can be split into two pairs of *complementary bases* that can form strong interactions, called base pairings, by hydrogen bonding. Two complementary sequences can thus glue together, forming a helix stabilised by the consecutive stretch of base pairings. This structure was proposed by Watson and Crick [154] and is called the *Watson-Crick model*. The two types of base pairings it involves are called *Watson-Crick base pairs*.



(a) General structure of the nineteen primary amino acids. Primary refers to the amino group having two hydrogen atoms. The type of amino acid is determined by the side chain R.



(b) Structure of proline, the only secondary amino acid encountered in proteins. The three undesignated corners of the pentagon are occupied by CH_2 groups.

Figure 1.3: The amino acids encountered in proteins are all α-amino acids, i.e. the amino group is attached to the carbon atom next to (in α position of) the carboxyl group.

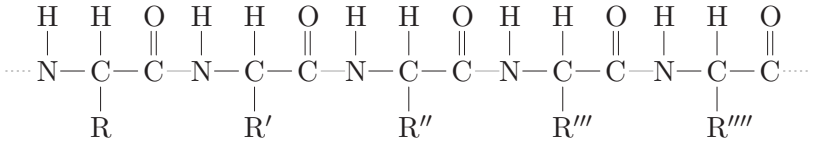


Figure 1.4: A segment of the backbone of a protein. The peptide bonds linking the amino acid residues are indicated by gray lines.

The rôle of DNA is as storage medium for information about the individual molecules needed in the biochemical processes of the organism. A region of the DNA that encodes a single functional molecule is refered to as a *gene*, and when the molecule is needed the gene is transcribed to an RNA – short for **r**ibo**n**ucleic **a**cid – sequence. By the transcription an RNA sequence complementary to the DNA sequence complementary to the gene – and thus (almost) identical to the gene – is constructed. Like DNA, RNA is actually a molecule rather than a sequence, but a molecule very similar to DNA. The only differences are ribose being the pentose sugar, cf. figure 1.1, and uracil replacing thymine, cf. figure 1.2. For RNA it is thus also valid to use a sequence abstraction. Furthermore, we can define an RNA sequence complementary to a DNA sequence in much the same way as for DNA with uracil taking the place of thymine.

Sometimes the RNA molecule generated by the transcription will itself be the functional molecule encoded by the gene, but most often it is intended as a template for a protein. Like DNA and RNA, proteins are chain-like molecules, but for proteins the backbone consists of α-*amino acids*, cf. figure 1.3, linked together by peptide bonds, cf. figure 1.4. Twenty different types of amino acids, cf. [103, table 15.1], are encountered in proteins, so again we observe that a sequence over a finite alphabet suffices to specify a biomolecule.

Before the RNA transcribed from the gene is translated to a protein some parts might need to be removed. In eukaryotes the coding parts of a gene, called *exons*, might be interspersed with non-coding parts, called *introns*. These introns need to be removed, joining the exons at the *splice sites*, to generate the functional *messenger RNA* coding for the protein. When the messenger RNA is translated to the protein it is read three bases at a time. These three bases, called a *codon*, uniquely determines the next amino acid added to the protein being constructed according to the (almost) universal genetic code, cf. [103, table 16.3].

1.2.2 Thermodynamics

When trying to solve a problem by a computer we usually have an objective we are trying to optimise. Nature has its own objective function, *entropy*, that it is perpetually optimising. According to the second law of thermodynamics, the entropy of an isolated system will be non-decreasing, cf. [7, section 3.7]. Biological systems are not isolated but we can derive another quantity, the *Gibbs free energy*, that will be non-increasing under conditions resembling those under which biochemical processes take place, cf. [7, section 4.3]. When modelling biochemical processes, e.g. like structure formation, it is thus often natural to choose an objective function imitating the free energy, as it for one thing lends an immediate interpretation.

The free energy of a system can be divided into an *enthalpic* contribution and an *entropic* contribution, cf. [7, equation 4.14]. The enthalpy of a system can be thought of as the energy stored in the system, e.g. the heat obtained by burning this dissertation equals the change in enthalpy between the dissertation and the ash and other products of the burning process. Entropy is usually refered to as disorder and can be thought of as the number of choices available to the system in its current state. To illustrate this, if we roll two dice and count the total number of spots, we get a number between two and twelve. But all outcomes are not equally probable; we only roll twelve if both dice come out with six but there are six different ways to roll seven. More choices are available for rolling seven and thus the 'entropy' of seven is larger than the 'entropy' of twelve.

To further exemplify this concept, consider the formation of a base pair in an RNA structure, cf. section 3.1.2. The change in enthalpy by forming this base pair will be advantageous, as the enthalpy decreases by the amount of energy needed to break the hydrogen bonds of the base pair. On the other hand, the base pair will fix the two bases pairing in close spatial proximity. This means that there are less conformations available for the RNA molecule. Thus the base pair formation will be entropically disadvantageous. Whether the change in free energy is advantageous or disadvantageous depends on the relative proportions of these two entities.

Chapter 2

Sequence Analysis

> 22:18 For I testify unto every man that heareth the words of the prophecy of this book, If any man shall add unto these things, God shall add unto him the plagues that are written in this book:
>
> 22:19 And if any man shall take away from the words of the book of this prophecy, God shall take away his part out of the book of life, and out of the holy city, and from the things which are written in this book.
>
> —Revelation, *The Bible, King James V version*

Contrary to the 'book of life', the 'articles of life' stored in the genetic sequences of all living organisms do not seem to have any desire for constancy. Over the course of time, these sequences evolve by small discrete changes called mutations such that after a couple of aeons it might be hard if not impossible to find any resemblance to the original sequence. These mutations are not unconstrained, though, as the sequence is the blueprint of some organism that has to live, thrive and survive in the real world. It is thus relevant to examine these sequences, both to look for regularities (that might be used to identify the sequence or from which some of the constraints of evolution can be inferred) and to compare two or more sequences believed to have evolved from the same ancestral sequence. Inferring structural information for the sequence, cf. chapter 3, could be considered part of looking for regularities but the extent and importance of this area usually justifies considering it a realm of its own.

2.1 Detecting Regularities

If we only have one sequence at our disposal we can only ask questions about that one particular sequence, and not questions about how it relates to other sequences. One such question could be detection of introns, exons and splice-sites in the sequence, that is, what parts of the sequence that code for a protein. These methods usually relies on inferring the patterns or regularities sought for from a set of known data, e.g. by training a hidden Markov model (cf. section 2.2.2) as reported by Krogh [79]; then occurrences of these patterns or regularities can be detected to gain information for a new sequence. As the patterns and regularities are inferred from a data set of known sequences, these

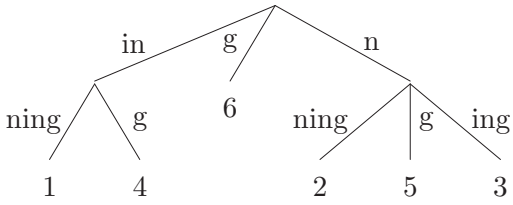


Figure 2.1: The suffix-tree of the word *inning*; the leafs are labeled with the index of the start of the suffix they represent. As g only occurs as the last letter we do not need to add a special end-marker.

methods are on the borderline between finding regularities in a single sequence and comparing sequences. If we truly only have a single sequence at our disposal, we will in some sense have to compare the sequence against itself to find regularities.

Perhaps the simplest possible, non-trivial regularity to look for in a sequence s is repeated occurrences of a substring. If the occurrences of a substring α are contiguous it is called a *tandem repeat* or a *square*; thus s contains a tandem repeat if it contains the substring $\alpha\alpha$ for some α. Recently it has been shown that (a compact representation of) all tandem repeats of a sequence s can be determined in time $\mathrm{O}(n)$ where $n = |s|$, cf. [77, 53]. This implies that we can output all tandem repeats in time $\mathrm{O}(n+z)$ where z is the number of tandem repeats. This improves on previous methods requiring time $\mathrm{O}(n \log n + z)$.

A convenient data structure when looking for repeated patterns in a sequence is the *suffix tree* of the sequence, cf. figure 2.1. A suffix tree $T(s)$ of the sequence s is the compressed trie of all suffixes of s, cf. definition 8 on page 58. A trie of a set of words over some alphabet Σ is a tree where

- each edge is labeled with a character from Σ and the edges going to the children of a node all have different labels.
- the path-label of the path from the root to a node (that is, the concatenation of the labels of the edges traversed on the path from the root to the node) of the trie is a prefix of one of the words in the set.
- for each word there is a node (a leaf unless the word is a prefix of another word) where the word is spelled by the path from the root to that node. If we extend each word with a special end-marker $\$ \notin \Sigma$ each word will be represented by one leaf.

A compressed trie is a trie where non-branching parts of the tree are compressed; all internal nodes with only one child are removed and the paths thus removed are replaced with edges labeled by the strings spelled by the paths, thus preserving the path-labels from the root down to the remaining nodes.

It is easy to see how we can use a suffix-tree to find repeated occurrences of a substring. If $s[i..i+k] = \alpha = s[j..j+k]$ for some $i \neq j$, then both the suffix of s starting at i and the suffix starting at j will begin with α. The paths from the root down to the leafs labeled i and j will thus coincide at least until α has

been spelled. Hence the repeated substrings corresponds to all strings spelled on a path from the root to some internal node, possibly ending somewhere on the middle of the edge, that is, using only a prefix of the edge label of the last edge traversed. In the word *inning* we can thus recognise the three repeated substrings *n*, *i* and *in* , cf. figure 2.1. As the suffix tree of a sequence s can be constructed in time $O(n)$ [155, 102, 144, 41] where $n = |s|$, we can determine all pairs of repeated substrings in a sequence in time $O(n + z)$ where z is the number of pairs of repeated substrings.

In many situations the full set of all repeated sequences might contain redundant and uninteresting information. For one thing, we can observe that the two occurrences of the repeated substring *i* in *inning* are substrings of the two occurrences of the repeated substring *in*. Thus we might want to restrict our attention to *maximal pairs*, that is, a pair of occurrences of a repeated substring where both the characters to the immediate right of the two occurrences and to the immediate left of the two occurrences are different, cf. definition 7 on page 57. Furthermore, as the sequence becomes longer we are bound to find repetitions of longer and longer subsequences simply by the pigeonhole principle. Thus we might not be interested in short repetitions unless the distance between them is in some bounded interval.

In chapter 4 we present a method to find all maximal pairs in a sequence s with an upper and lower bound on the length of the gap between the occurrences, bounds that can depend on the length of the substrings constituting the pair. The method requires time $O(n \log n + z)$ and space $O(n)$, where $n = |s|$ and z is the number of maximal pairs reported; if we only want to impose a lower bound on the length of the gap between the occurrences, the time requirements can be reduced to $O(n + z)$. The problem of finding all maximal pairs with bounded gap length can in some sense be considered an intermediary between finding tandem repeats (where there is no gap between the occurrences) and finding all pairs as described above; in the following we will give a brief sketch of the basic ideas of the method.

Our method is an extension of the suffix-tree based method for finding all pairs described above. We will use the notation (i, j, k) for a pair with the meaning that the two substrings $s[i..i + k - 1]$ and $s[j..j + k - 1]$ are identical and thus forms a pair of substrings of length k, cf. figure 4.1. The first observation is that we can sort out all pairs (i, j, k) where $(i, j, k + 1)$ is also a pair, that is, pairs that can be extended to the right with an extra character, by only reporting pairs of indices from different subtrees of an internal node. Thus a straightforward approach would be to examine all internal nodes; for a particular internal node with α spelled by the path from the root to that node all pairs of indices i, j from different subtrees are examined; if the gap length between $s[i..i + |\alpha| - 1]$ and $s[j..j + |\alpha| - 1]$ is within the bounds and $s[i - 1] \neq s[j - 1]$ the pair $(i, j, |\alpha|)$ is reported. This approach might be to costly, however, as we might inspect numerous pairs of indices that are not reported.

To improve on the efficiency the first thing we need is an improved way of handling each of the indices in one subtree of an internal node in $T(s)$. For a particular index, the range of indices in another subtree of the internal node that it can be reported against should be determined efficiently. This is achieved

by maintaining the indices in balanced search trees. By lemma 5 on page 60, given the indices of one subtree we can find the indices in another subtree from which to start reporting in time $O(\log \binom{n+m}{n})$, where n and m are the number of indices in the two subtrees, if we have the indices stored in balanced search trees. We can now proceed from these starting points, scanning through indices until we reach the other end of the bounded interval and reporting all pairs provided they do not have identical characters to the immediate left. This provision constitutes the last obstacle as we again might find ourselves inspecting too many pairs that cannot be reported. This is handled by maintaining an extra balanced search tree allowing us to skip blocks of consecutive indices with the same character to the immediate left. The full data structure constructed at each node in the suffix tree is illustrated in figure 4.3. By lemma 4 on page 60, if we traverse $T(s)$ in a bottom-up fashion we can maintain this data structure at the nodes visited in time proportional to the time we use to search the data structure. By applying the "smaller-half trick", that is, always searching with the smaller of two sets of indices, we can limit the time required, disregarding time that can be attributed to reporting of pairs, by $O(n \log n)$. The details can be found in section 4.3.

If only a lower bound is imposed on the gap length between occurrences we get off easier on two accounts. First, for an index from one subtree we do not need to search for the indices in another subtree from which to report; we can just start from the index farthest away and continue until we get to an index that is too close. This trick has to be applied twice, though, as it is too costly to look through all indices in one subtree if most of them cannot be reported against any indices in the other subtree. We thus start from opposite ends and work our way inwards. Secondly, as we only need to find extreme indices – the ones farthest away – we do not need to maintain a fully ordered tree structure; a heap-ordered tree structure will suffice. This allows us to merge the indices of two subtrees in amortised constant time. Thus, disregarding time that can be attributed to reporting of pairs, we only need time proportional to the number of nodes of $T(s)$, which is $O(n)$, to report all maximal pairs with a lower bound on gap length for a sequence s. The details can be found in section 4.4.

So why look for these repetitions? Occurrences of tandem repeats – or rather tandem arrays, that is, multiple contiguous occurrences of a substring – are well-known in e.g. the human genome where tandem repeats are found in several interesting connections, cf. [52, pp. 139–142]. A widespread use of tandem arrays is as 'genetic fingerprints' as the number of repetitions shows a high variability between individuals, cf. [24]. In most, if not all, of these applications it is however known where to look for the repeated sequences, so the problem of finding repetitions is probably most correctly termed a problem inspired by biology rather than a biological problem.

2.2 Comparing Sequences

As soon as we have two or more sequences we can compare them against each other. The uses of comparing sequences are many. We might compare a new se-

$$
\begin{array}{c}
-\,t\;h-e---\\
-\,t\;h\;r\;e-e-\\
s\;\;t-r\;\;i\;\;n\;g\;\;s
\end{array}
$$

Figure 2.2: An example alignment of the three strings *the*, *three* and *strings*.

quence to a database of known sequences to find homologous sequences, that is, sequences which have evolved from a common ancestral sequence. Homologous sequences can be compared simply to gain information about how distantly related they are or to try to infer evolutionary constraints, e.g. like conserved residues in a protein that can indicate an active site[1], or compensatory changes that can indicate a base pairing in an RNA molecule, cf. section 3.1.2.

The evolution of DNA sequences usually takes place by small, local changes, where one nucleic acid is substituted for another or a short sequence of nucleic acids are either inserted or deleted in the sequence. These are not the only kinds of mutations known but the most frequently observed; a large part of the work concerned with comparing biological sequences has been devoted to models that only involves substitutions and indels (*indels* is a contraction of insertions and deletions – in some situations there is made no distinction between these, and indel is used as common nominator). Except where specifically stated otherwise, in the following we will assume an evolutionary model that only allows these two types of mutations.

Traditionally the *alignment* notation, cf. figure 2.2, has been used to illustrate a comparison between two or more sequences. An alignment of m sequences is an $m \times n$ matrix where an entry in the i'th row either contains a character from the i'th sequence or a special *gap character*; the character '$-$' is commonly used as gap character. Furthermore the concatenation of the non-gap entries of the i'th row is identical to the i'th sequence (thus $n \geq n_i$ for $1 \leq i \leq m$ where n_i is the length of the i'th sequence). The alignment notation makes it easy by eye to see the regions of highest similarity and to get an impression of which parts of the sequences that correspond to each other. Apart from being an informative representation of a comparison, alignments can also be considered a useful tool when manually comparing sequences. The notation is helpful in spotting possible improvements.

We mentioned above that an alignment makes it easy to see which regions of the sequences that corresponds to each other, but the information in an alignment is even more fine-grained than this. A column in an alignment will contain at most one character of each sequence; these characters can be said to occupy the same *position* in the alignment or in the family of sequences that are aligned. Assume that the sequences we have aligned all descend from the same ancestral sequence. In a perfect evolutionary alignment[2] of these sequences the

[1]Proteins usually works as catalysts – they form a close interaction with one or more molecules by which the speed of some biochemical reaction is increased; the place on the protein molecule where the actual catalysis takes place is called the active site.

[2]In some cases it might be desirable to have an alignment express information other than evolutionary, e.g. the positions corresponding to amino acids occupying roughly the same spatial positions in the structures of a set of aligned protein sequences.

characters in a position should be characters that by a process of zero or more substitutions can be traced back to the same character in the ancestral sequence or to a character that at some time was inserted into one of the lineages. A gap thus indicates either that the character in that position was deleted from the sequence at some point or that the sequence is not part of the lineage into which the character at that position was inserted.

As alignments are useful and well established in the context of comparing biological sequences the problem of comparing biological sequences is often referred to as the alignment problem, even when other information than an actual alignment is sought for. Traditionally alignments were constructed by an expert – and still are when an alignment of utmost quality is needed – who aligned regions and characters of the sequences he deemed to be corresponding. To make a computer 'deem' anything we need a model and an objective. In the following we will discuss two widely used types of models and related parts of our work.

2.2.1 Distance Models

Assume that we have two sequences a and b believed to have evolved from an unknown common ancestor c. As we do not know c, for all we know any sequence could be the ancestor of a and b. Furthermore, even if we did know c, we would not know which characters in a and b that should be in the same position as we do not have information on how a and b evolved from c. The arbiter for handling this situation is Occam's Razor, or, as it is better known as in the area of evolution, the parsimony principle. It basically states, that we ought to assume that nature is cheap and thus the most probable of all possible explanations is the simplest.

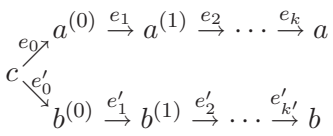
Let us first assume that we have the question of simplicity of evolving one sequence from another decided by some (real-valued) function $evol(x, y)$, that tells us how costly it is for evolution to get from sequence x to sequence y. Applying the parsimony principle we can then define the *evolutionary distance* between our two sequences a and b as

$$dist(a, b) = \min_c \{evol(c, a) + evol(c, b)\}, \tag{2.1}$$

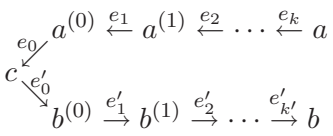
that is, the minimum over all possible choices of an ancestral sequence of the sum[3] of the costs of evolving the sequence into a and b respectively. The sequences minimising the above expression are the most parsimonious estimates for an ancestral sequence.

As mentioned above we assume that evolution can be explained by a series of substitutions, insertions and deletions, or more generally as a series of discrete events changing one sequence into another. Assume we are given a function $cost(x \xrightarrow{e} y)$ assigning costs to an event e that transforms x into y, and a

[3]It should be noted that we could have chosen other, equally suitable, ways to combine the evolutionary costs than a sum, e.g. taking the maximum of the two costs. The model we are specifying here is thus not founded solely on parsimony but also takes other considerations into account, prominent among which is convenience. Other models would fit equally well in a section on distance models.



(a) The standard situation where we evolve a and b independently from the common ancestor c.



(b) With reversible events we can reverse the direction of evolving a from c, thus changing the view to b evolving from a over the intermediary sequence c.

Figure 2.3: Evolution of the two sequences a and b from a common ancestor c by independent sequences of evolutionary events E and E'.

sequence E of events $e_1, e_2, \ldots, e_k$ transforming one sequence $x^{(0)}$ into another sequence $x^{(k)}$ by $x^{(0)} \xrightarrow{e_1} x^{(1)} \xrightarrow{e_2} \cdots \xrightarrow{e_k} x^{(k)}$. We can now define the cost of evolving $x^{(0)}$ into $x^{(k)}$ by the sequence of events E as the sum of the costs of each of the events,

$$cost(x^{(0)} \xrightarrow{E} x^{(k)}) = \sum_{e_i \in E} cost(x^{(i-1)} \xrightarrow{e_i} x^{(i)}), \tag{2.2}$$

cf. equation 5.2 on page 85. Once again using the parsimony principle we can define the cost of evolving sequence x into sequence y as

$$evol(x, y) = \min\{cost(x \xrightarrow{E} y) \mid E \text{ is a sequence of events}\}, \tag{2.3}$$

cf. equation 5.3 on page 85.

Combining equations 2.1 and 2.3 we get a new expression for the evolutionary distance between a and b,

$$\begin{aligned} dist(a, b) = \min\{&cost(c \xrightarrow{E} a) + cost(c \xrightarrow{E'} b) \mid \\ &c \text{ is a sequence and } E \text{ and } E' \text{ sequences of events}\}. \end{aligned} \tag{2.4}$$

This expression is illustrated in figure 2.3(a) where the two sequences a and b independently evolve from an ancestral sequence c. If we assume evolutionary events to be reversible, that is, $cost(x \xrightarrow{e} y) = cost(y \xrightarrow{e'} x)$, the situation simplifies. Instead of having a and b both evolving from the ancestral sequence c, we can take the alternative view that a evolves first into c and then from c to b as illustrated in figure 2.3(b). Using the assumptions of additive costs this allows us to simplify equation 2.4 to

$$\begin{aligned} dist(a, b) &= \min\{cost(a \xrightarrow{E} c) + cost(c \xrightarrow{E'} b) \mid \\ &\qquad c \text{ is a sequence and } E \text{ and } E' \text{ sequences of events}\} \\ &= \min\{cost(a \xrightarrow{E} b) \mid E \text{ is a sequence of events}\}, \end{aligned} \tag{2.5}$$

eliminating the minimum over ancestral sequences from the equation. Hence we only need to search over all possible sequences of evolutionary events changing

a into b. From a sequence of events we can easily construct an alignment simply by keeping track of which characters are substituted, inserted or deleted; the distance between sequences thus gives us a model and an objective allowing a computer to 'deem' what alignment is the best.

Even with the above simplifications the problem of finding the most parsimonious evolutionary path remains in the realm of undecidable problems as we have not restricted the dependencies of the cost function of an event. It can depend arbitrarily on the context in which it takes place. If we have the set of events changing one sequence into the other – a set obtained e.g. from an alignment by postulating the substitutions, insertions and deletions expressed by each column – finding the optimal order of this set of events is still **NP** complete if we allow arbitrary context dependence in the cost function of single events.

If we require the cost function to be context-free, that is, only depending on the actual part of the sequence that changes, one of the simplest imaginable cost functions is the function where each substitution and each character inserted or deleted has unit cost. The distance thus defined is usually called the edit distance or Levenshtein distance and measures the number of single-character changes – substitutions, deletions or insertions – needed to change one sequence into the other.

To find the edit distance between two sequences a and b we can specify a recursion in terms of edit distances between prefixes of a and b,

$$\begin{aligned} dist(a[1..i], b[1..j]) = \min\{ & dist(a[1..i-1], b[1..j]) + 1, \\ & dist(a[1..i], b[1..j-1]) + 1, \\ & dist(a[1..i-1], b[1..j-1]) + \delta(a[i], b[j])\} \end{aligned} \tag{2.6}$$

where

$$\delta(\sigma_1, \sigma_2) = \begin{cases} 0 & \text{if } \sigma_1 = \sigma_2. \\ 1 & \text{otherwise.} \end{cases}$$

For reasons of succinctness we omit the special boundary cases that are trivial modifications of the above recursion. The intuition behind the recursion is that either $a[i]$ has been deleted, $b[j]$ has been inserted or $a[i]$ and $b[j]$ are in the same position. The simplicity of the recursion is a consequence of the context-free nature of our cost function – the cost of e.g. substituting $a[i]$ with $b[j]$ does not depend on whether $a[i-1]$ has been deleted, $b[j-1]$ has been inserted or $a[i-1]$ has been substituted with $b[j-1]$.

Using the above recursion it is easy to devise an algorithm using dynamic programming to compute an alignment corresponding to the minimum edit distance in time and space $O(|a||b|)$. Kruskal [81, pp. 23–24] lists a number of independent discoveries of this algorithm. If we are only interested in the edit distance we can do with $O(|a|)$ space by computing the distances in order of increasing length of the prefix of b; Hirschberg [63] shows how linear space can be obtained even when computing a corresponding alignment by recursively finding the alignment of the mid character of a subsequence of one of the sequences.

Unit costs might not be the best choice when comparing biological sequences as some events are more likely than others. One can observe, however, that the

recursion of equation 2.6 applies for arbitrary choices of costs for substitutions and single character indels as long as δ is a metric. Gotoh [46] describes how to compute the distance between two sequences when the cost of an indel of k characters is an affine function of k. This method does not lead to an increase in time and space complexities compared to the simple algorithm sketched above, and works by introducing two new arrays for storing distances between prefixes when an insertion or deletion has already been initiated. More complex cost functions for indels can be handled while increasing the time complexity by at most a factor $O(\log(|a| + |b|))$, cf. [39, 106, 83].

When comparing two DNA sequences each coding for a protein we are faced with a difficult choice: Should we just compare the DNA sequences or should we compare the two proteins they code for? The evolutionary events our model postulates takes place in the DNA sequence but the evolutionary constraints mainly depends on the protein expressed from the DNA. In the protein we can only make insertions and deletions between codons; as many amino acids are coded for by several different codons there is no way of telling how alike the underlying codons of two amino acids are if we restrict our attention to proteins. On the other hand, proteins evolve slower than their coding DNA; thus the protein might be more informative, especially when comparing distantly related sequences. It would be desirable to consider the DNA and the protein coded for simultaneously[4].

Hein [60] proposes a model for assigning costs to evolutionary events in a DNA sequence where changes at both the DNA and the protein level are considered. The evolutionary events affects the DNA level directly and are thus scored by a cost function (of the type presented above where the cost of a substitution only depends on the nucleotides involved and the cost of an indel only depends affinely on its length). On the protein level evolutionary events only have an indirect affect through the mapping from DNA to protein; a deletion on the DNA level might not be explainable by a single substitution or indel, cf. figure 5.3(c) on page 87. On the protein level an evolutionary event is thus scored by the distance between the protein encoded by the DNA before and after the event. Given two DNA sequences a and b coding for protein sequences A and B the cost of an event e changing a into b is thus

$$cost(a \xrightarrow{e} b) = cost_d(a \xrightarrow{e} b) + dist_p(A, B), \tag{2.7}$$

where $cost_d$ is the event cost function on the DNA level and $dist_p$ is the evolutionary distance between protein sequences (we will in the following assume that the distance between proteins is defined by a model of the type described above, i.e. a model only allowing substitution of single amino acids and indels of consecutive amino acids), cf. section 5.2. Furthermore, indels are restricted to be of a length divisible by 3, which reduces the context sensitivity for indels at the protein level. This restriction can be justified by the rareness of indels

[4]This is evidently not the only two levels of information that we might find it opportune to combine, e.g. for DNA coding for RNA molecules structure predictions can be incorporated in the comparison, cf. [131, 45]. A major asset of this particular combination, however, is that it requires no further input than the DNA sequence, and does not rely on uncertain prediction methods.

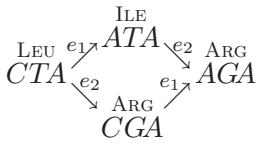


Figure 2.4: An example of the non-commutativity of the model proposed in [60]. One event e_1 changes the first nucleotide of a DNA sequence from C to A and the other event e_2 changes the second nucleotide from T to G. The sum of the costs at the protein level depends on the order in which e_1 and e_2 takes place.

causing frame shifts, a rareness that is due to the cataclysmic effects of such events, cf. figure 5.1 on page 86.

This model significantly increases the complexity of finding the distance between two models. The simple recursion of equation 2.6 no longer holds as the protein level introduces context sensitivity to the cost function. Furthermore we do not even retain commutativity between evolutionary events as illustrated in figure 2.4 (an example inspired by a similar example in [60]); it matters in which order the events takes place and we thus cannot assign a cost to one event without knowledge of other events.

We can however retain the context-free property to some degree. Hein [60] observes that we can split an alignment of two sequences a and b into indels occurring between codons and a number of *codon alignments*, cf. figure 5.5 on page 90. A codon alignment is a minimal part of the alignment flanked by positions occupied by nucleotides from the first position[5] in a codon from both sequences to the left and from the third position in a codon to the right. In between these two positions there will be one position occupied by nucleotides from the second position in a codon from both sequences and the rest of the positions will represent indels. One can observe that the events represented by a codon alignment does not affect the protein sequence except for the part encoded by nucleotides occupying positions within that codon alignment. Coupled with some fair assumptions about the relative costs of substitutions and indels discussed in section 5.2.3 this ensures that the cost of a codon alignment only depends on the events of the codon alignment.

Hein [60] uses this to formulate a recursion similar to equation 2.6 – the distance between $a[1..3i]$ and $b[1..3j]$ is the minimum over all appropriate choices of i' and j' of the sum of the distance between $a[1..3i']$ and $b[1..3j']$ and an optimal codon alignment of $a[3i'+1..3i]$ and $b[3j'+1..j]$. By some further assumptions limiting the number of different types of codon alignment, cf. section 5.3, this is used to formulate an algorithm requiring time $O(|a|^2|b|^2)$ for determining the distance between two sequences in the combined DNA and protein cost model.

In chapter 5 we present an improved algorithm for this problem. The basic ideas of the algorithm are a vast extension of the ideas of [46] having numerous arrays for storing extendable distances between prefixes of the sequences, and

[5]The position in a codon should not be confused with the position in an alignment; a codon consists of three nucleotides that are said to be in the first, second and third position of the codon.

the introduction of the concept of *witnesses*. A deletion in a codon alignment might on the protein level not just remove a stretch of consecutive amino acids but instead compress them to a single amino acid, cf. figure 5.3(c) on page 87. We thus need to keep track of possible witnesses in the compressed stretch of amino acids that can be matched with the remaining amino acid. The time complexity of the algorithm is $O(|a||b|)$ with a constant depending on how many arrays 'numerous arrays' are; by being very meticulous, this number can be kept at roughly 400 thus yielding an acceptable running time for most applications. With 400 arrays space usage could be a cause for worrying but the algorithm allows using the trick of [63] to obtain linear space complexity.

At a first glance the increase in complexity of our algorithm compared with the simple algorithm based on equation 2.6 might seem stunning. One way to account for this increase in complexity is by looking at the difficulties extracting a minimum cost of the events expressed in an alignment. With Levenshtein distance we can simply sum over the cost of the event postulated by each column. Using an affine cost for indels increases the complexity slightly as we have to group consecutive columns postulating an insertion or a deletion; Gotoh's algorithm [46] reflects this by increasing the number of arrays from one to three. In the model of Hein [60] we can no longer look at events separately but have to group the alignment into codon alignments; for each codon alignment we then have to find the minimum cost over all permutations of the events postulated in the codon alignment.

So far we have gracefully avoided discussing the problem of comparing more than two sequences. Adding more sequences adds new problems. First of all, how should we define distances? A comparison of all pairs of sequences might lead to sets of incompatible events, e.g. the comparison between a and b claiming $a[i]$ and $b[j]$ occupies the same position, the comparison between b and c claiming that $b[j]$ and $c[k]$ occupies the same position and the comparison between c and a claiming that $c[k]$ and $a[i']$, with $i \neq i'$, occupies the same position. But any multiple alignment of all the sequences, cf. figure 2.2, induces an alignment of all pairs of the sequences (removing the rows not corresponding to either of the sequences and, possibly, removing columns in the remaining alignment containing only gap characters), a set of pairwise alignments that are without such incompatibilities. The score defined by taking the sum over the costs of all these alignments is called the sum-of-pairs score; the problem of finding the multiple alignment with least sum-of-pairs score is **NP**–hard, cf. [152]. Other scoring schemes might seem more relevant, e.g. scoring schemes taking the tree-like shape of evolutionary relationships, cf. figure 2.5, into account; this does not help any on the intractability of the problem as finding the optimal tree alignment is **MAX SNP**–hard, cf. [152].

The problem of finding a good multiple alignment is of so grave importance that the results on tractability stated above has not deterred people from taking stabs at it. Bafna *et al.* [12] present an approximation algorithm finding a multiple alignment of k sequences with a sum-of-pairs score that is at most $2 - \frac{l}{k}$ from the optimal score in polynomial time for fixed l (and $k \geq l$). Carrillo and Lipman [25] and Gupta *et al.* [51] suggest bounding the search space for finding the optimal multiple alignment. A multitude of heuristics have been

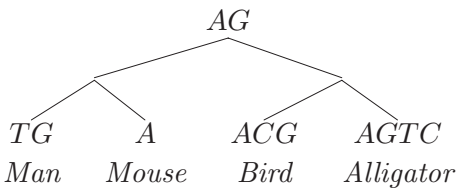


Figure 2.5: An example evolution of a hypothetical ancestral sequence into four different homologous sequences.

proposed; e.g. Hein [59] proposes a heuristic simultaneously aligning the sequences and building a phylogeny[6] for the sequences; Bucka-Lassen *et al.* [23] propose a method for combining many multiple alignments into an improved alignment. One of the most successful and popular heuristics, introduced by Krogh *et al.* [80], is using profile hidden Markov models to generate an alignment.

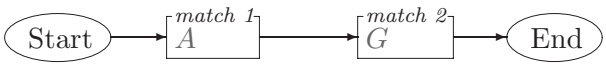
2.2.2 Hidden Markov Models

A Markov chain is a sequence of symbols or states $q_{i_0}q_{i_1}\ldots q_{i_j}\ldots$ where the probability of observing a specific state in the j'th position only depends on the state observed in the $j-1$'st position, that is, the conditional probability $P(q_{i_j} = p \mid q_{i_0}\ldots q_{i_{j-1}}) = P(q_{i_j} = p \mid q_{i_{j-1}})$. We can think of a Markov chain as generated by a Markov model consisting of a set of states S and a transition function $P_q : S \to \mathbf{R}$ for each state $q \in S$. The transition function $P_q(p) = P(q_{i_j} = p \mid q_{i_{j-1}} = q)$ gives the probability of moving to state p when the current state is q. Starting in a specific start state or in a state chosen according to some probability distribution over the states we can now generate a Markov chain by outputting the current state q and choosing the next state according to P_q. A special end-state can be added to explicitly stop the sequence generation.

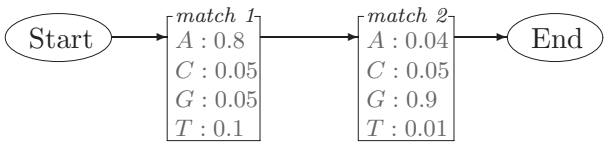
In a hidden Markov model instead of outputting the states entered, at each state q a character from some alphabet Σ is output according to a probability distribution $P_q : \Sigma \to \mathbf{R}$. Thus we do not observe the state sequence but only the sequence of characters outputted at the states, hence the term hidden. In some situations states not outputting any character are useful; such states are called *silent* states in contrast to the *non-silent* states outputting characters. One use of hidden Markov models is for annotating a sequence s; given a path through a hidden Markov model that generates s we can annotate the characters of s by the states outputting them. In the following we will make this concept clear by developing the architecture known as profile hidden Markov models, cf. [80].

Assume we have some sequences that all descend from a common ancestral sequence AG of two characters, cf. figure 2.5. If we start out by only modelling positions corresponding to characters present in the ancestral sequence, we can

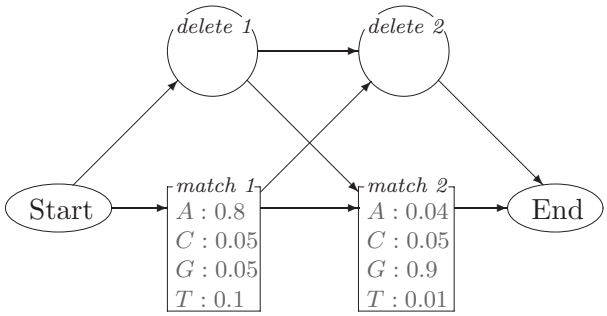
[6] A phylogeny for a set of sequences is a tree showing evolutionary relationships between the sequences, cf. figure 2.5.



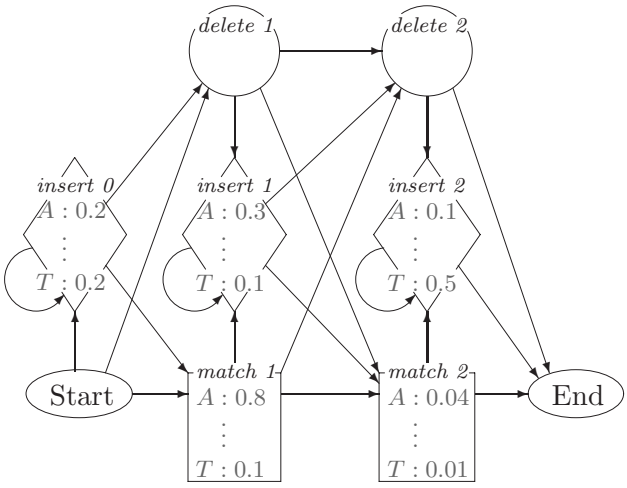
(a) A hidden Markov model for the sequence AG.



(b) A hidden Markov model for a sequence of two characters allowing substitutions.



(c) A hidden Markov model for an ancestral sequence of two characters allowing substitutions and deletions.



(d) A profile hidden Markov model with two ancestral positions.

Figure 2.6: Incremental construction of the profile hidden Markov model architecture. For clarity the transition probabilites have been omitted and only transitions with nonzero probabilities are shown.

model the ancestral sequence by a hidden Markov model that from the start-state moves to a state that outputs an A, then to a state that outputs a G and finally to the end-state, cf. figure 2.6(a). During evolution the characters do not stay fixed but can be changed by substitutions, e.g. the A of the ancestral sequence has changed to a T in the sequence labelled *man* in figure 2.5. Thus instead of outputting an A with probability 1 in the first position and a G with probability 1 in the second position we assign a probability distribution over the characters of our alphabet to each state, cf. figure 2.6(b). These two states are called *match-states* as a character output from one of these states is said to match the corresponding character in the ancestral sequence.

Another possible event affecting a position is the deletion of the character in that position, e.g. the G in the ancestral sequence has been deleted in the sequence labelled *mouse* in figure 2.5. This could be modelled by assigning a probability to not outputting any character in the output distribution of the match-states. But the chance that the character in the next position has been

deleted depends on whether the character in the current position has been deleted and thus a silent *delete-state* is added to the model for each position, cf. figure 2.6(c).

The last event is the insertion of new characters, e.g. a C is inserted between A and G in the sequence labelled *bird* and a TG is inserted at the end of the sequence labelled *alligator* in figure 2.5. Insertions do not affect positions corresponding to characters in the ancestral sequence but rather takes place either between two such positions or at the beginning or the end of the sequence. Thus we add *insert-states* at these places, cf. figure 2.6(d). Furthermore, as more than one character can be inserted between two positions corresponding to characters in the ancestral sequence, an insert-state has a self-loop allowing a transition to itself.

Given a sequence we can thus use a path through a profile hidden Markov model to annotate it with positions. There will usually be a host of different paths through a hidden Markov model generating a specific sequence but each path has an associated probability. It is thus (usually) a natural choice to annotate a sequence by (one of) the most probable path generating it. The probability of the most probable path can be found by recursions similar to equation 2.6 by for each state q and prefix $s[1..i]$ of s to find the probability $P(q, i)$ of the most probable path generating $s[1..i]$ and ending in q; the basic form of these recursions are

$$P(q, i) = P_q(a[i]) \cdot \max_{p \to q}\{P_p(q) \cdot P(p, i-1)\} \tag{2.8}$$

for non-silent states and

$$P(q, i) = \max_{p \to q}\{P(p, i)\} \tag{2.9}$$

for silent states where the meaning of the $p \to q$ notation is that the maximum is taken over all states p with a transition to q. This gives us a method for generating a multiple alignment for a set of sequences using a (profile) hidden Markov model; the quality of the alignment of course depends on how well the model models the family of sequences we want to align.

For the typical situation where we do not have the evolutionary history of the sequences we want to align ready at hand for determining the parameters of our model, there are methods for improving the quality of the model by 'training' (iteratively adjusting the parameters) the model with the sequences. One commonly used method is to adjust the parameters based on the relative probabilities of using a transition or outputting a character in a state, summed over all paths generating any of our sequences. To this end, for any prefix $s[1..i]$ of s and state q we need to be able to compute the sum, instead of the maximum, of the probabilities of all paths generating $s[1..i]$ and ending in the state q. This can be done by a trivial modification of equation 2.8 and equation 2.9, changing maximum to sum. Being able to find the total probability that a hidden Markov model generates a given sequence by any path also comes in handy when using hidden Markov models as classifiers, i.e. to answer whether a new sequence belongs to the family modelled by the hidden Markov model.

But what if we instead of just having a single sequence have a family of sequences we want to compare to the family of sequences represented by a hidden Markov model? We could of course make all pairwise comparisons between the sequences of the two families or between the the sequences of one of the families and the hidden Markov model representing the other family of sequences. If we have hidden Markov models representing both families M_1 and M_2 it will often be desirable to be able to just compare these as

- the hidden Markov models is a general representation of the entire family and not just a single observation from the family.
- it is less confusing having only a single comparison result.
- it will be faster if we can efficiently compare the models as we need to make only one comparison.

In chapter 6 we present an algorithm to compute the *co-emission probability* of two profile hidden Markov models, that is, the probability that the two models independently generate identical sequences. The algorithm is a generalisation of the algorithm to find the total probability that a model generates a specific sequence. For any pair of states $q \in M_1$ and $q' \in M_2$ we compute the sum over all pair of paths π and π' generating identical sequences and ending in q and q' respectively of their joint probability. This sum is found recursively by looking at pairs of states with transitions to q and q'. The only two problems introduced by this generalisation is

- we have to be careful not counting any pair of paths more than once.
- the insert-states have transitions to themselves and thus we have a cyclic problem, needing the co-emission probability at q, q' for computing the co-emission probability at q, q' when q and q' are both insert-states.

The latter problem is solved by recognising the geometric series in the derived expression for the co-emission probability at a pair of insert-states. The time complexity of the algorithm is $\mathrm{O}(m_1 m_2)$ where m_1 is the number of transitions in M_1 and m_2 is the number of transitions in M_2.

The algorithm immediately generalises to all left-right models, i.e. models where the underlying graph of nonzero transitions is a directed acyclic graph when self-loops are ignored. In section 6.5.1 we observe that we at a pair of states q, q' only need to be able to compute the probability of returning to q, q' by paths generating identical sequences to apply the geometric series trick. This widens the class of models that can be handled to all models where a state is part of at most one cycle.

In section 6.5.2 we remark that the co-emission probability for general hidden Markov models can be determined by solving a set of linear equations with a variable for each pair of states. Furthermore, we present an algorithm to approximate the co-emission probability for general hidden Markov model. The algorithm incrementally finds the probabilities of pairs of longer and longer paths having emitted identical sequences. In k rounds it finds upper and lower

bounds on the co-emission probability differing by at most c^k where $c < 1$ is a constant that depends on the models compared. The time complexity of each round is $O(m_1 m_2)$ where m_1 is the number of transitions in M_1 and m_2 is the number of transitions in M_2. This algorithm can thus be expected to obtain a decent estimate for the co-emission probability significantly faster than the linear equation method in many cases.

We now have a method for computing the co-emission probability of two models, but as stated in proposition 2 on page 112 the co-emission probability does have some deficiencies as a measure on the similarity between two models. In section 6.4 we observe that a hidden Markov model – or rather the probability distribution over all finite sequences represented by the model – can be viewed as a vector in the infinite-dimensional space spanned by all finite sequences over our alphabet. With this view the co-emission probability becomes the inner product of two models and based on this we can define angles and distances between models. We also present two similarity measures based on the co-emission probability possessing some of the desirable properties the co-emission probability is lacking.

As repeatedly hinted at above, profile hidden Markov models are not by far the only type of hidden Markov models used in computational biology. Hidden Markov models have been used for gene prediction [79], recognition of transmembrane proteins [134], prediction of signal peptides and signal anchors [111] and prediction of protein secondary structure [11] just to mention a few applications. Our method can thus also be used to compare families of sequences against these more structural or pattern oriented models, or even to add information from such models in training a new model. Another use that we are currently in the process of investigating is to what extent we are able to reconstruct a model based on data. Given a model we can generate sequences according to the probability parameters of the model and examine e.g.

- how does the distance between a trained model and the original model depend on the size of the set of training sequences?
- what are the effects of taking a wrong guess at the underlying architecture, e.g. how close can we come to the probability distribution of the original model when varying the length of a profile hidden Markov model relative to the length of the original model?
- how much do Dirichlet mixtures [21] help?
- are there differences between different packages for constructing hidden Markov models, e.g. HMMER [1], SAM [3] and HMMpro [2]?

Two major sources of uncertainties when working with hidden Markov models is how successful we are at reconstructing a model based on a set of data and to what extent the part of reality we are trying to model can be modelled by a hidden Markov model. With our method for comparing hidden Markov models we can answer the first in a meaningful way.

Chapter 3

Structure Prediction

> Nuts.
>
> —Anthony Clement McAuliffe, *Bastogne, December 22nd, 1944*

The purpose of all the genetic material stored in all living organisms is not to give biologists useful information for comparing the organisms. Rather the genetic material is the blueprint for the biochemical molecules the organisms need in the processes that make them live. The genetic material is stored in DNA and translated to RNA or proteins, all types of molecules that allows us to view it as sequences, cf. section 1.2.1. There is strong evidence that most of the 'characters', or residues, in these sequences are not by themselves important for the chemical reactions the molecules participates in; the functionality of a protein or an RNA molecule is primarily determined by the three-dimensional structure and a few key residues. On the other hand, experiments by Anfinsen *et al.* [8] supports that the total sequence of characters determines the structure, thus allowing us to refer to the structure of an RNA or protein sequence.

As the structure of a biomolecule is essential for its function, determining this structure is of interest on at least two accounts. First of all it might help us determining the function of the molecule, thus identifying the roles of different molecules and helping in understanding biological processes; it might also allow designing new molecules with desirable functions. Secondly, the structure will be more conserved than the underlying sequences – if the function of the molecule is sufficiently important, a mutation causing a major distortion of the structure will almost always leave the organism inviable or severely impaired; a mutation not causing a significant structural change will, unless it affects one of the few key residues have at most a negligible effect. Thus we might use structural information to improve alignments of sequences, e.g. by simply aligning the structures [139].

Experimentally determining the structure of a protein or RNA molecule is at present a complicated and laborious task easily requiring several years of work. Being able to computationally determine the structure of a protein or RNA molecule based on the sequence of amino acids or nucleic acids, and possibly other easily obtainable information, is thus a quest-worthy project, qualifying as one of the numerous holy grails in computational biology. Traditionally structural information is categorised in four levels, cf. [103], for proteins:

> At its simplest, protein structure is the sequence in which amino acid residues are bound together. Called the **primary structure** of a protein [...] Thus, the term **secondary structure** refers to the way in which *segments* of the peptide backbone are oriented into a regular pattern; **tertiary structure** refers to the way in which the *entire* protein molecule is coiled into an overall three-dimensional shape; and **quaternary structure** refers to the way in which several protein molecules come together to yield large aggregate structures.

Only for secondary structures does the hierarchy for RNA molecules differ although the secondary structure of RNA also deals with structurally local information.

As biomolecules are part of reality they (are believed to) obey the laws of thermodynamics, cf. section 1.2.2, and thus the most stable structure should be that of lowest free energy. At the time scale of biochemical processes other aspects, e.g. kinetics, might also play an important role in the structure formation but still it is common to frame models for structure predictions in terms of a free energy that we want to minimise. Whenever we in the following refer to energies of structures it will be based on this concept.

3.1 Secondary Structure

The secondary structure of a biomolecule consists of local structural elements of the full tertiary structure of the molecule. For proteins this local information consists of segments of consecutive amino acids forming regular structural patterns. For RNA it is a set of pairs of bases, not necessarily located close to each other, forming strong interactions. An important use for secondary structure information is to reduce the number of degrees of freedom when trying to predict the full tertiary structure, but also as a help in aligning sequences and for inferring functional information does it find use. In the following we will briefly mention protein secondary structure prediction. We then give a thorough handling of RNA secondary structure prediction with special focus on the commonly used free energy model for RNA secondary structures.

3.1.1 Protein Secondary Structure Prediction

In protein structures small segments of consecutive amino acids are observed to form very regular structural patterns called α-helices and β-strands. The α-helix is a winding spiral resembling a corkscrew and it stabilises itself by formation of hydrogen bonds between neighbouring turns. The β-strands are stretched-out parts of the amino acid sequence that stabilises by forming hydrogen bonds to neighbouring β-strands that runs parallel or anti parallel to the strand in the structure but might be located far away in the sequence. Computationally protein secondary structure prediction is usually in the realm of pattern recognition, e.g. by neural networks [123, 126] or hidden Markov models [11]. The methods described in chapter 6 can be used in combination

with hidden Markov models for protein secondary structure prediction and for a family of protein sequences to predict the secondary structure of the family of sequences as represented by the hidden Markov model.

3.1.2 RNA Secondary Structure Prediction

In the tertiary structure of an RNA molecule the most distinctive feature is the stacking of base pairs. An RNA molecule, just like a DNA molecule, consists of a sugar/phosphate backbone with (almost) the same four amine bases as possible side chains. It is thus not surprising that complementary bases in an RNA molecule form strong interactions by hydrogen bonding similar to those found in the double helix of DNA. It should be mentioned that the four bases of RNA can form other types of hydrogen bonds than the classical Watson-Crick bonds, cf. [140], and as an RNA molecule forms a structure by folding back on itself, and not by forming base pairs with a molecule with a complementary base sequence as DNA, other base pairs than the classical Watson-Crick base pairs are observed in RNA structures. Only G, U base pairs, or wobble base pairs, are of so frequent occurrence to qualify as a standard RNA base pair, though.

Furthermore, as the bases mainly consists of flat, aromatic rings, consecutive base pairs can stack, much like Pringles® chips, to form compact helices shielding the hydrophobic aromatic structures from the surrounding aqueous environment. Or rather shielding the surrounding aqueous environment from the hydrophobic aromatic structures. Besides being a distinctive feature in RNA structures, the forces involved in these interactions are also quite strong, making them a key component governing the structure formation of an RNA molecule.

A secondary structure for an RNA sequence $s \in \{A, C, G, U\}^*$, cf. figure 7.1 on page 130, is a set $\mathcal{S}$ of base pairs $i \cdot j$ with $1 \leq i < j \leq |s|$ such that no base is paired with more than one other base, that is $\forall i \cdot j, i' \cdot j' \in \mathcal{S} : i = i' \Leftrightarrow j = j'$. Generally $\cdot$ is used as notation for base pairs – thus $X \cdot Y$ means any base pair formed between a base of type X and a base of type Y and $X_i \cdot Y_j$ means the specific $X \cdot Y$ formed between bases $s[i] = X$ and $s[j] = Y$. Often it is assumed that a secondary structure does not contain pseudoknots, that is overlapping base pairs $i \cdot j, i' \cdot j' \in \mathcal{S}$ with $i < i' < j < j'$. Nature does not prohibit pseudoknots, cf. [121], and numerous structures containing pseudoknots have been reported. In the following we will at first assume that structures do not contain pseudoknots and return to pseudoknots at the end of this section.

If $i < k < j$ and $i \cdot j \in \mathcal{S}$ then k is said to be accessible from $i \cdot j$ if there is no base pair between $i \cdot j$ and k, that is, if $\neg\exists i' \cdot j' \in \mathcal{S} : i < i' < k < j' < j$. If $k \cdot l \in \mathcal{S}$ (or $l \cdot k \in \mathcal{S}$) for some l, then it is an easy consequence of the absence of pseudoknots that l will also be accessible from $i \cdot j$ – otherwise we would have a base pair $i' \cdot j' \in \mathcal{S}$ with $k < i' < l < j'$ (with $i' < l < j' < k$) – and we say that the base pair $k \cdot l$ (the base pair $l \cdot k$) is accessible from $i \cdot j$. A further observation is that any base or base pair is accessible from at most one base pair. If a base or base pair is not accessible from any base pairs it is called external.

For each possible base pair $i \cdot j \in \mathcal{S}$ we can define the *loop* closed by that base pair – or having that base pair as exterior base pair – to be the loop formed by $i \cdot j$ and all bases and base pairs accessible from $i \cdot j$. The base pairs other than $i \cdot j$ in the loop are called interior base pairs. This means that we can partition a secondary structure into a number of loops, with any two loops being disjoint, except if the exterior base pair of the one loop is an interior base pair of the other loop in which case they overlap by one base pair. As illustrated in figure 7.1 on page 130 a loop closed by base pair $i \cdot j$ is named according to the number of interior base pairs and unpaired bases:

- With zero interior base pairs the loop is called a *hairpin loop*.
- With one interior base pair $i' \cdot j'$ the loop is called
  - a *stacking base pair* if there are no unpaired bases in the loop, that is, if $i' = i + 1$ and $j' = j - 1$. A stretch of consecutive stacking base pairs is called a *helix*.
  - a *bulge* if there are unpaired bases between the exterior and interior base pair only on one side, that is, if $i' = i + 1, j' < j - 1$ or if $i' > i + 1, j' = j - 1$.
  - an *internal loop* if there are unpaired bases between the exterior and interior base pair on both sides, that is, if $i' > i + 1$ and $j' < j - 1$.
- With more than one interior base pair the loop is called a *multibranched loop*.

Using this decomposition into loops Tinoco *et al.* [141] propose a model for calculating the energy of a secondary structure that has successfully celebrated its twenty fifth anniversary. The model is sufficiently close to reality so that calculations based on it yields decent results but still simple enough to allow for efficient algorithms for e.g. structure prediction. The model states that we can calculate the energy of a structure as a sum of independent contributions from each of the loops of the structure. Experimentally determining and estimating the energies of loops in RNA structures has been ongoing work at the Turner Group and the most recent parameters are published in [99].

Based on this model Zuker and Stiegler [167] and Nussinov and Jacobson [112] propose a recursive algorithm for finding the minimum energy of a structure for an RNA sequence s. A structure of this minimum energy can then be determined by backtracking the computations that yielded this energy. The algorithm works by finding the minimum energies of structures for substrings $s[i..j]$ of s closed by $i \cdot j$, that is, the structure for $s[i..j]$ is required to contain the base pair $i \cdot j$. When determining the optimal structure for the substring $s[i..j]$ we simply have to minimise over all choices of loops that $i \cdot j$ can close, the sum of the energy of the loop and the energies of optimal structures closed by the interior base pairs of the loop. The full algorithm – that is frequently called the `mfold` algorithm – is outlined in section 7.2 and has time complexity $\mathrm{O}(n^3)$ where $n = |s|$.

An examination of the above description of the algorithm, combined with counting the number of possible loops closed by a generic base pair brings out an apparent contradiction with the claimed time complexity. A specific base pair $i \cdot j$ will close $O((j-i)^2)$ different internal loops for a total of $O(n^4)$ different internal loops when summing over all possible base pairs. Multibranched loops do nothing to help in this situation by introducing an extra $O(2^n)$ loops.

With general energy parameters for loops, finding a structure of minimum energy for an RNA sequence in the model of Tinoco *et al.* would still be intractable (though the energy of a specific structure could still be computed efficiently). The `mfold` algorithm thus has to exploit specific properties of the energy functions to reduce the time requirements. Hence, the model thus only allows efficient algorithms for structure prediction when these properties are included in the model.

For multibranched loops it seems to be a good approximation to assume that the energy only depends on

- the size of the loop, that is, the number of unpaired bases and interior base pairs in the loop.
- exterior and interior base pairs stacking with neighbouring unpaired bases or base pairs.

Not only does this allow for a vast improvement in the time requirements for handling multibranched loops (though further approximations are applied to obtain a time complexity of $O(n^3)$ as we discuss below), but it would also be difficult akin to impossible to determine parameters for more complex types of energy functions.

For internal loops experiments have led to the adaption of an energy function that is a sum of energy parameters for

- the exterior and interior base pairs stacking with neighbouring unpaired bases.
- an energy function depending on the size of the loop, that is, the total number of unpaired bases in the loops.
- an energy function depending on the asymmetry of the loop, that is, the relation between the number of unpaired bases on each side of the exterior and interior base pairs.

If one ignores this last asymmetry term, Waterman and Smith [153] propose an $O(n^3)$ algorithm for the internal loop part of computing the minimum energy of an RNA secondary structure. This method, however, does not work with the type of asymmetry functions proposed by Papanicolaou *et al.* in [114]. This type of asymmetry function, called Ninio type asymmetry functions, is firmly established as it improves energy estimations. To retain a time complexity of $O(n^3)$ a commonly used heuristic is to upper bound the size of internal loops considered by some constant k (usually 30), as internal loops even close to this size are seldomly encountered. With this upper bound there are only $O(k^2)$

internal loops to consider for each base pair, thus reducing the total number of internal loops considered to $O(k^2n^2)$.

In chapter 7 we observe that asymmetry functions of the Ninio type only depends on the lopsidedness, i.e. the difference between the number of unpaired bases on each side of the base pairs, of an internal loop when the number of unpaired bases on each side of the base pairs is larger than some (small) constant. This is used to formulate a modified recursion that allows for constant time determination of the optimal interior base pair among most internal loops of the same size closed by a specific base pair $i \cdot j$. This reduces the time complexity of handling internal loops with an upper bound on the loop size of k to $O(kn^2)$; thus the time complexity of the unbounded, general case is reduced to $O(n^3)$.

Finding only the structure(s) of minimum energy might not give the full information one is interested in though. In some cases it might be of interest to be able to compare competing structures of low energies, not to mention the fact that the structure of minimum free energy on average only contains 73 % of the base pairs of the true secondary structure while a near-optimal structure often exists containing more of the true base pairs, cf. [99]. Zuker [164] proposes a base pair oriented method for finding suboptimal structures. Just as we can calculate the energy of an optimal structure of the subsequence $s[i..j]$ of an RNA sequence s with the restriction that the structure contains $i \cdot j$, we can calculate the energy of an optimal structure of s excluding $s[i+1..j-1]$ with the restriction that the structure contains $i \cdot j$; adding these two energies we get the energy of an optimal structure of s containing $i \cdot j$. This allows us to report structures for all base pairs that participates in a structure with an energy not to far from the optimal. McCaskill [101] presents a similar algorithm computing the full equilibrium partition function $\sum_{\mathcal{S}:i\cdot j\in\mathcal{S}} e^{E(\mathcal{S})/kT}$ for each base pair, thus allowing the computation of the base pair probability according to the Maxwell-Boltzmann distribution, cf. [7], as $\sum_{\mathcal{S}:i\cdot j\in\mathcal{S}} e^{E(\mathcal{S})/kT} / \sum_{\mathcal{S}} e^{E(\mathcal{S})/kT}$. By using a meticulous backtracking procedure Wuchty *et al.* [157] present an algorithm to generate all structures with an arbitrary upper bound on the energy, and in Cupal *et al.* [31] a method for finding the density of states, that is, the distribution of energies of structures for a sequence, is proposed. The method we suggest for handling internal loops in chapter 7 uses nothing but the commutativity and associativity of the order in which structures are handled and of combining structural elements. It thus applies quite generally, e.g. in all the cases just mentioned as well as in many of the recursions used by Rivas and Eddy [125] to predict structures containing pseudoknots.

A secondary structure for an RNA sequence can of course be used on its own right when looking for interesting features, e.g. like the anticodon of tRNA's that are located in the hairpin loop of the middle stem of the cloverleaf structure. Of other uses, Gruener *et al.* [49] use energy-based structure prediction to analyse connections between the space of RNA sequences and the space of secondary structures under the fold mapping (as defined by the structure prediction) and Hofacker [64] suggests RNA secondary structures as a tractable model for biopolymer folding, cf. section 3.2.2. Major *et al.* [98] report on using, among other information, the secondary structure of yeast tRNA^{Phe} as constraints to predict the three-dimensional structure.

Especially when trying to determine the tertiary structure it is questionable whether a secondary structure with only 73 % of the true base pairs is sufficient. It is thus highly desirable to improve on the predictions of secondary structure for RNA sequences. If several related sequences are available we can use the mutual information generated by sequence divergence while at the same time keeping a conserved structure to vastly improve structure determination. As only six of the sixteen possible combinations of bases form stable base pairs, often it is the case that when a base involved in a base pair is mutated, the base with which it pairs has to undergo a compensatory change for the base pair to be retained in the structure. Similarly, insertions and deletions might call for compensatory changes. One approach to using this information is simply to use the correlation between bases at any two positions in an alignment of the sequences as base pair scores, thus reducing the problem of determining the optimal structure to a maximum weighted matching problem (for which there is no need for the requirement of disallowing pseudoknots), cf. [136]. Other approaches include construction of stochastic context-free grammars [128, 76], covariance models [38] or simultaneous alignment of sequences and construction of a consensus structure [131, 45]. The authoritative method for determining the secondary structure of an RNA sequence, if the tertiary structure is not available, is comparative modelling, that is, manually constructing alignments revealing compensatory base pairs and insertions or deletions in related sequences.

Multibranched Loops

As mentioned in sections 6.6 and 6.7, when folding RNA sequences at elevated temperatures we did not predict any long range base pairings. Rather the predicted structure consisted of a number of structural fragments, each covering only a short subsequence. We conjecture that a major reason for this shortcoming of the algorithm might be the application of a linear penalty function for unpaired bases in multibranched loops.

If one assumes the contributions to the change in free energy of forming a multibranched loop is predominantly entropic, that is, the change in free energy is predominantly caused by the reduced number of attainable conformations, Jacobson and Stockmayer [72] show that the free energy's dependence on the size of the loop should be logarithmic.

An inspection of the energy parameters for RNA secondary structure formation reveals, that two stacking base pairs increase the stability of a structure while almost all other loops decrease the stability of a structure (for some small loops the added stability for the closing base pairs stacking with neighbouring unpaired bases may exceed the destabilising effect of the loop). As the stability of stacking base pairs decrease with temperature while the destabilising effects of loops increase, at elevated temperatures it requires longer helices of consecutively stacking base pairs to compensate for loops. One should thus expect multibranched loops to contain more unpaired bases at elevated temperatures.

Though a reasonable logarithmic function for multibranched loop stability can be approximated quite well by a linear function when the size of the loop is

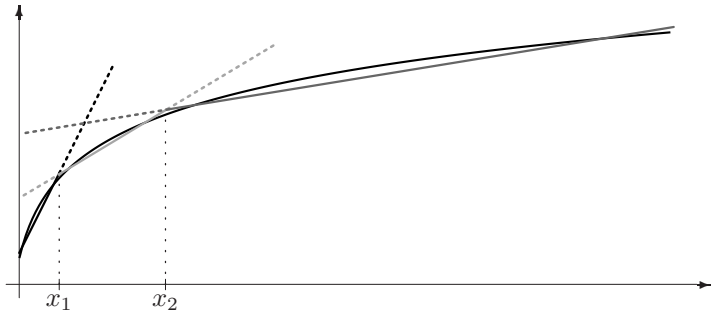


Figure 3.1: Approximation a logarithmic size dependence with a piecewise linear function – one linear function is used for sizes less than x_1, another for sizes between x_1 and x_2 and a third for sizes greater than x_2.

small, for loops with a larger number of unpaired bases the discrepancy becomes significant and the excess penalising of unpaired bases becomes prohibitively high. Compared to this, exterior unpaired bases, that is unpaired bases that are not part of any loop, are not penalised but considered neutral for the stability of the structure.

It is thus not surprising that the predicted structures at elevated temperatures only contains very few, if any, multibranched loops (the predicted structure of the *thermococcus celer* 23S subunit at 88 °C e.g. contains one multibranched loop with two internal base pairs and eight unpaired bases). One can expect a predicted structure at elevated temperatures to consist mostly of local structural fragments, that is, helices closing hairpin loops and internal loops and bulges (for which a logarithmic size dependence is used), but with most of the global, or long-range, structure formed by helices closing multibranched loops being absent.

Waterman and Smith [153] propose a general way to handle more complex size dependencies for multibranched loops, by extending the the *WM* array, that is, the array holding the energy of an optimal structure that constitutes a part of a multibranched loop (see section 7.2), with an extra index counting the number of unpaired bases in the structure. This method implies no restrictions on the size dependence and furthermore applies to calculating partition functions. It does however increase both the time and space complexities of the algorithm with a factor of n, for a $O(n^4)$ time and $O(n^3)$ space algorithm for RNA secondary structure prediction. A more careful accounting of the space usage of this algorithm reveals the constant hidden by the O to be roughly 1 byte (assuming entries in the arrays require 2 bytes each), and thus it would require approximately 128 MB to fold a sequence of 500 nucleotides with this algorithm – a sequence of 1000 nucleotides would require roughly 1 GB!

We propose meeting halfway between the tractability of a linear size dependence and the desirability of a logarithmic size dependence as a feasible alternative. Though a linear function cannot successfully imitate the wanted logarithmic dependence over the entire range of expectable loop sizes, for a smaller interval a good approximation can be achieved by a linear function.

The expected range of loop sizes can thus be divided into intervals, in each of which the desired size dependence can be approximated by a linear function as illustrated in figure 3.1, a method also mentioned for sequence comparison by Myers [107].

Assume that we want to use a piecewise linear function consisting of m linear functions to model the size dependence of multibranched loops,

$$\text{size}_{multibranch}(k,k') = bk + \begin{cases} a_1 + c_1 k' & \text{for } k' < x_1 \\ a_2 + c_2 k' & \text{for } x_1 \le k' < x_2 \\ \quad\vdots & \\ a_m + c_m k' & \text{for } x_{m-1} \le k' \end{cases}$$

where k is the number of interior base pairs (for simplicity we assume a linear dependence on the number of interior base pairs, but the technique also applies for more complex interdependencies between the number of interior base pairs and the number of unpaired bases) and k' is the number of unpaired bases in the multibranched loop. By concavity of the logarithmic function we want to imitate, it is fair to assume decreasing slopes for the linear functions, that is,

$$c_1 > c_2 > \ldots > c_m, \tag{3.1}$$

and that the value of any of the linear functions is less than the value of its neighbours at the endpoints of the interval it covers, that is,

$$a_l + c_l x_l \le a_{l-1} + c_{l-1} x_l \text{ and } a_{l-1} + c_{l-1}(x_l - 1) \le a_l + c_l(x_l - 1) \tag{3.2}$$

for $0 < l \le m$. These two assumptions combined yields that for $x_l \le k' < x_{l+1}$, $1 \le l \le m$ (with $x_0 = 0$ and $x_{m+1} = \infty$), the value of the linear function $a_l + c_l k'$ is less than the value of any of the other linear functions. We introduce a recursion similar to the one for WM on page 131,

$$WM_l(i,j) = \min\{V(i,j) + b, WM_l(i,j-1) + c_l, WM_l(i+1,j) + c_l, \min_{i<k\le j}\{WM_l(i,k-1) + WM_l(k,j)\}\},$$

for each segment $1 \le l \le m$. In this recursion $V(i,j)$ holds the minimum energy of a structure of $s[i..j]$ closed by $i \cdot j$; $WM_l(i,j)$ holds the minimum energy of a structure of $s[i..j]$ with at least one base pair, with base pairs and unpaired bases penalised according to the l'th size dependence function. We can now calculate the energy of the optimal multibranched loop closed by base pair $i \cdot j$ as

$$VM(i,j) = \min_{\substack{1\le l\le m \\ i+1<k\le j-1}} \{WM_l(i+1,k-1) + WM_l(k,j-1) + a_l\}.$$

An inspection of the above recursions reveals that we do not keep track of the number of unpaired bases. Instead each possible multibranched loop is evaluated using each of the linear functions, irregardless of the number of unpaired bases in the loop.

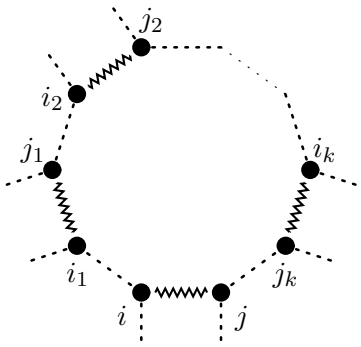


Figure 3.2: A multibranched loop with k interior base pairs.

So why does it work? Consider a specific candidate multibranched loop with k interior base pairs and k' unpaired bases, cf. figure 3.2. This loop will be considered for each of the $WM_l(i,j)$'s with a size dependence of $a_l + c_l k'$. For a specific l, the value of $WM_l(i,j)$ might thus be wrong in the sense that it is the energy of a loop with a number of unpaired bases that is not in the interval between x_{l-1} and x_l. But by equations 3.1 and 3.2 the energy with which the loop will be considered for the correct WM array, that is, the array for which the size dependence follows the linear function to be used with k' unpaired bases, will be lower. Thus we can trust the value of the minimum over all the $WM_l(i,j)$'s to be correct, though we can't be sure that the energy stored in a particular $WM_l(i,j)$ entry is correct. By this, the minimum value calculated for $VM(i,j)$ will be correct.

It is straightforward to verify that using m linear functions for the size dependence of multibranched loops requires time $\mathrm{O}(mn^3)$ and space $\mathrm{O}(mn^2)$ for the multibranched loop part of the RNA secondary structure prediction algorithm. So how large an m will be required? To get an estimate we assume that we want to imitate a logarithmic size dependence of 1.67 kcal/mol $\cdot \ln(k'+1)$ (fitting the previously[1] used linear dependence of 0.4 kcal/mol $\cdot\, k'$ for $k' = 0$ and $k' = 10$ – Zuker *et al.* [165] proposes a logarithmic size dependence of 1.079 kcal/mol $\cdot\, k'$). Table 3.1 shows for how large loops we can guarantee the discrepancy to be less than ϵ for different choices of m and ϵ. Considering the certainty with which parameters for RNA secondary structure can be determined, it seems that three or four linear functions will usually be sufficient. Instead of using a fixed maximum discrepancy as we did when calculating table 3.1, one can of course also design the piecewise linear function to keep within some discrepancy relative to the number of unpaired bases, or to be upper and lower bounded by two different functions.

The technique of approximating a complex function by a number of simpler functions is by no means new, but to our knowledge it has not been applied

[1] Mathews *et al.* [99] reports a set of improved energy parameters for RNA secondary structure prediction. These contain coefficients for both the linear and the logarithmic size dependency cases, but in the linear case the coefficient is zero and in the logarithmic case it is negative.

$\epsilon \backslash m$	1	2	3	4
0.1 kcal/mol	1	7	23	65
0.2 kcal/mol	3	19	84	348
0.3 kcal/mol	4	32	190	1081
0.5 kcal/mol	8	95	930	8949

Table 3.1: The number of unpaired bases we can handle for various choices of maximum discrepancy and number of linear functions.

to the problem of predicting multibranched loops in RNA secondary structures before. The perhaps greatest advantage of this technique is the ease of implementation. It requires little more than a **for**-loop around existing code for multibranched loop prediction. Furthermore it can be applied to other choices of concave size dependence functions. On the other hand it relies on the object being to find a minimum energy structure and thus does not apply to calculations of partition functions, cf. [101].

Pseudoknots

Usually it is assumed that any two base pairs, $i \cdot j$ and $i' \cdot j'$, of an RNA secondary structure are either nested, i.e. $i < i' < j' < j$, or disjoint, i.e. $i < j < i' < j'$. If the third possibility, the two base pairs being overlapping, i.e. $i < i' < j < j'$, is encountered, the structure is said to contain a pseudoknot. The term secondary structure is quite often even taken to mean a set of non-overlapping base pairs; thus Pleij [121] refers to unknotted structures as 'classical' or 'orthodox' secondary structures, Major *et al.* [98] uses the term 'tertiary base pairs' for four pseudoknot forming base pairs in the structure of yeast tRNA^{Phe}, and Sankoff [131] simply states it as a condition for secondary structures.

The reasons for this are several, but prominent among them are probably that a number of algorithms associated with RNA secondary structure, e.g. predicting secondary structure [167, 112], computing the full partition function [101], comparing secondary structures [161], simultaneous alignment and structure prediction of RNA sequences [131, 45] and stochastic models for RNA secondary structures [128, 38, 76], are unable to handle structures containing pseudoknots. Another reason is that the possible presence of a given pseudoknot involves spatial constraints – the constraints imposed by the other base pairs in a knotted structure might prevent the two bases of a candidate pseudoknot base pair being in the proximity of each other. The stability of a general pseudoknot base pair cannot be determined solely by local features like stacking base pairs etc., but depends on the tertiary structure of the RNA molecule.

Rivas and Eddy [125] propose a modification to the classical dynamic programming algorithm for RNA secondary structure prediction to allow for some pseudoknotted structures. The basic idea of the algorithm is to keep a region free for pseudoknot interactions when determining the energy of an optimal structure for subsequences of the RNA sequence. Tables are maintained such that the entries indexed by i, j, k, l holds the energy of an optimal structure for

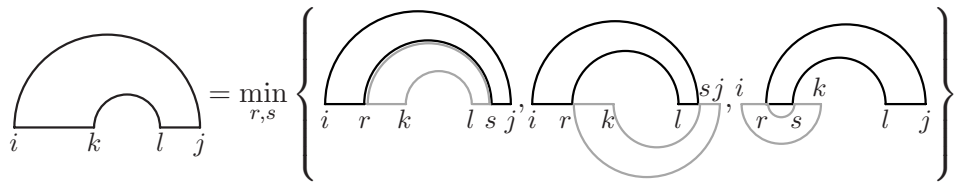


Figure 3.3: General recursion scheme for the Rivas/Eddy RNA secondary structure prediction algorithm.

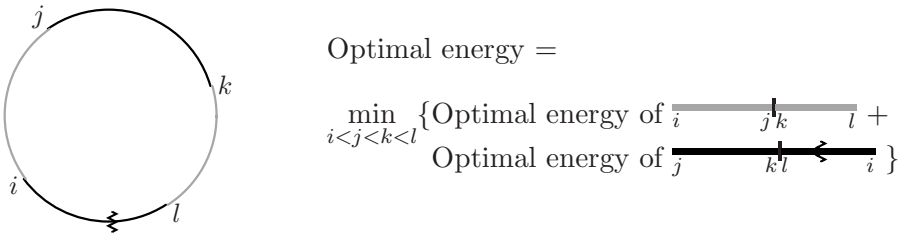


Figure 3.4: A model for a class of pseudoknots. The sequence has been drawn as a circle to highlight that one of the four parts of the sequence might extend across the sequence ends, here shown with a zigzagged line.

the subsequence from base i to base j with bases from k to l yet unpaired, and with various restrictions on which of the bases i, j, k, and l are paired and with what base. The basic recursion used to maintain these tables is illustrated in figure 3.3 – two indices are chosen where to split a subsequence with an unpaired region into two subsequences with unpaired regions. The requirements of time $O(n^6)$ and space $O(n^4)$ follows immediately from this recursion scheme.

In [88] we propose a more restricted model for structures containing pseudoknots based on the same principle. The sequence is split into four parts and the optimal structures of the two pairs of opposite structures are computed independently. Then the two structures are put together such that one forms the classical secondary structure while the other forms the pseudoknot interactions as illustrated in figure 3.4. Here we assume that the energies of the two structures are just added but more elaborate rules for combining the energies, e.g. based on which of the bases neighbouring the splitting points that are paired, can be used. Figure 3.4 also illustrates that an alternative view of the model is, that we find structures of sequences with a subsequence removed by which the analogy to the Rivas/Eddy algorithm is clear.

A straightforward algorithm to solve this problem would be to run through all the $O(n^4)$ choices of splits and compute the energy of the optimal structures of the two pairs of subsequences. This would require time $O(n^7)$ and space $O(n^2)$. One can however observe, that when we compute the energy of the optimal structure of the subsequence from base i to base l with the subsequence from base j to base k removed, we also compute the energy of the optimal

Algorithm 1 An algorithm for predicting RNA secondary structures containing pseudoknots based on the model illustrated in figure 3.4.

```
/* V_{j,k}(i,l) denotes the energy of the optimal structure for s[i..j] concatenated
with s[k..l]. */
E = ∞
for k = 1 to |s| do /* Fix one of the endpoints of the excluded region */
  Allocate memory for storing and calculating V_{j,k}(i,l) and V_{k-1,l}(j,i) for
  i < j < k < l
  /* Compute tables with k (or k − 1) as right (or left) endpoint of excluded
  region. */
  for j = 1 to k − 1 do
    Compute table V_{j,k}
  end for
  for l = k to |s| do
    Compute table V_{k-1,l}
  end for
  /* Combine tables. */
  for 1 ≤ i < j < k < l ≤ |s| do
    E = min{E, V_{j,k}(i,l) + V_{k-1,l+1}(j + 1, i − 1)}
  end for
  Free allocated memory
end for
```

structure of the subsequence from base i' to base l' with the subsequence from base j to base k removed for all $i \leq i' \leq j$ and $k \leq l' \leq l$. By using these intermediate results from the dynamic programming algorithm we can thus reduce the time requirements to $\mathrm{O}(n^5)$ by just running through all the $\mathrm{O}(n^2)$ choices of the removed subsequence. Unfortunately we then have to store some intermediate results until other results become available, increasing the space requirements to $\mathrm{O}(n^4)$. However, a more thorough investigation shows that the intermediate results computed with $k-1$ as the right endpoint of the removed subsequence are only combined with intermediate results computed with k as the left endpoint of the removed subsequence. This allows us to split the computation into n independent phases, each requiring only space $\mathrm{O}(n^3)$, thus reducing the overall space requirements to $\mathrm{O}(n^3)$ while maintaining the $\mathrm{O}(n^5)$ time requirements.

This method is sketched in algorithm 1, and one observes that the method only relies on intermediate results being computed and not the specific RNA secondary structure prediction used to compute the $V_{j,k}$ tables. One could thus use the Rivas/Eddy algorithm instead, or even apply the method recursively. Using an algorithm with time complexity $T(n)$ and space complexity $S(n)$ for computing the $V_{j,k}$ tables, will result in an algorithm with time complexity $\mathrm{O}(n^2T(n))$ and space complexity $\mathrm{O}(n^3 + S(n))$.

Despite the polynomial complexities of the above algorithms, it is arguable whether this justifies calling the problems they solve tractable. Based on our experiences when looking for large internal loops in natural occurring sequences,

cf. section 6.6, we will use the 4269 bases of Qβ as a rough upper bound on the length of sequences for which the classical $O(n^3)$ RNA secondary structure prediction algorithm finishes in reasonable time. Assuming similar constants hidden by the O notation this gives an upper bound of 65 bases for the Rivas/Eddy algorithm[2] and 150 bases for our $O(n^5)$ algorithm. Even a continued rapid increase in computing power does not do much to improve on the situation. Zuker and Stiegler [167] in 1981 report folding a sequence of 459 bases. Assuming an equivalent 1000-fold increase in computing power over the next 18 years, in 2017 the Rivas/Eddy algorithm should be able to handle sequences of up to 200 bases, and our $O(n^5)$ algorithm – that only considers a very restricted class of structures containing pseudoknots – should be able to handle sequences of up to 600 bases. Furthermore, as we will show in the following, there are sound reasons for restraining one's hopes for an efficient algorithm to predict general RNA secondary structures containing arbitrary pseudoknots.

Definition 1 (Nearest Neighbour Pseudoknot Model) *Let $\mathcal{S}$ be a secondary structure on a sequence $s \in \{A, C, G, U\}^*$, with $|s| = n$, without the unknotted restriction, that is, $\mathcal{S}$ is a set of base pairs $i \cdot j$ where $1 \leq i < j \leq n$ and $\forall i \cdot j, i' \cdot j' \in \mathcal{S} : i = i' \Leftrightarrow j = j'$. The energy of $\mathcal{S}$ is an independent sum of energies of each of the base pairs in $\mathcal{S}$,*

$$E(\mathcal{S}) = \sum_{i \cdot j \in \mathcal{S}} E(i \cdot j),$$

where the energy of a base pair $i \cdot j$ only depends on

- *the base pair itself, that is, the types of bases forming the pair.*
- *the two neighbouring bases $i + 1$ and $j - 1$, that is, the types of these two bases, and what base pairs they are forming if any.*

Proposition 1 *The problem of determining whether the optimal structure in the Nearest Neighbour Pseudoknot Model has energy lower than some energy value E is* **NP***-hard*[3]*.*

We will prove proposition 1 by a reduction to the special case of 3SAT where each literal occurs at most two times, cf. [113, proposition 9.3]. Throughout the proof of the proposition we will allow only Watson-Crick base pairs. This will become explicit in the final specification of the base pair energy function. Before proving proposition 1 we need some building blocks.

Definition 2 *The d digit* binary representation *of k, where $0 \leq k \leq 2^d - 1$, over the alphabet $\{A, U\}$, is the string $b_{\{A,U\}}(k, d)$ where $|b_{\{A,U\}}(k, d)| = d$ and $b_{\{A,U\}}(k, d)$ interpreted as a binary number with A representing 0 and U representing 1 equals k. Similarly $b_{\{C,G\}}(k, d)$ is the d digit binary representation*

[2] Rivas and Eddy [125] reports on folding a sequence of 105 bases. This stresses that our estimates should be taken as just that – mere estimates that depend on computing power, efficiency of implementation not to mention patience.

[3] As the problem trivially is in **NP** this implies that the problem is **NP**-complete.

of k over the alphabet $\{C, G\}$. The k'th distinct $\{A, U\}$ pattern *with d digit binary representations, $p_{\{A,U\}}(k, d)$, is the string*

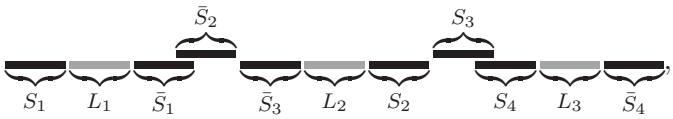
$$\underbrace{A \dots A}_{d+2} U b_{\{A,U\}}(k, d) A U A b_{\{A,U\}}(k, d) U \underbrace{A \dots A}_{d+2}$$

and similarly $p_{\{C,G\}}(k, d)$ is the k'th distinct $\{C, G\}$ pattern with d digit binary representations.

Definition 3 *For a string s the* complementary string $\bar{s}$ *is the string constructed by reversing s and replacing A's with U's, U's with A's, C's with G's and G's with C's.*

These distinct patterns and their complementary strings will be used to build an RNA sequence corresponding to a boolean formula on restricted 3SAT form, such that the energy of an optimal structure of the string implies whether the formula is satisfiable. The string will consist of two parts, a part where the literals are grouped according to the clauses and a part where the literals are grouped according to the variables. We will use distinct $\{C, G\}$ patterns and their complementary strings in the clauses and variables parts, respectively, to represent the literals of the formula. The distinct $\{A, U\}$ patterns and their complementary strings will be used to form structures nullifying the benefit of pairing a distinct $\{C, G\}$ pattern with its complementary string.

Definition 4 *Let $C = l_1 \vee l_2 \vee l_3$ be a boolean disjunction of three literals. The* clause block $\mathcal{C}$ *of C with d digit binary representations is the string*



where the S_i's are distinct $\{A, U\}$ patterns with d digits for four different k's, and the L_i's are distinct $\{C, G\}$ patterns with d digits for three different k's. The overlaps between $\bar{S}_1$ and $\bar{S}_2$ indicates that the terminal $d+2$ U's of $\bar{S}_1$ and initial $d+2$ U's of $\bar{S}_2$ are shared. Similarly the terminal $d+2$ A's of S_3 and initial $d+2$ A's of S_4 are shared.

The rationale behind this construction is, that if a structure $\mathcal{S}$ contains the helix formed by S_1 and its complementary pattern $\bar{S}_1$ it cannot also contain the helix formed by S_2 and $\bar{S}_2$. Similarly, a structure cannot at the same time contain the helix formed by S_3 and $\bar{S}_3$ and the helix formed by S_4 and $\bar{S}_4$. Furthermore, if both the helices formed by S_2 and S_3 and their complementary strings are present, there will be a base pair $i \cdot j \in \mathcal{S}$ with a neighbouring base pair forming a pseudoknot, that is, $i + 1 \cdot j' \in \mathcal{S}$ with $j' \notin \{i + 2, \dots, j - 1\}$.

If $\mathcal{S}$ is to be without neighbouring base pairs forming a pseudoknot, we can thus form helices of either S_1 and S_3 and their complementary strings blocking L_1 and L_2 – blocking meaning that if L_1 or L_2 form a helix with their complementary strings it will result in the innermost base pair of the

helix formed by either S_1 or S_3 having a neighbouring base pair forming a pseudoknot – of S_1 and S_4 blocking L_1 and L_3, or of S_2 and S_4 blocking L_2 and L_3. For a clause block we can thus form helices of two of the distinct patterns straightaway, and a third helix if we can pair one of the L_i patterns with its complementary string in the variables part.

Definition 5 *Let x be a variable occurring in a boolean formula where each literal occurs at most twice. The* variable block $\mathcal{V}$ *of x with d digit binary representations is the string*

$$\underbrace{}_{\bar{S}_1}\underbrace{}_{\bar{P}_1} G \underbrace{}_{\bar{P}_2}\underbrace{}_{S_1}\underbrace{}_{\bar{N}_1} G \underbrace{}_{\bar{N}_2}\underbrace{}_{\bar{S}_1},$$

where S_1 is a distinct $\{A, U\}$ pattern for some k, the $\bar{P}_i$'s are complementary strings to the distinct $\{C, G\}$ patterns used for the at most two positive occurrences of x (if x occurs positive only once, one of the $\bar{P}$ patterns is omitted from $\mathcal{V}$) and the $\bar{N}_i$'s are complementary strings to the distinct $\{C, G\}$ patterns used for the at most two negative occurrences of x (if x occurs negative only once, one of the $\bar{N}$ patterns is omitted from $\mathcal{V}$).

The rationale behind this construction is, that if a structure $\mathcal{S}$ contains the helix formed by S_1 and one of the two occurrences of its complementary string, this helix will block either the complementary strings corresponding to the two positive occurrences of x, or the complementary sequences corresponding to the two negative occurrences of x. If $\mathcal{S}$ is to be without neighbouring base pairs forming a pseudoknot, either the distinct pattern S_1 does not form a helix with one of its complementary strings, the complementary strings corresponding to the positive occurrences of x does not form helices, or the complementary strings corresponding to the negative occurrences of x does not form helices. We are now ready to define an RNA sequence representing a boolean formula on restricted 3SAT form.

Definition 6 *Let ϕ be a boolean formula on conjunctive normal form where each clause has 3 literals and each literal occurs at most three times. Assume that ϕ consists of c clauses and uses v variables. The RNA sequence corresponding to ϕ is the sequence*

$$s_\phi = \mathcal{C}_1 G \mathcal{C}_2 G \dots G \mathcal{C}_c G \mathcal{V}_1 \mathcal{V}_2 \dots \mathcal{V}_v,$$

where $\mathcal{C}_i$ is the clause block with $\lceil \log_2(4c+v) \rceil$ digit binary representations corresponding to the i'th clause of ϕ, $\mathcal{V}_i$ is the variable block with $\lceil \log_2(4c+v) \rceil$ digit binary representations corresponding to the i'th variable of ϕ, and no distinct pattern is used more than once.

The choice of number of digits ensures that we can choose at least $4c + v$ different values for distinct patterns. Each clause block uses four distinct $\{A, U\}$ patterns and three distinct $\{C, G\}$ patterns while each variable block uses one distinct $\{A, U\}$ pattern. Thus we do not run out of patterns. We will use

the term *complementary pattern* for the deliberate occurrences of the complementary string to a distinct pattern, that is, the strings indicated by a barred pattern in definitions 4 and 5.

So far we have assumed that helices only form between a distinct pattern and the complementary string designed to form a helix with it. Helices can of course form between parts of distinct patterns not designed to form helices together, but the following lemma limits the length of such helices.

Lemma 1 *Let s_ϕ be an RNA sequence constructed from a boolean formula ϕ according to definition 6. In any structure $\mathcal{S}$ of s_ϕ, any helix of consecutively stacking pairs of length at least $4d + 7$, where d is the number of digits used for the binary representations, will have at least $2d + 3$ bases at the end of a distinct pattern forming base pairs with the intended bases of (one of) the complementary pattern to this distinct pattern.*

Proof. By construction any substring of s_ϕ of length at least $4d + 7$ will contain at least $2d + 3$ bases from one of the ends of a distinct pattern or its complementary pattern. Consider one of the two substrings forming the helix. This will be of length at least $4d + 7$ and thus contain at least $2d + 3$ bases from a distinct pattern or its complementary pattern. Assume without loss of generality that it contains the first $2d + 3$ bases from $p_{\{A,U\}}(k, d)$, that is, the substring $A^{d+2}Ub_{\{A,U\}}(k, d)$. By construction, the only occurrences of $d + 2$ consecutive U's preceeded by an A in s_ϕ are at the ends of complementary patterns to distinct $\{A, U\}$ patterns, and thus $A^{d+2}Ub_{\{A,U\}}(k, d)$ forms base pairs with $\bar{b}_{\{A,U\}}(k', d)AU^{d+2}$ for some k' (by the assumption that only Watson-Crick base pairs are allowed). As $b_{\{A,U\}}(k, d)$ pairs with $\bar{b}_{\{A,U\}}(k', d)$ it follows that $k = k'$. □

We have now established that any helix of considerable length will contain at least part of a designed pairing. The next lemma establishes that this will be all it contains.

Lemma 2 *Let s_ϕ be an RNA sequence constructed from a boolean formula ϕ according to definition 6 with d digits used for the binary representations. In any structure $\mathcal{S}$ of s_ϕ, there are no helices of more than $4d + 9$ consecutively stacking base pairs containing only A's and U's or containing only C's and G's. The only helices of length $4d + 9$ containing only A's and U's or containing only C's and G's are helices formed by distinct patterns and (one of) their complementary pattern.*

Proof. By lemma 1 we know that a helix of length $4d + 9$ will contain one of the ends of a distinct pattern paired with its complementary pattern. All we have to show is, that we cannot extend a helix with an extra stacking pair of bases of the same type at the end of a helix formed by a distinct pattern and its complementary pattern.

If the distinct pattern is a $\{C, G\}$ pattern this is straightforward, as it will be in a clause block and thus bordered by an A and a U. Similarly a distinct $\{A, U\}$ pattern from a variable block will be bordered by two G's. A distinct

$\{A, U\}$ pattern from a clause block corresponding to S_1 or the complementary pattern corresponding to S_4 in definition 4 will be bordered by a C and a G. Finally, patterns corresponding to S_2 in definition 4 will have a C to its left and an A to its right not matching either the U to the right or the A to the left of its complementary pattern. Patterns corresponding to S_3 will have an A to its left and a U to its right not matching either the C to the right or the U to the left of its complementary pattern. □

Proof (of proposition 1). As mentioned above the reduction will be from 3SAT with the restriction that each literal appears at most twice. So let ϕ be a valid formula for this restriction of 3SAT with c clauses and v variables. In polynomial time, we can construct s_ϕ according to definition 6, and the base pair energy function

$$E(X_i \cdot Y_j, V_{i+1}, W_{j-1}) =$$
$$\begin{cases} -1 & \text{if } V_{i+1} \cdot W_{j-1} \in \mathcal{S} \text{ and either } X \cdot Y, V \cdot W \in \{A \cdot U, U \cdot A\} \\ & \text{or } X \cdot Y, V \cdot W \in \{C \cdot G, G \cdot C\} \\ 4d + 7 & \text{if } X \cdot Y \in \{A \cdot U, U \cdot A, C \cdot G, G \cdot C\} \text{ and for } j' \notin \{i+1, \ldots j-1\} \\ & \text{we have } V_{i+1} \cdot Z_{j'}, W_{j-1} \cdot Z_{j'}, Z_{j'} \cdot V_{i+1}, Z_{j'} \cdot W_{j-1} \notin \mathcal{S} \\ 4d + 8 & \text{otherwise} \end{cases}$$

where d is the number of digits used for the binary representations in s_ϕ and $\mathcal{S}$ is the structure for which the energy is calculated. Now we claim that the optimal structure of s_ϕ with the above energy function has energy $-(3c + v)$ if and only if ϕ is satisfiable.

By the energy function, the only helices for which the base pairs combined yields a negative contribution to the energy of the structure are helices of at least $4d + 9$ base pairs, base pairs that are either all A's pairing with U's or all C's pairing with G's. By lemma 2, the only such helices that can be formed are between distinct patterns and their complementary patterns; these helices will consist of exactly $4d + 9$ base pairs and thus contribute -1 to the total score of a structure, *provided* that the innermost base pair of the helix does not have a neighbouring base pair that forms a pseudoknot. If a distinct pattern is blocked[4] by a helix, it can thus not form a helix yielding a negative contribution to the total energy.

If there is an assignment of truth values to the variables of ϕ satisfying ϕ, we can construct a structure $\mathcal{S}$ on s_ϕ with energy $-(3c + v)$ based on this assignment by

- For each variable block forming the helix of the distinct $\{A, U\}$ pattern and the complementary pattern blocking the complementary patterns of the literals that become **false** by the assignment.

[4]The term 'block' is used here with the same meaning as in the discussion following definition 4. A helix blocks a distinct pattern (or a complementary pattern), if forming the helix between this pattern and its complementary pattern would result in the innermost base pair of the original helix getting a neighbouring base pair forming a pseudoknot.

- For each clause block forming two helices between distinct $\{A, U\}$ patterns that leaves the distinct $\{C, G\}$ pattern of a literal that becomes **true** by the assignment unblocked.

- Forming the helices between the unblocked distinct patterns of literals in the clauses part and their complementary patterns (that are unblocked as the assignment satisfies ϕ, and as the G between the two complementary patterns of a literal prevents these from interfering with each other) in the variables part.

By the discussion following definition 4, the distinct patterns of a clause block can form at most three helices, each yielding a contribution of -1, and each variable block introduces only one new distinct pattern; hence the energy of $\mathcal{S}$ of $-(3c + v)$ is optimal.

Assume now that s_ϕ has an optimal structure $\mathcal{S}$ of energy $-(3c + v)$. By the above and the discussion following definition 4, we get that each clause block will contain a distinct pattern corresponding to a literal forming a helix with its unblocked complementary pattern in the variables part, and that the complementary patterns corresponding either to a variable or to its negation will be blocked. We can thus infer a truth assignment to the variables of ϕ satisfying ϕ from the unblocked complementary patterns of literals in $\mathcal{S}$. □

The energy function specified in the proof of proposition 1 rewards stacking some base pairs, penalises loops by penalising the first base pair in a helix, and further penalises neighbouring base pairs that forms a pseudoknot. The only two remarkable oddities are the disallowance of base pairings between G and U, and penalising stacking an A, U base pair with a C, G base pair.

One can observe that we could allow G, U base pairs without having to change anything but swapping the distinct pattern and its complementary patterns in the variable blocks, and using C instead of G to separate clause blocks in s_ϕ. As for penalising stacking A, U base pairs with C, G base pairs, this was chosen to ease establishing the fact no energy benefits are obtained by extending a helix formed by a distinct pattern and its complementary pattern by further stacking base pairs. A proof where the energy function rewards stacking of all combinations of A, U base pairs, C, G base pairs and G, U base pairs can be achieved by slightly changing the construction of the variables part of s_ϕ. To limit the complexity of the proof, we have however chosen to present the above version.

Even with the oddities of the energy function mentioned above, proposition 1 tells us, that if $\mathbf{P} \neq \mathbf{NP}$ there is little hope for a worst case polynomial time algorithm for RNA secondary structure prediction in the nearest neighbour pseudoknot model or models extending it. An algorithm allowing energy functions sufficiently general to be specialised to the above values and running in worst case polynomial time would imply $\mathbf{P} = \mathbf{NP}$. Thus an algorithm for predicting RNA secondary structures with general pseudoknots would most likely have to exploit properties of a fixed, reasonable energy function to obtain polynomial running time.

By the above result it seems quite sensible to explore alternative approaches

to predict RNA secondary structures with pseudoknots. One such approach is to limit the types of pseudoknots considered as done by Rivas and Eddy [125] and in algorithm 1. Another approach is that taken by Tabaska *et al.* [136], where interactions with neighbouring base pairs are ignored, thus reducing the problem of RNA secondary structure prediction to compute a maximal weighted pairing. Numerous papers, e.g. [26], propose algorithms for predicting RNA secondary structures containing pseudoknots, that generates a set of (all the) helices that can be formed from the sequence, and then combines a compatible – compatible meaning that no two helices contain the same base – subset of these helices into a structure. Finally, heuristics can be applied to search for structures of low energy; van Batenburg *et al.* [147] reports on successful experiments with applying genetic algorithms to the problem of finding low energy RNA secondary structures containing pseudoknots.

3.2 Tertiary Structure

Being able to predict the tertiary structure of a protein (or an RNA molecule) is of much higher importance than predicting the secondary structure. First of all, from the full tertiary structure it is trivial to deduct the secondary structure. Secondly, the tertiary structure is a lot more informative for inferring functional properties. It should be mentioned, though, that a protein is not a completely rigid molecule but has a somewhat dynamic structure that vibrates around an equilibrium known as the *native state*. A change in the environment of the protein will often modify this equilibrium, a modification that is of importance during the catalytic processes the protein participates in cf. [40, section 1.4.1]. Thus the quaternary structure is the functionally most important structural level. Nevertheless the tertiary structure usually does not deviate dramatically from the quaternary structure, and furthermore reveals information, e.g. about the surface and electric field of the protein at the beginning of the catalytic process, helpful in deducting the pathway of the process. Much like predicting the secondary structure is a stepping stone towards predicting the tertiary structure, predicting the tertiary structure is a stepping stone towards understanding the functional aspects of a protein.

3.2.1 Structure Models

As always when trying to bring reality into a computer we need a model of the part of reality we are interested in. For proteins this involves a model for the protein molecule, a model of possible conformations, possibly including rules determining legal conformations, and, if we want to use the model for prediction, some objective, usually an energy function that should be minimised. For such a model to be relevant, it has to reflect at least part of the reality we are trying to model. For models for protein structure an obvious property to reflect is *visual equivalence* between conformations, and, hopefully, between conformations of minimum energy in the model and native states for real proteins. A more subtle, but useful, property is *behavioural equivalence*, that is, proteins in the model and in reality has some similar behavioural traits.

Once again claiming that proteins are part of reality, we may assume that they obey the laws of quantum mechanics. A fundamental model for protein structures is thus the Schrödinger equation

$$\hat{H}\psi = i\hbar\frac{\partial\psi}{\partial t},$$

cf. [32, equation 6.31]. Despite its apparent simplicity it contains several unwieldy or impossible elements, as determining the Hamiltonian operator $\hat{H}$ and solving the equation, either analytically or computationally.

In chemistry molecules are rarely viewed as wave functions, but rather as atoms connected by various kinds of bonds, that is, the classical balls-and-sticks model where the atoms are represented by balls that are connected by sticks representing the bonds. Using this as underlying model for molecules, we can specify the tertiary structure of a protein by giving the angles, lengths and torsions of all the bonds in the structure. To reduce complexity, some atoms, e.g. hydrogen atoms, are often omitted from the structural description or grouped; the entire side chain of an amino acid might thus be represented by a single 'superatom'. Models with this detailed chemical and physical description of the real world system are often called *analytic models*.

We can assign energies to structures in analytic models, e.g. by specifying energy functions for bond lengths, angles and torsions, possibly depending on local information on bond and atom types, and for non-local interactions, that is, energy functions depending on distances between non-bonded atoms. These energy functions might be based on physical principles like Coulombic and van der Waals forces or on statistical information obtained from known structures[5] known as mean force potentials [133]. These models tend to be at least intractable, e.g. finding the optimal structure in a model of this type has been proven **NP**-hard by Ngo and Marks [109].

A first step towards reducing the complexity of the model, is by restricting the attainable angles and lengths to finite sets of values; thus we could in theory solve the structure prediction problem in the model, simply by enumerating all possible conformations. As bond lengths only varies little, especially compared to bond angles and bond torsions, these are typically set to a fixed value depending on the bond type.

For angles and torsions, instead of choosing an arbitrary (uniform) distribution of values, these can be restricted to sets of values compiled from known structures, e.g. as by Pedersen and Moult [118], thus adding extra information derived from the real world to the model. Such models are often used as a starting point for finding good candidate conformations that can be refined in more fine-grained models. It is thus arguable whether they should be considered models of their own right, or just useful heuristics restricting the search space in a broader model. For a specific conformation the energy is usually determined by an energy function applicable to (or even stemming from) a general analytic model.

[5]Somewhat depressing, part of the progress in protein structure prediction in recent years, has been achieved by discarding our knowledge of intra- and intermolecular forces and replacing it by statistical information retrieved from databases of known structures.

Extending on this principle, instead of just compiling angles and torsions from known structures, we can compile structural segments from known structures, cf. [18]. Taking this to the limit where we fit a new protein to a complete known structure, we arrive at the threading problem suggested by Jones *et al.* [74]. This method has proven to be quite successful for protein structure prediction and methods using branch-and-bound (Lathrop and Smith [85]) or genetic algorithms (Yadgari *et al.* [159]) have been proposed. Even a polynomial time algorithm has been proposed for a class of threading problems (Xu and Uberbacher [158]), but Lathrop [84] has proven the general threading problem **NP**-complete.

Another path of pursuit, is to restrict the allowed values for angles and torsions to the point where legal conformations are embeddings in a lattice. These types of models are called *lattice models*, and it can be difficult to draw a clear line between lattice models and analytical models with fixed bond lengths and a very restricted set of allowed bond angles and torsions. A lattice model will often be characterised by neighbouring atoms (atoms connected by a bond) being required to occupy neighbouring lattice points, amino acids being grouped into one 'superatom', and the energy function only depending on non-local interactions, that is, pairs of atoms not connected by a bond occupying neighbouring lattice points.

Probably the most widely used type of lattice is the two- or three-dimensional square lattice (corresponding to an analytic model with bond angles restricted to multiples of 90° and torsions restricted to multiples of either 180° or 90° if neighbouring atoms are to occupy neighbouring lattice points). It is evident that square lattice models are not striving for visual equivalence with real proteins. Dill *et al.* [34] give an extensive report on experiments supporting some behavioural equivalences between square lattice models and real proteins, and Šali *et al.* [129] suggest only sequences with a pronounced minimum energy can be expected to fold to their minimum energy structures based on experiments in a square lattice model. The reason that we want to use lattice models for behavioural studies is of course that they are more wieldy than analytic models. For one thing it is preferable if we are able to determine the conformations of minimum energy in the model; unfortunately there exists **NP**-completeness results for various square lattice models [115, 145, 43].

One of the more popular square lattice models is the HP model proposed by Dill [33]. In the HP model, cf. section 8.2 for a detailed description, a protein is abstracted as a sequence of superatoms, each representing an entire amino acid that is either hydrophobic or hydrophilic; we will use 0's to denote hydrophiles and 1's to denote hydrophobes, such that a protein in the HP model is a sequence over the alphabet $\{0, 1\}$; a legal conformation is a self-avoiding embedding where consecutive amino acids occupies neighbouring lattice point; the energy is proportional to the number of non-local hydrophobic bonds, that is, pairs of non-consecutive hydrophobic amino acids occupying neighbouring lattice points, cf. figure 8.1 on page 150. The first approximation algorithm for protein structure prediction was proposed by Hart and Istrail [57], cf. section 3.2.2, for the HP model, and for some time there was hope that the HP model would allow for efficient determination of the conformations of mini-

mum energy. However, this problem has recently been proven **NP**-complete, cf. [17, 28]. In this apparent absence of tractable models for protein structure formation, Hofacker [64] has suggested the RNA secondary structure model, cf. section 3.1.2, as a suitable model for studying structure formation.

3.2.2 Approximation Algorithms in the HP Model

Approximation algorithms are algorithms that yield solutions guaranteed to be within some range from the optimal solution. In the HP model the energy of a structure is proportional to the number of non-local hydrophobic bonds. For a sequence $s \in \{0,1\}^*$, an approximation algorithm should thus find a structure with a number of non-local hydrophobic bonds $\mathcal{A}(s)$ that is within some predetermined range from the number of non-local hydrophobic bonds in the optimal structure $\mathrm{OPT}(s)$. In this section we will focus on the case where this guarantee is given as an *approximation ratio*, that is, the deviation is in terms of a multiplicative constant. An approximation ratio r is called *absolute* if $\forall s \in \{0,1\}^* : \mathcal{A}(s) \geq r \cdot \mathrm{OPT}(s)$. In some cases we might fall short of obtaining such a guarantee by e.g. an additive constant, or at least an additive term that becomes negligible small relative to $\mathrm{OPT}(s)$ when $\mathrm{OPT}(s)$ becomes large. If $\forall \epsilon > 0 \ \exists k \ \forall s \in \{0,1\}^* : \mathrm{OPT}(s) \geq k \Rightarrow \mathcal{A}(s) \geq (r - \epsilon) \cdot \mathrm{OPT}(s)$ we will thus refer to r as an *asymptotic* approximation ratio.

Hart and Istrail [57] present an approximation algorithm for the two-dimensional HP model[6] based on separating the hydrophobes into two sets, $\mathrm{EVEN}(s)$ that are the hydrophobes in even-indexed positions in s, and $\mathrm{ODD}(s)$ that are the hydrophobes in the odd-indexed positions in s. By 'checkerboarding' the lattice points they obtain a $2 \cdot \min\{|\mathrm{EVEN}(s)|, |\mathrm{ODD}(s)|\}$ upper bound for $\mathrm{OPT}(s)$, cf. equation 8.1, and observe that for any sequence s

- there is an index $1 \leq i \leq |s|$ such that at least half of the hydrophobes in even-indexed positions are on one side of i and at least half of the hydrophobes in odd-indexed positions are on the other side of i.
- for a substring of s we can fold loops out to one side, making a *face* or *stem* where the hydrophobes in even-indexed (or odd-indexed) positions of the substring occupy every other lattice point.

This leads to an approximation algorithm with an asymptotic approximation ratio of 1/4. The algorithm has time complexity $\mathrm{O}(|s|)$, and works by dividing the sequence at i and folding a face of hydrophobes in even-indexed positions against a face of hydrophobes in odd-indexed positions in what we call a U-fold, cf. figure 8.2 on page 151. A slight modification leads to an absolute approximation ratio of 1/4, cf. [57].

In section 8.3 we present several attempts to improve on the Hart/Istrail approximation algorithm. First of all, instead of only allowing hydrophobes from either $\mathrm{EVEN}(s)$ or $\mathrm{ODD}(s)$ in a face, we might as well find optimal faces to fold

[6] Hart and Istrail [57] also present an approximation algorithm for the three-dimensional HP model with an approximation ratio of 3/8. The principles of this algorithm are similar to those of the algorithm for the two-dimensional case.

against each other. This can be done in time $O(|s|^2)$, cf. section 8.3, but unfortunately does not improve on the approximation ratio. Two generalisations of this, cf. section 8.3, are

- the S-fold, cf. figure 8.5(a) on page 153: Instead of just two faces, we stack an arbitrary number of faces where only the two outermost faces can be contracted by folding loops out to one side. It can be proven that this generalisation does not lead to an approximation ratio better than 1/4.

- the C-fold, cf. figure 8.5(b) on page 153: Instead of having the two faces consisting of a prefix and the corresponding suffix of s, we might as well have one of them extending 'across the ends' of s. By this we mean constructing one of the faces from a substring in the middle of s and the other from the remaining ends of s. We have not been able to prove that this leads to an approximation ratio better than 1/4, but by a transformation to what we call the *circle problem*, cf. section 8.4, we can prove that the C-fold generalisation does not lead to an approximation ratio better than 1/3. Furthermore, from the circle problem we have obtained experimental evidence supporting an approximation ratio of 1/3.

The optimal structure for both these two types of structures can be determined in time $O(|s|^3)$ by dynamic programming.

Hart and Istrail [57] also mention the C-fold generalisation. As their method only considers hydrophobes in even-indexed positions in one of the faces forming non-local hydrophobic bonds with hydrophobes in odd-indexed positions in the other face, they can however prove that it does not improve on the approximation ratio of 1/4. Mauri *et al.* [100] also present an approximation algorithm for the two-dimensional HP model formulated in terms of a context-free grammar for $\{0,1\}^*$. Parsings by this grammar lead to C-fold structures, and by an algorithm to find the most probable parsing for stochastic context-free grammars they find the optimal structure in time $O(|s|^3)$. Based on experiments they conjecture a 3/8 approximation ratio, a discrepancy with the upper bound of 1/3 we establish. This discrepancy can be explained by the empirical basis of their conjecture, and the fact that they also count non-local hydrophobic bonds in the loops contracting the stems. Our algorithms can without increasing the time complexity be modified to count these non-local hydrophobic bonds too, but they complicate the analysis of the approximation ratio and we conjecture that counting them will not improve the approximation ratio.

So what is the use of these approximation algorithms? Hart and Istrail [57] suggest that the structures determined might resemble the molten globule state, an intermediate state with compact structures observed in real protein structure formation, based on arguments that to a varying degree can be considered sound. Furthermore, many approximation algorithms perform much better than their guaranteed ratios. Hart and Istrail [56, 58] show how to transform an approximate result to more realistic lattices than the square lattice. It is thus conceivable that we could use approximate structures in the two-dimensional HP model as a starting point towards finding good structures in realistic analytic models.

The approximation algorithms here do however probably fail in vastly outperforming their guarantees, as we only look for one non-local hydrophobic bond for each hydrophobe instead of the maximum of two; a structure with more than around 50% non-local hydrophobic bonds compared to the optimal structure should thus not be expected. Furthermore, even more is lost in the transformation to more realistic lattices. These approximation algorithms for protein folding in simple lattice models should thus probably, even more so than the problem of finding maximal pairs presented in section 2.1, be termed as solving problems inspired by biology and not as solving biological problem.

Part II

Papers

Chapter 4

Finding maximal pairs with bounded gap

> Gemini: The Twins
>
> Another of Ptolemy's original forty-eight groups, and one of the constellations of the Zodiac. It takes its name from Castor and Pollux, two of the mythological heroes—twin boys, sons of a Spartan king, Tyndarus, and his queen Leda.
>
> —Patrick Moore, *The Observer's Book of Astronomy*

This paper describes methods to find repeated occurrences of strings within a bounded distance from each other in a sequence. The results were presented at the Tenth Annual Symposium on Combinatorial Pattern Matching and a short version of the paper, not containing the section describing how to achieve linear time when removing the upper bound on gap size, is published in the proceedings of this conference [19]. Furthermore, the paper has been published as a technical report in the BRICS report series [20]. The algorithms presented have not been implemented.

Finding maximal pairs with bounded gap

Gerth Stølting Brodal* Rune B. Lyngsø*
Christian N. S. Pedersen* Jens Stoye†

Abstract

A pair in a string is the occurrence of the same substring twice. A pair is maximal if the two occurrences of the substring cannot be extended to the left and right without making them different. The gap of a pair is the number of characters between the two occurrences of the substring. In this paper we present methods for finding all maximal pairs under various constraints on the gap. In a string of length n we can find all maximal pairs with gap in an upper and lower bounded interval in time $O(n \log n + z)$ where z is the number of reported pairs. If the upper bound is removed the time reduces to $O(n + z)$. Since a tandem repeat is a pair where the gap is zero, our methods can be seen as a generalization of finding tandem repeats. The running time of our methods equals the running time of well known methods for finding tandem repeats.

4.1 Introduction

A pair in a string is the occurrence of the same substring twice. A pair is left-maximal (right-maximal) if the characters to the immediate left (right) of the two occurrences of the substring are different. A pair is maximal if it is both left- and right-maximal. The gap of a pair is the number of characters between the two occurrences of the substring. For example, the two occurrences of the substring *ma* in the string *maximal* form a maximal pair of *ma* with gap two.

Gusfield [52, Section 7.12.3] describes how to report all maximal pairs in a string using the suffix tree of the string in time $O(n+z)$ and space $O(n)$, where n is the length of the string and z is the number of reported pairs. Since there is no restriction on the gap of the maximal pairs reported by this algorithm, many of them probably describe occurrences of substrings that are overlapping or far apart in the string. In many applications in computational biology this is unfortunate, so several papers address the problem of finding occurrences of similar substrings not too far apart [75, 86, 127].

In the first part of this paper we describe how to find all maximal pairs in a string with gap in an upper and lower bounded interval in time $O(n \log n + z)$ and space $O(n)$. The interval of allowed gaps can be chosen such that we

*Basic Research in Computer Science (BRICS), Centre of the Danish National Research Foundation, Department of Computer Science, University of Aarhus, Ny Munkegade, 8000 Århus C, Denmark. E-mail: {gerth,rlyngsoe,cstorm}@brics.dk.

†Deutsches Krebsforschungszentrum (DKFZ), Theoretische Bioinformatik, Im Neuenheimer Feld 280, 69120 Heidelberg, Germany. E-mail: j.stoye@dkfz-heidelberg.de

report a maximal pair only if the gap is between constants c_1 and c_2, but more generally, it can be chosen such that we report a maximal pair of α only if the gap is between $g_1(|\alpha|)$ and $g_2(|\alpha|)$, where g_1 and g_2 are functions that can be computed in constant time. This, for example, makes it possible to find all maximal pairs with gap between zero and some fraction of the length of the repeated substring. In the second part of this paper we describe how removing the upper bound $g_2(|\alpha|)$ on allowed gaps, and only require the gap of a reported pair of α to be at least $g_1(|\alpha|)$, makes it possible to reduce the running time to $O(n+z)$. The methods we present all use the suffix tree as the fundamental data structure combined with efficient methods for merging search trees and heap-ordered trees.

The problem of finding occurrences of repeated substrings in a string is well studied. Most of the work has been concerned with efficient methods for finding occurrences of contiguously repeated substrings. An occurrence of a substring of the form $\alpha\alpha$ is called an occurrence of a square or a tandem repeat. Most well-known methods for finding the occurrences of all tandem repeats in a string require time $O(n \log n + z)$, where n is the length of the string and z is the number of reported occurrences of tandem repeats [29, 9, 96, 78, 135]. Work has also been done on just detecting whether or not a string contains a tandem repeat [97, 30]. Recently, extending on the idea presented in [30], two methods have been presented that find a compact representation of all tandem repeats in a string in time $O(n)$ [77, 53]. Other papers consider the problem of finding occurrences of contiguous repeats of substrings that are within some Hamming- or edit-distance of each other [82].

In biological sequence analysis searching for tandem repeats is used to reveal structural and functional information [52, pp. 139–142], but searching for exact tandem repeats can be too restrictive because of sequencing and other experimental errors. By searching for maximal pairs with small gaps (maybe depending on the length of the substring) it could be possible to compensate for these errors. On the other hand, finding maximal pairs with a gap within an interval can be seen as a generalization of finding occurrences of tandem repeats. Stoye and Gusfield [135] say that an occurrence of the tandem repeat $\alpha\alpha$ is a branching occurrence of the tandem repeat $\alpha\alpha$ if and only if the characters to the immediate right of the two occurrences of α are different, and they explain how to deduce the occurrence of all tandem repeats in a string from the occurrences of branching tandem repeats in time proportional to the number of tandem repeats. Since a branching occurrence of a tandem repeat is just a right-maximal pair with gap zero, the methods presented in this paper can be used to find all tandem repeats in time $O(n \log n + z)$. This matches the time bounds of previous published methods for this problem [29, 9, 96, 78, 135].

The rest of this paper is organized in two parts which can be read independently. In Section 4.2 we present the preliminaries necessary to read either of the two parts; we define pairs and suffix trees and describe how in general to find pairs using the suffix tree. In the first part, Section 4.3, we present the methods to find all maximal pairs in a string with gap in an upper and lower bounded interval. This part also presents facts about efficient merging of search trees which are essential to the formulation of the methods. In the second part,

Section 4.4, we present the methods to find all maximal pairs in a string with gap in a lower bounded interval. This part also includes the presentation of two novel data structures, the heap-tree and the colored heap-tree, which are essential to the formulation of the methods. Finally, in Section 4.5 we summarize our work and discuss open problems.

4.2 Preliminaries

Throughout this paper S will denote a string of length n over a finite alphabet Σ. We will use $S[i]$, for $i = 1, 2, \ldots, n$, to denote the ith character of S, and use $S[i\,..\,j]$ as notation for the substring $S[i]S[i+1]\cdots S[j]$ of S. To be able to refer to the characters to the left and right of every character in S without worrying about the first and last character, we define $S[0]$ and $S[n+1]$ to be two distinct characters not appearing anywhere else in S.

In order to formulate methods for finding repetitive structures in S, we need a proper definition of such structures. An obvious definition is to find all pairs of identical substrings in S. This, however, leads to a lot of redundant output, e.g. in the string that consists of n identical characters there are $\Theta(n^3)$ such pairs. To limit the redundancy without sacrificing any meaningful structures Gusfield [52] defines maximal pairs.

Definition 7 (Pair) *We say that $(i, j, |\alpha|)$ is a pair of α in S formed by i and j if and only if $1 \leq i < j \leq n - |\alpha| + 1$ and $\alpha = S[i\,..\,i+|\alpha|-1] = S[j\,..\,j+|\alpha|-1]$. The pair is left-maximal (right-maximal) if the characters to the immediate left (right) of two occurrences of α are different, i.e. left-maximal if $S[i-1] \neq S[j-1]$ and right-maximal if $S[i+|\alpha|] \neq S[j+|\alpha|]$. The pair is maximal if it is right- and left-maximal. The gap of a pair $(i, j, |\alpha|)$ is the number of characters $j - i - |\alpha|$ between the two occurrences of α in S.*

It follows from the definition that a string of length n in the worst case contains $\Theta(n^2)$ right-maximal pairs. The string a^n contains the worst case number of right-maximal pairs but only $\Theta(n)$ maximal pairs. The string $(aab)^{n/3}$ however contains $\Theta(n^2)$ maximal pairs. This shows that the worst case number of maximal pairs and right-maximal pairs in a string are asymptotically equal.

Figure 4.1 illustrates the occurrence of a pair. In some applications it might be interesting only to find pairs that obey certain restrictions on the gap, e.g. to filter out pairs of substrings that are overlapping or far apart and thus to reduce the number of pairs to report. Using the "smaller-half trick", see Section 4.3.1, and Lemma 3 it is easy to prove that a string of length n in the worst case contains $\Theta(n \log n)$ right-maximal pairs with gap in an interval of constant size.

In this paper we present methods for finding all right-maximal and maximal pairs $(i, j, |\alpha|)$ in S with gap in a bounded interval. These methods all use the suffix tree of S as the fundamental data structure. We briefly review the suffix tree and refer to [52] for a more comprehensive treatment.

Definition 8 (Suffix tree) *The suffix tree $T(S)$ of the string S is the compressed trie of all suffixes of S. Each leaf in $T(S)$ represents a suffix $S[i\,..\,n]$*

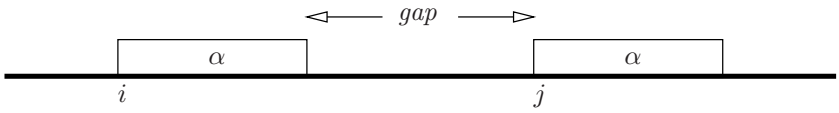


Figure 4.1: An occurrence of a pair $(i, j, |\alpha|)$ with gap $j - i - |\alpha|$.

of S and is annotated with the index i. We refer to the set of indices stored at the leaves in the subtree rooted at node v as the leaf-list of v and denote it $LL(v)$. Each edge in $T(S)$ is labelled with a nonempty substring of S such that the path from the root to the leaf annotated with index i spells the suffix $S[i\,..\,n]$. We refer to the substring of S spelled by the path from the root to node v as the path-label of v and denote it $L(v)$.

The suffix tree $T(S)$ can be constructed in time $O(n)$ [155, 102, 144, 41]. It follows from the definition that all internal nodes in $T(S)$ have out-degree between two and $|\Sigma|$. We can turn the suffix tree $T(S)$ into the binary suffix tree $T_B(S)$ by replacing every node v in $T(S)$ with out-degree $d > 2$ by a binary tree with $d - 1$ internal nodes and $d - 2$ internal edges in which the d leaves are the d children of node v. We label each new internal edge with the empty string such that the $d - 1$ nodes replacing node v all have the same path-label as node v has in $T(S)$. Since $T(S)$ has n leaves, constructing the binary suffix tree $T_B(S)$ requires adding at most $n - 2$ new nodes. Since each new node can be added in constant time, the binary suffix tree $T_B(S)$ can be constructed in time $O(n)$.

The binary suffix tree is an essential component of our methods. Definition 8 implies that there is a node v in $T(S)$ with path-label α if and only if α is the longest common prefix of $S[i\,..\,n]$ and $S[j\,..\,n]$ for some $1 \leq i < j \leq n$. In other words, there is a node v with path-label α if and only if $(i, j, |\alpha|)$ is a right-maximal pair in S. Since $S[i + |\alpha|] \neq S[j + |\alpha|]$ the indices i and j cannot be elements in the leaf-list of the same child of v. Using the binary suffix tree $T_B(S)$ we can thus formulate the following lemma.

Lemma 3 *There is a right-maximal pair $(i, j, |\alpha|)$ in S if and only if there is a node v in the binary suffix tree $T_B(S)$ with path-label α and distinct children w_1 and w_2 where $i \in LL(w_1)$ and $j \in LL(w_2)$.*

Lemma 3 gives an approach to find all right-maximal pairs in S; for every internal node v in the binary suffix tree $T_B(S)$ consider the leaf-lists at its two children w_1 and w_2, and for every element (i, j) in $LL(w_1) \times LL(w_2)$ report a right-maximal pair $(i, j, |\alpha|)$ if $i < j$ and $(j, i, |\alpha|)$ if $j < i$. To find all maximal pairs in S the problem remains to filter out all right-maximal pairs that are not left-maximal.

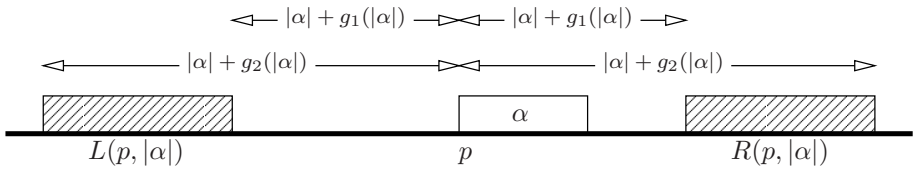


Figure 4.2: If $(p, q, |\alpha|)$ (respectively $(q, p, |\alpha|)$) is a pair with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$, then one occurrence of α is at position p and the other occurrence is at a position q in the interval $R(p, |\alpha|)$ (respectively $L(p, |\alpha|)$) of positions.

4.3 Pairs with upper and lower bounded gap

We want to find all maximal pairs $(i, j, |\alpha|)$ in S with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$, i.e. $g_1(|\alpha|) \leq j - i - |\alpha| \leq g_2(|\alpha|)$, where g_1 and g_2 are functions that can be computed in constant time. An obvious approach is to generate all maximal pairs in S and only report those with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$, but as shown above there might be asymptotically fewer maximal pairs in S with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$ than maximal pairs in S in total. We therefore want to find all maximal pairs $(i, j, |\alpha|)$ in S with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$ *without* generating and considering all maximal pairs in S. A step towards finding all maximal pairs with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$ is to find all right-maximal pairs with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$.

Figure 4.2 shows that if one occurrence of α in a pair with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$ is at position p, then the other occurrence of α must be at a position q in one of the two intervals $L(p, |\alpha|) = [\, p - |\alpha| - g_2(|\alpha|) \,..\, p - |\alpha| - g_1(|\alpha|) \,]$ or $R(p, |\alpha|) = [\, p + |\alpha| + g_1(|\alpha|) \,..\, p + |\alpha| + g_2(|\alpha|) \,]$. Together with Lemma 3 this gives an approach to find all right-maximal pairs in S with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$; from every internal node v in the binary suffix tree $T_B(S)$ with path-label α and children w_1 and w_2, we report for every p in $LL(w_1)$ the pairs $(p, q, |\alpha|)$ for all q in $LL(w_2) \cap R(p, |\alpha|)$ and the pairs $(q, p, |\alpha|)$ for all q in $LL(w_2) \cap L(p, |\alpha|)$.

To report right-maximal pairs efficiently using this procedure, we must be able to find for every p in $LL(w_1)$, without looking at all the elements in $LL(w_2)$, the proper elements q in $LL(w_2)$ to report it against. It turns out that search trees make this possible. In this paper we use AVL trees, but other types of search trees, e.g. (a, b)-trees [68] or red-black trees [50], can also be used as long as they obey Lemmas 4 and 5 stated below. Before we can formulate algorithms we review some useful facts about AVL trees.

4.3.1 Data Structures

An AVL tree T is a balanced search tree that stores an ordered set of elements. AVL trees were introduced in [6], but are explained in almost every textbook on data structures. We say that an element e is in T, or $e \in T$, if it is stored at a node in T. For short notation we use e to denote both the element and the node at which it is stored in T. We can keep links between the nodes in T

in such a way that we in constant time from the node e can find the nodes $next(e)$ and $prev(e)$ storing the next and previous element in increasing order. We use $|T|$ to denote the size of T, i.e. the number of elements stored in T.

Efficient merging of two AVL trees is essential to our methods. Hwang and Lin [70] show how to merge two sorted lists using the optimal number of comparisons. Brown and Tarjan [22] show how to implement merging of two height-balanced search trees, e.g. AVL trees, in time proportional to the optimal number of comparisons. Their result is summarized in Lemma 4, which immediately implies Lemma 5.

Lemma 4 *Two AVL trees of size at most n and m can be merged in time $O(\log \binom{n+m}{n})$.*

Lemma 5 *Given a sorted list of elements $e_1, e_2, \ldots, e_n$ and an AVL tree T of size at most m, $m \geq n$, we can find $q_i = \min\{x \in T \mid x \geq e_i\}$ for all $i = 1, 2, \ldots, n$ in time $O(\log \binom{n+m}{n})$.*

Proof. Construct the AVL tree of the elements $e_1, e_2, \ldots, e_n$ in time $O(n)$. Merge this AVL tree with T according to Lemma 4, except that whenever the merge-algorithm would insert one of the elements $e_1, e_2, \ldots, e_n$ into T, we change the merge-algorithm to report the neighbor of the element in T instead. This modification does not increase the running time. □

The "smaller-half trick" is essential to several methods for finding tandem repeats [29, 9, 135]. It says that the sum over all nodes v in an arbitrary binary tree of size n of terms that are $O(n_1)$, where $n_1 \leq n_2$ are the numbers of leaves in the subtrees rooted at the two children of v, is $O(n \log n)$. Our methods rely on a stronger version of the "smaller-half trick" hinted at in [104, Ex. 35] and used in [105, Chap. 5, p. 84]; we summarize it in the following lemma.

Lemma 6 *Let T be an arbitrary binary tree with n leaves. The sum over all internal nodes v in T of terms that are $O(\log \binom{n_1+n_2}{n_1})$, where n_1 and n_2 are the numbers of leaves in the subtrees rooted at the two children of v, is $O(n \log n)$.*

Proof. As the terms are $O(\log \binom{n_1+n_2}{n_1})$ we can find constants, a and b, such that the terms are upper bounded by $a + b \log \binom{n_1+n_2}{n_1}$. We will by induction in the number of leaves of the binary tree prove that the sum is upper bounded by $(2n-1)a + b \log n!$. As $\log n! = O(n \log n)$ the lemma follows.

If T is a leaf the upper bound holds vacuously. Now assume inductively that the upper bound holds for all trees with at most $n-1$ leaves. Let T be a tree with n leaves where the number of leaves in the subtrees rooted at the two children of the root are $n_1 < n$ and $n_2 < n$. According to the induction hypothesis the sum over all nodes in these two subtrees, i.e. the sum over all nodes of T except the root, is bounded by $(2n_1 - 1)a + b \log n_1! + (2n_2 - 1)a + b \log n_2!$ and thus

the entire sum is bounded by

$$\begin{aligned}(2n_1-1)a+&b\log n_1! + (2n_2-1)a + b\log n_2! + a + b\log\binom{n_1+n_2}{n_1}\\ &= (2(n_1+n_2)-1)a + b\log n_1! + b\log n_2! + \\ &\qquad b\log(n_1+n_2)! - b\log n_1! - b\log n_2!\\ &= (2n-1)a + b\log n!\end{aligned}$$

which proves the lemma. □

4.3.2 Algorithms

We first describe an algorithm that finds all right-maximal pairs in S with bounded gap using AVL trees to keep track of the elements in the leaf-lists during a traversal of the binary suffix tree $T_B(S)$. We then extend it to find all maximal pairs in S with bounded gap using an additional AVL tree to filter out efficiently all right-maximal pairs that are not left-maximal. Both algorithms run in time $O(n\log n + z)$ and space $O(n)$, where z is the number of reported pairs. In the following we assume, unless stated otherwise, that v is a node in the binary suffix tree $T_B(S)$ with path-label α and children w_1 and w_2 named such that $|LL(w_1)| \leq |LL(w_2)|$. We say that w_1 is the small child of v and that w_2 is the big child of v.

Right-maximal pairs with upper and lower bounded gap

To find all right-maximal pairs in S with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$ we consider every node v in the binary suffix tree $T_B(S)$ in a bottom-up fashion, e.g. during a depth-first traversal. At every node v we use AVL trees storing the leaf-lists $LL(w_1)$ and $LL(w_2)$ at its two children to report the proper right-maximal pairs of its path-label α. The details are given in Algorithm 2 and explained below.

At every node v in $T_B(S)$ we construct an AVL tree, the leaf-list tree T, that stores the elements in $LL(v)$. If v is a leaf then we construct T directly in Step 1. If v is an internal node then $LL(v)$ is the union of the disjoint leaf-lists $LL(w_1)$ and $LL(w_2)$ which by assumption are stored in the already constructed T_1 and T_2, so we construct T by merging T_1 and T_2, $|T_1| \leq |T_2|$, using Lemma 4. Before constructing T in Step 2c we use T_1 and T_2 to report right-maximal pairs from node v by reporting every p in $LL(w_1)$ against every q in $LL(w_2)\cap L(p,|\alpha|)$ and $LL(w_2)\cap R(p,|\alpha|)$. This is done in two steps. In Step 2a we find for every p in $LL(w_1)$ the minimum element $q_r(p)$ in $LL(w_2)\cap R(p,|\alpha|)$ and the minimum element $q_l(p)$ in $LL(w_2)\cap L(p,|\alpha|)$ by searching in T_2 using Lemma 5. In Step 2b we report pairs $(p,q,|\alpha|)$ and $(q,p,|\alpha|)$ for every p in $LL(w_1)$ and increasing q's in $LL(w_2)$ starting with $q_r(p)$ and $q_l(p)$ respectively, until the gap violates the upper or lower bound.

To argue that Algorithm 2 finds all right-maximal pairs with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$ it is enough to argue that we for every p in $LL(w_1)$ report all right-maximal pairs $(p,q,|\alpha|)$ and $(q,p,|\alpha|)$ with gap between $g_1(|\alpha|)$

and $g_2(|\alpha|)$. The rest follows because we at every node v in $T_B(S)$ consider every p in $LL(w_1)$. Consider the call $\mathsf{Report}(q_r(p), p + |\alpha| + g_2(|\alpha|))$ in Step 2b. From the implementation of Report follows that this call reports p against every q in $LL(w_2) \cap [q_r(p) \mathinner{..} p + |\alpha| + g_2(|\alpha|)]$. By construction of $q_r(p)$ and definition of $R(p, |\alpha|)$ follows that $LL(w_2) \cap [q_r(p) \mathinner{..} p + |\alpha| + g_2(|\alpha|)]$ is equal to $LL(w_2) \cap R(p, |\alpha|)$, so the call reports all pairs $(p, q, |\alpha|)$ with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$. Similarly we can argue that the call $\mathsf{Report}(q_l(p), p - |\alpha| - g_1(|\alpha|))$ reports all pairs $(q, p, |\alpha|)$ with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$.

Now consider the running time of Algorithm 2. Building the binary suffix tree $T_B(S)$ and creating an AVL tree of size one at each leaf in Step 1 takes time $O(n)$. At every internal node in $T_B(S)$ we do Step 2. Since $|T_1| \leq |T_2|$ searching in Step 2a and merging in Step 2c takes time $O(\log \binom{|T_1|+|T_2|}{|T_1|})$ by Lemmas 5 and 4 respectively. Reporting of pairs in Step 2b takes time proportional to $|T_1|$, because we consider every p in $LL(w_1)$, plus the number of reported pairs. Summing this over all nodes gives by Lemma 6 that the total running time is $O(n \log n + z)$, where z is the number of reported pairs. Since constructing and keeping $T_B(S)$ requires space $O(n)$, and since no element at any time is in more than one leaf-list tree, Algorithm 2 requires space $O(n)$.

Theorem 1 *Algorithm 2 finds all right-maximal pairs $(i, j, |\alpha|)$ in a string S with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$ in space $O(n)$ and time $O(n \log n + z)$, where z is the number of reported pairs and n is the length of S.*

Maximal pairs with upper and lower bounded gap

We now turn towards finding all maximal pairs in S with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$. Our approach to find all maximal pairs in S with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$ is to extend Algorithm 2 to filter out all right-maximal pairs that are not left-maximal. A simple solution is to extend the procedure Report to check if $S[p-1] \neq S[q-1]$ before reporting the pair $(p, q, |\alpha|)$ or $(q, p, |\alpha|)$ in Step 2b. This solution takes time proportional to the number of inspected right-maximal pairs, and not time proportional to the number of reported maximal pairs. Even though the maximum number of right-maximal pairs and maximal pairs in strings of a given length are asymptotically equal, many strings contain significantly fewer maximal pairs than right-maximal pairs. We therefore want to filter out all right-maximal pairs that are not left-maximal *without* inspecting all right-maximal pairs. In the remainder of this section we describe one way to do this.

Consider the reporting step in Algorithm 2 and assume that we are about to report from a node v with children w_1 and w_2. The leaf-list trees T_1 and T_2, $|T_1| \leq |T_2|$, are available and they make it possible to access the elements in $LL(w_1) = \{p_1, p_2, \ldots, p_s\}$ and $LL(w_2) = \{q_1, q_2, \ldots, q_t\}$ in sorted order. We divide the sorted leaf-list $LL(w_2)$ into blocks of contiguous elements such that the elements q_{i-1} and q_i are in the same block if and only if $S[q_{i-1}-1] = S[q_i-1]$. We say that we divide the sorted leaf-list into blocks of elements with equal left-characters. To filter out all right-maximal pairs that are not left-maximal we must avoid to report p in $LL(w_1)$ against any element q in $LL(w_2)$ in a block of

Algorithm 2 Find all right-maximal pairs in string S with bounded gap.

1. *Initializing:* Build the binary suffix tree $T_B(S)$ and create at each leaf an AVL tree of size one that stores the index at the leaf.

2. *Reporting and merging:* When the AVL trees T_1 and T_2, $|T_1| \leq |T_2|$, at the two children w_1 and w_2 of node v with path-label α are available, we do the following:

   (a) Let $\{p_1, p_2, \ldots, p_s\}$ be the elements in T_1 in sorted order. For each element p in T_1 we find

   $$\begin{aligned} q_r(p) &= \min\{x \in T_2 \mid x \geq p + |\alpha| + g_1(|\alpha|)\} \\ q_l(p) &= \min\{x \in T_2 \mid x \geq p - |\alpha| - g_2(|\alpha|)\} \end{aligned}$$

   by searching in T_2 with the sorted lists $\{p_i + |\alpha| + g_1(|\alpha|) \mid i = 1, 2, \ldots, s\}$ and $\{p_i - |\alpha| - g_2(|\alpha|) \mid i = 1, 2, \ldots, s\}$ using Lemma 5.

   (b) For each element p in T_1 we do $\mathsf{Report}(q_r(p), p + |\alpha| + g_2(|\alpha|))$ and $\mathsf{Report}(q_l(p), p - |\alpha| - g_1(|\alpha|))$ where Report is the following procedure.

   ```
   def Report(from, to) :
       q = from
       while q ≤ to :
           report pair (p, q, |α|) if p < q, and (q, p, |α|) otherwise
           q = next(q)
   ```

   (c) Build the leaf-list tree T at node v by merging T_1 and T_2 using Lemma 4.

elements with left-character $S[p-1]$. This gives the overall idea of the extended algorithm; we extend the reporting step in Algorithm 2 such that whenever we are about to report p in $LL(w_1)$ against q in $LL(w_2)$ where $S[p-1] = S[q-1]$ we skip all elements in the current block containing q and continue reporting p against the first element q' in the following block, which by the definition of blocks satisfies that $S[p-1] \neq S[q'-1]$.

To implement this extended reporting step efficiently we must be able to skip all elements in a block without inspecting each of them. We achieve this by constructing an additional AVL tree, the block-start tree, that keeps track of the blocks in the leaf-list. At each node v during the traversal of $T_B(S)$ we thus construct two AVL trees; the leaf-list tree T that stores the elements in $LL(v)$, and the block-start tree B that keeps track of the blocks in the sorted leaf-list by storing all the elements in $LL(v)$ that start a block. We keep links from the block-start tree to the leaf-list tree such that we in constant time can go from an element in the block-start tree to the corresponding element in the leaf-list tree. Figure 4.3 illustrates the leaf-list tree, the block-start tree and the links between them. Before we present the extended algorithm and explain

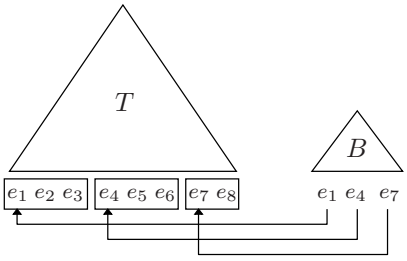


Figure 4.3: The data structure constructed at each node v in $T_B(S)$. The leaf-list tree T stores all elements in $LL(v)$. The block-start tree B stores all elements in $LL(v)$ that start a block in the sorted leaf-list. We keep links from the elements in the block-start tree to the corresponding elements in the leaf-list tree.

how to use the block-start tree to efficiently skip all elements in a block, we first describe how to construct the leaf-list tree T and block-start tree B at node v from the leaf-list trees, T_1 and T_2, and block-start trees, B_1 and B_2, at its two children w_1 and w_2.

Since $LL(v)$ is the union of the disjoint leaf-lists $LL(w_1)$ and $LL(w_2)$ stored in T_1 and T_2 respectively, we can construct the leaf-list tree T by merging T_1 and T_2 using Lemma 4. It is more involved to construct the block-start tree B. The reason is that an element p_i that starts a block in $LL(w_1)$ or an element q_j that starts a block in $LL(w_2)$ does not necessarily start a block in $LL(v)$ and vice versa, so we cannot construct B by merging B_1 and B_2. Let $\{e_1, e_2, \ldots, e_{s+t}\}$ be the elements in $LL(v)$ in sorted order. By definition the block-start tree B contains all elements e_k in $LL(v)$ where $S[e_{k-1} - 1] \neq S[e_k - 1]$. We construct B by modifying B_2. We choose to modify B_2, and not B_1, because $|LL(w_1)| \leq |LL(w_2)|$, which by the "smaller-half trick" allows us to consider all elements in $LL(w_1)$ without spending too much time in total. To modify B_2 to become B we must identify all the elements that are in B but not in B_2 and vice versa.

Lemma 7 *If e_k is in B but not in B_2 then $e_k \in LL(w_1)$ or $e_{k-1} \in LL(w_1)$.*

Proof. Assume that e_k is in B and that e_k and e_{k-1} both are in $LL(w_2)$. In $LL(w_2)$ the elements e_k and e_{k-1} are neighboring elements q_j and q_{j-1}. Since e_k starts a block in $LL(v)$ then $S[q_j - 1] = S[e_k - 1] \neq S[e_{k-1} - 1] = S[q_{j-1} - 1]$. This shows that $q_j = e_k$ is in B_2 and the lemma follows. □

Let *NEW* be the set of elements e_k in B where e_k or e_{k-1} are in $LL(w_1)$. It follows from Lemma 7 that this set contains at least all elements in B that are not in B_2. It is easy to see that we can construct *NEW* in sorted order while merging T_1 and T_2; whenever an element e_k from T_1, i.e. $LL(w_1)$, is placed in T, i.e. $LL(v)$, we include it, and/or the next element e_{k+1} placed in T, in *NEW* if they start a block in $LL(v)$.

If we insert the elements in *NEW* we are halfway done modifying B_2 to

become B. We still need to identify and remove the elements that should be removed from B_2, that is, the elements that are in B_2 but not in B.

Lemma 8 *An element q_j in B_2 is not in B if and only if the largest element e_k in NEW smaller than q_j in B_2 has the same left-character as q_j.*

Proof. If q_j is in B_2 but does not start a block in $LL(v)$, then it must be in a block started by some element e_k with the same left-character as q_j. This block cannot contain q_{j-1} because q_j being in B_2 implies that $S[q_j - 1] \neq S[q_{j-1} - 1]$. We thus have the ordering $q_{j-1} < e_k < q_j$. This implies that e_k is the largest element in NEW smaller than q_j. If e_k is the largest element in NEW smaller than q_j, then no block starts in $LL(v)$ between e_k and q_j, i.e. all elements e in $LL(v)$ where $e_k < e < q_j$ satisfy that $S[e-1] = S[e_k-1]$, so if $S[e_k-1] = S[q_j-1]$ then q_j does not start a block in $LL(v)$. □

By searching in B_2 with the sorted list NEW using Lemma 5 it is straightforward to find all pairs of elements (e_k, q_j), where e_k is the largest element in NEW smaller than q_j in B_2. If the left-characters of e_k and q_j in such a pair are equal, i.e. $S[e_k - 1] = S[q_j - 1]$, then by Lemma 8 the element q_j is not in B and must therefore be removed from B_2. It follows from the proof of Lemma 8 that if this is the case then $q_{j-1} < e_k < q_j$, so we can, without destroying the order among the nodes in B_2, remove q_j from B_2 and insert e_k instead, simply by replacing the element q_j with the element e_k at the node storing q_j in B_2.

We can now summarize the three steps it takes to modify B_2 to become B. In Step 1 we construct the sorted set NEW that contains all elements in B that are not in B_2. This is done while merging T_1 and T_2 using Lemma 4. In Step 2 we remove the elements from B_2 that are not in B. The elements in B_2 being removed and the elements from NEW replacing them are identified using Lemmas 5 and 8. In Step 3 we merge the remaining elements in NEW into the modified B_2 using Lemma 4. Adding links from the new elements in B to the corresponding elements in T can be done while replacing and merging in Steps 2 and 3. Since $|NEW| \leq 2\,|T_1|$ and $|B_2| \leq |T_2|$, the time it takes to construct B is dominated by the the time it takes merge a sorted list of size $2\,|T_1|$ into an AVL tree of size $|T_2|$. By Lemma 4 this is within a constant factor of the time it takes to merge T_1 and T_2, so the time is takes to construct B is dominated by the time it takes to construct the leaf-list tree T.

Now that we know how to construct the leaf-list tree T and block-start tree B at node v from the leaf-list trees, T_1 and T_2, and block-start trees, B_1 and B_2, at its two children w_1 and w_2, we can proceed with the implementation of the extended reporting step. The details are shown in Algorithm 3. This algorithm is similar to Algorithm 2 except that we at every node v in $T_B(S)$ construct two AVL trees; the leaf-list tree T that stores the elements in $LL(v)$, and the block-start tree B that keeps track of the blocks in $LL(v)$ by storing the subset of elements that start a block. If v is a leaf, we construct T and B directly. If v is an internal node, we construct T by merging the leaf-list trees T_1 and T_2 at its two children w_1 and w_2, and we construct B by modifying the block-start tree B_2 as explained above.

Algorithm 3 Find all maximal pairs in string S with bounded gap.

1. *Initializing:* Build the binary suffix tree $T_B(S)$ and create at each leaf two AVL trees of size one, the leaf-list and the block-start tree, both storing the index at the leaf.

2. *Reporting and merging:* When the leaf-list trees T_1 and T_2, $|T_1| \leq |T_2|$, and the block-start trees B_1 and B_2 at the two children w_1 and w_2 of node v with path-label α are available, we do the following:

   (a) Let $\{p_1, p_2, \ldots, p_s\}$ be the elements in T_1 in sorted order. For each element p in T_1 we find

   $$\begin{aligned} q_r(p) &= \min\{x \in T_2 \mid x \geq p + |\alpha| + g_1(|\alpha|)\} \\ q_l(p) &= \min\{x \in T_2 \mid x \geq p - |\alpha| - g_2(|\alpha|)\} \\ b_r(p) &= \min\{x \in B_2 \mid x \geq p + |\alpha| + g_1(|\alpha|)\} \\ b_l(p) &= \min\{x \in B_2 \mid x \geq p - |\alpha| - g_2(|\alpha|)\} \end{aligned}$$

   by searching in T_2 and B_2 with the sorted lists $\{p_i + |\alpha| + g_1(|\alpha|) \mid i = 1, 2, \ldots, s\}$ and $\{p_i - |\alpha| - g_2(|\alpha|) \mid i = 1, 2, \ldots, s\}$ using Lemma 5.

   (b) For each element p in T_1 we do $\mathsf{ReportMax}(q_r(p), b_r(p), p + |\alpha| + g_2(|\alpha|))$ and $\mathsf{ReportMax}(q_l(p), b_l(p), p - |\alpha| - g_1(|\alpha|))$ where $\mathsf{ReportMax}$ is the following procedure.

```
def ReportMax(from_T, from_B, to):
    q = from_T
    b = from_B
    while q ≤ to:
        if S[q − 1] ≠ S[p − 1]:
            report pair (p, q, |α|) if p < q, and (q, p, |α|) otherwise
            q = next(q)
        else:
            while b ≤ q:
                b = next(b)
            q = b
```

   (c) Build the leaf-list tree T at node v by merging T_1 and T_2 using Lemma 4. Build the block-start tree B at node v by modifying B_2 as described in the text.

Before constructing T and B we report all maximal pairs from node v with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$ by reporting every p in $LL(w_1)$ against every q in $LL(w_2) \cap L(p, |\alpha|)$ and $LL(w_2) \cap R(p, |\alpha|)$ where $S[p-1] \neq S[q-1]$. This is done in two steps. In Step 2a we find for every p in $LL(w_1)$ the minimum elements $q_l(p)$ and $q_r(p)$, as well as the minimum elements $b_l(p)$ and $b_r(p)$ that start

a block, in $LL(w_2) \cap L(p, |\alpha|)$ and $LL(w_2) \cap R(p, |\alpha|)$ respectively. This is done by searching in T_2 and B_2 using Lemma 5. In Step 2b we report pairs $(p, q, |\alpha|)$ and $(q, p, |\alpha|)$ for every p in $LL(w_1)$ and increasing q's in $LL(w_2)$ starting with $q_r(p)$ and $q_l(p)$ respectively, until the gap violates the upper or lower bound. Whenever we are about to report p against q where $S[p-1] = S[q-1]$, we instead use the block-start tree B_2 to skip all elements in the block containing q and continue with reporting p against the first element in the following block.

To argue that Algorithm 3 finds all maximal pairs with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$ it is enough to argue that we for every p in $LL(w_1)$ report all maximal pairs $(p, q, |\alpha|)$ and $(q, p, |\alpha|)$ with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$. The rest follows because we at every node in $T_B(S)$ consider every p in $LL(w_1)$. Consider the call $\mathsf{ReportMax}(q_r(p), b_r(p), p + |\alpha| + g_2(|\alpha|))$ in Step 2b. From the implementation of $\mathsf{ReportMax}$ follows that unless we skip elements by increasing b then we consider every q in $LL(w_2) \cap R(p, |\alpha|)$. The test $S[q-1] \neq S[p-1]$ before reporting a pair ensures that we only report maximal pairs and whenever $S[q-1] = S[p-1]$ we increase b until $b = \min\{x \in B_2 \mid x > q\}$. This is, by construction of B_2 and $b_r(p)$, the element that starts the block following the block containing q, so all elements q', $q < q' < b$, we skip by setting q to b satisfy that $S[p-1] = S[q-1] = S[q'-1]$. We thus conclude that $\mathsf{ReportMax}(q_r(p), b_r(p), p + |\alpha| + g_2(|\alpha|))$ reports p against exactly those q in $LL(w_2) \cap R(p, |\alpha|)$ where $S[p-1] \neq S[q-1]$, i.e. it reports all maximal pairs $(p, q, |\alpha|)$ at node v with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$. Similarly, the call $\mathsf{ReportMax}(q_l(p), b_l(p), p - |\alpha| - g_1(|\alpha|))$ reports all maximal pairs $(q, p, |\alpha|)$ with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$.

Now consider the running time of Algorithm 3. We first argue that the call $\mathsf{ReportMax}(q_r(p), b_r(p), p + |\alpha| + g_2(|\alpha|))$ takes constant time plus time proportional to the number of reported pairs $(p, q, |\alpha|)$. To do this all we have to show is that the time used to skip blocks, i.e. the number of times we increase b, is proportional to the number of reported pairs. By construction $b_r(p) \geq q_r(p)$, so the number of times we increase b is bounded by the number of blocks in $LL(w_2) \cap R(p, |\alpha|)$. Since neighboring blocks contain elements with different left-characters, we report p against an element from at least every second block in $LL(w_2) \cap R(p, |\alpha|)$. The number of times we increase b is thus proportional to the number of reported pairs. The call $\mathsf{ReportMax}(q_l(p), b_l(p), p - |\alpha| - g_1(|\alpha|))$ also takes constant time plus time proportional to the number of reported pairs $(q, p, |\alpha|)$. We thus have that Step 2b takes time proportional to $|T_1|$ plus the number of reported pairs. Everything else we do at node v, i.e. searching in T_2 and B_2 and constructing the leaf-list tree T and block-start tree B, takes time $O(\log \binom{|T_1|+|T_2|}{|T_1|})$. Summing this over all nodes gives by Lemma 6 that the total running time of the algorithm is $O(n \log n + z)$ where z is the number of reported pairs. Since constructing and keeping $T_B(S)$ requires space $O(n)$, and since no element at any time is in more than one leaf-list tree, and maybe one block-start tree, Algorithm 3 requires space $O(n)$.

Theorem 2 *Algorithm 3 finds all maximal pairs $(i, j, |\alpha|)$ in a string S with gap between $g_1(|\alpha|)$ and $g_2(|\alpha|)$ in space $O(n)$ and time $O(n \log n + z)$, where z is the number of reported pairs and n is the length of S.*

We observe that Algorithm 3 never uses the block-start tree B_1 at the small child w_1. This observation can be used to ensure that only one block-start tree exists during the execution of the algorithm. If we implement the traversal of $T_B(S)$ as a depth-first traversal in which we at each node v first recursively traverse the subtree rooted at the small child w_1, then we do not need to store the block-start tree returned by this recursive traversal while recursively traversing the subtree rooted at the big child w_2. This implies that only one block-start tree exists at all times during the recursive traversal of $T_B(S)$. The drawback is that we at each node v need to know in advance which child is the small child, but this knowledge can be obtained in linear time by annotating each node with the size of the subtree it roots.

4.4 Pairs with lower bounded gap

If we relax the constraint on the gap and only want to find all maximal pairs in S with gap at least $g(|\alpha|)$, where g is a function that can be computed in constant time, then a straightforward solution is to use Algorithm 3 with $g_1(|\alpha|) = g(|\alpha|)$ and $g_2(|\alpha|) = n$. This obviously finds all maximal pairs with gap at least $g_1(|\alpha|) = g(|\alpha|)$ in time $O(n \log n + z)$. However, the missing upper bound on the gap, i.e. the trivial upper bound $g_2(|\alpha|) = n$, makes it possible to reduce the running time to $O(n + z)$ since reporting from each node during the traversal of the binary suffix tree is simplified.

The reporting of pairs from node v with children w_1 and w_2 is simplified, because the lack of an upper bound on the gap implies that we do not have to search $LL(w_2)$ for the first element to report against the current element in $LL(w_1)$. Instead we can start by reporting the current element in $LL(w_1)$ against the biggest (and smallest) element in $LL(w_2)$ and then continue reporting it against decreasing (and increasing) elements from $LL(w_2)$ until the gap becomes smaller than $g(|\alpha|)$. Unfortunately this simplification alone does not reduce the asymptotic running time because inspecting every element in $LL(w_1)$ and keeping track of the leaf-lists in AVL trees alone requires time $\Theta(n \log n)$. To reduce the running time we must thus avoid to inspect every element in $LL(w_1)$ and find another way to store the leaf-lists. We achieve this by using the data structures presented below to store the leaf-lists during the traversal of the binary suffix tree.

4.4.1 Data structures

A heap-ordered tree is a tree in which each node stores an element and has a key. Every node other than the root satisfies that its key is greater than or equal to the key at its parent. Heap-ordered trees have been widely studied and are the basic structure of many priority queues [156, 42, 150, 44]. In this section we utilize heap-ordered trees to construct two data structures, *the heap-tree* and *the colored heap-tree*, that are useful in our application of finding pairs with lower bounded gap but might also have applications elsewhere.

A heap-tree stores a collection of elements with comparable keys and supports the following operations.

$\mathsf{Init}(e, k)$:	Return a heap-tree of size one that stores element e with key k.
$\mathsf{Find}(H, x)$:	Return all elements e stored in the heap-tree H with key $k \leq x$.
$\mathsf{Min}(H)$:	Return the element e stored in H with minimum key.
$\mathsf{Meld}(H, H')$:	Return a heap-tree that stores all elements in H and H' with unchanged keys.

A colored heap-tree stores a collection of colored elements with comparable keys. We use $color(e)$ to denote the color of element e. A colored heap-tree supports the same operations as a heap-tree except that it allows us to find all elements not having a particular color. The operations are as follows.

$\mathsf{ColorInit}(e, k)$:	Return a colored heap-tree of size one that stores element e with key k.
$\mathsf{ColorFind}(H, x, c)$:	Return all elements e stored in the colored heap-tree H with key $k \leq x$ and $color(e) \neq c$.
$\mathsf{ColorMin}(H)$:	Return the element e stored in H with minimum key.
$\mathsf{ColorSec}(H)$:	Return the element e stored in H with minimum key such that $color(e) \neq color(\mathsf{ColorMin}(H))$.
$\mathsf{ColorMeld}(H, H')$:	Return a colored heap-tree that stores all elements in H and H' with unchanged keys.

In the following we will describe how to implement heap-trees and colored heap-trees using heap-ordered trees such that Init, Min, ColorInit, ColorMin and ColorSec take constant time, Find and ColorFind take time proportional to the number of returned elements, and Meld and ColorMeld take amortized constant time. This means that we can meld n (colored) heap-trees of size one into a single (colored) heap-tree of size n by an arbitrary sequence of $n - 1$ meld operations in time $O(n)$ in the worst case.

Heap-trees

We implement heap-trees as binary heap-ordered trees as illustrated in Figure 4.4. At every node in the heap-ordered tree we store an element from the collection of elements we want to store. The key of a node is the key of the element it stores. We use $v.elm$ to refer to the element stored at node v, $v.key$ to refer to the key of node v, and $v.right$ and $v.left$ to refer to the two children of node v. Besides the heap-order we maintain the invariant that the root of the heap-ordered tree has no left-child.

We define the *backbone* of a heap-tree as the path in the heap-ordered tree that starts at the root and continues via nodes reachable from the root via a sequence of right-children. We define the length of the backbone as the number of edges on the path it describes. Consider the heap-trees H and H' in Figure 4.4; the backbone of H is the path $r, v_1, \ldots, v_s$ of length s and the

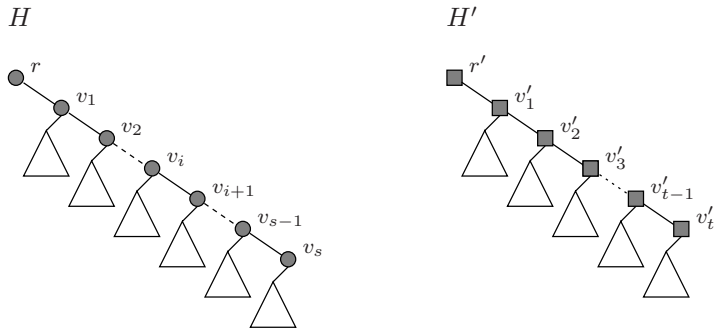


Figure 4.4: The implementation of heap-trees as binary heap-ordered trees. The figure shows two heap-trees H and H'. The nodes on the backbone of the two heap-trees are shaded.

backbone of H' is the path $r', v'_1, \ldots, v'_t$ of length t. We say that the node on the backbone farthest from the root is at the bottom of the backbone. We keep track of the nodes on the backbone of a heap-tree using a stack, *the backbone-stack*, in which the root is at the bottom and the node farthest from the root is at the top. The backbone-stack makes it easy to access the nodes on the backbone from the bottom and up towards the root.

We now turn to the implementation of Init, Min, Find and Meld. $\mathsf{Init}(e, k)$ is straightforward. We construct a single node v where $v.elm = e$, $v.key = k$ and $v.right = v.left = null$ and a backbone-stack of size one that contains node v. $\mathsf{Min}(H)$ is also straightforward. The heap-order implies that root r of H stores the element with minimum key, i.e. $\mathsf{Min}(H) = r.elm$.

We implement $\mathsf{Find}(H, x)$ as a recursive traversal of H starting at the root. At each node v we compare $v.key$ to x. If $v.key \leq x$, we report $v.elm$ and continue recursively with the two children of v. If $v.key > x$, then by the heap-order all keys at nodes in the subtree rooted at v are greater than x, so we return from v without reporting. Clearly this traversal reports all elements stored at nodes v with $v.key \leq x$, i.e. all elements stored with key $k \leq x$. Since each node has at most two children, we make, for each reported element, at most two additional comparisons against x corresponding to the at most two recursive calls from which we return without reporting. The running time of the traversal is thus proportional to the number of reported elements.

We implement $\mathsf{Meld}(H, H')$ in two steps. Figure 4.5 illustrates the melding of the heap-trees H and H' from Figure 4.4. We assume that $r.key \leq r'.key$. In Step 1 we merge the backbones of H and H' together such that the heap-order is satisfied in the resulting tree. The merged backbone is constructed from the bottom and up towards the root by popping nodes from the backbone-stacks of H and H'. Step 1 results in a heap-tree with a backbone of length $s + t + 1$. Since $r.key \leq r'.key$, a prefix of the merged backbone consists of nodes $r, v_1, v_2, \ldots, v_i$ solely from the backbone of H. In Step 2 we shorten the merged backbone. Since the root r' of H' has no left-child, the node r' on the merged backbone has no left-child either, so by moving the right-child of r' to this empty spot,

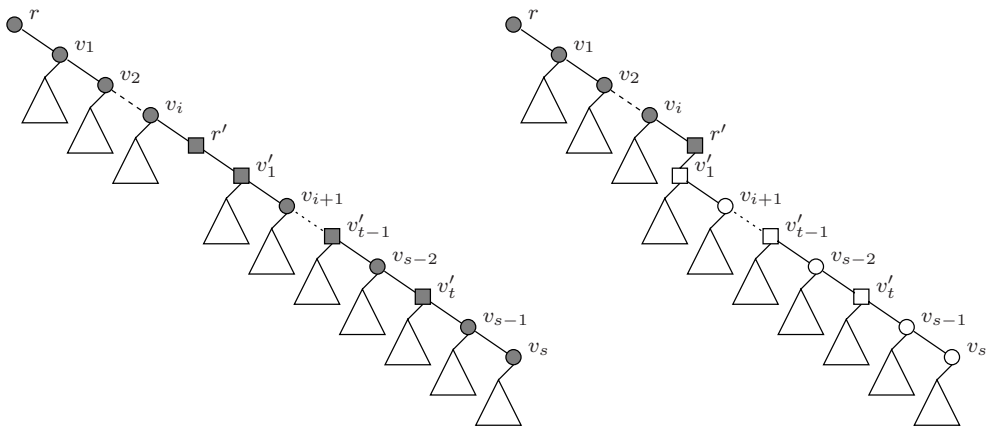


Figure 4.5: The two steps of melding the heap-trees H and H' shown in Figure 4.4. The heap-tree to the left is the result of merging the backbones. The heap-tree to the right is the result of shortening the backbone by moving the right-child of r' in the merged backbone to the left-child. The nodes on the backbone of the two heap-trees are marked.

making it the left-child of r', we shorten the length of the merged backbone to $i+1$.

The two steps of $\mathsf{Meld}(H, H')$ clearly construct a heap-ordered tree that stores all elements in H and H' with unchanged keys. Since $r.key \leq r'.key$, the root of the constructed heap-ordered tree is the root of H and therefore has no left-child. The constructed heap-ordered tree is thus a heap-tree as wanted. The backbone of the new heap-tree is the path $r, v_1, \ldots, v_i, r'$. We observe that the backbone-stack of H after Step 1 contains exactly the nodes $r, v_1, \ldots v_i$. We can thus construct the backbone-stack of the new heap-tree by pushing r' onto what remains of the backbone-stack of H after Step 1.

Now consider the running time of $\mathsf{Meld}(H, H')$. Step 1 takes time proportional to the total number of nodes popped from the two backbone-stacks. Since $i+1$ nodes remains on the backbone-stack of H, Step 1 takes time $(s+1)+(t+1)-(i+1) = s+t-i+1$. Step 2 and construction of the new backbone-stack takes constant time, so, except for a constant factor, melding two heap-trees with backbones of length s and t takes time $T(s,t) = s+t-i+1$. In our application of finding pairs we are more interested in bounding the total time required to do a sequence of melds rather than bounding the time of each individual meld. We therefore turn to amortized analysis [137].

On a forest F of heap-trees we define the potential function $\Phi(F)$ to be the sum of the lengths of the backbones of the heap-trees in the forest. Melding two heap-trees with backbones of length s and t, as illustrated in Figure 4.5, changes the potential of the forest with $\Delta\Phi = i+1-(s+t)$. The amortized running time of melding the two heap-trees is $T(s,t)+\Delta\Phi = (s+t-i+1)+(i-s-t+1) = 2$, so starting with n heap-trees of size one, i.e. a forest F_0 with potential $\Phi(F_0) = 0$, and doing a sequence of $n-1$ meld operations until the forest F_{n-1} consists of a single heap-tree, takes time $O(n)$ in the worst case.

Colored heap-trees

We implement colored heap-trees as colored heap-ordered trees in much the same way as we implemented heap-trees as uncolored heap-ordered trees. The implementation only differs in two ways. First, a node in the colored heap-ordered tree stores a set of elements instead of just a single element. Secondly, a node, including the root, can have several left-children. The elements stored at a node, and the references to the left-children of a node, are kept in uncolored heap-trees. More precisely, a node v in the colored heap-ordered tree has the following attributes.

$v.elms$: A heap-tree that stores the elements at node v. $\mathsf{Find}(v.elms, x)$ returns all elements stored at node v with key less than or equal to x. All elements stored at node v have identical colors. We say that this color is the color of node v and denote it by $color(v)$.

$v.key$: The key of node v. We set the key of a node to be the minimum key of an element stored at the node, i.e. the key of node v is the key of the element stored at the root of $v.elms$.

$v.right$: A reference to the right-child of node v.

$v.lefts$: A heap-tree that stores the references to the left-children of node v. A reference is stored with a key equal to the key of the referenced left-child, so $\mathsf{Find}(v.lefts, x)$ returns the references to all left-children of node v with key less than or equal to x.

As for a heap-tree we define the backbone of a colored heap-tree as the path that starts at the root and continues via nodes reachable from the root via a sequence of right-children. We use a stack, the backbone-stack, to keep track of the nodes on the backbone. In addition to the heap-order, saying that the key of every node other than the root is greater than or equal to the key of its parent, we maintain the following three invariants about the color of the nodes and the relation between the elements stored at a node and its left-children.

I_1: Every node v other than the root r has a color different from its parent.

I_2: Every node v satisfies that $|\mathsf{Find}(v.elms, x)| \geq |\mathsf{Find}(v.lefts, x)|$ for any x.

I_3: The root r satisfies that $|\mathsf{Find}(r.elms, x)| \geq |\mathsf{Find}(r.lefts, x)| + 1$ for any $x \geq \mathsf{Min}(r.elms)$.

We can now turn to the implementation of the operations on colored heap-trees. $\mathsf{ColorInit}(e, k)$ is straightforward. We simply construct a single node v where $v.key = k$, $v.elms = \mathsf{Init}(e, k)$ and $v.right = v.lefts = null$ and a backbone-stack that contains node v. $\mathsf{ColorMin}(H)$ is also straightforward. The heap-order implies that the element with minimum key is stored in the heap-tree $r.elms$ at the root r of H, so $\mathsf{ColorMin}(H) = \mathsf{Min}(r.elms)$. The heap-order and I_1 imply that $\mathsf{ColorSec}(H)$ is the element stored with minimum key at a child of r. The element stored with minimum key at the right-child is

$\mathsf{Min}(r.\mathit{right})$ and the element stored with minimum key at a left-child must by the heap-order of $r.\mathit{lefts}$ be the element stored with minimum key at the left-child referenced by the root of $r.\mathit{lefts}$, i.e. $\mathsf{Min}(\mathsf{Root}(r.\mathit{lefts}).\mathit{elm})$. Both $\mathsf{ColorMin}(H)$ and $\mathsf{ColorSec}(H)$ can thus be found in constant time.

We implement $\mathsf{ColorFind}(H, x, c)$ as a recursive traversal of H starting at the root. More precisely, we implement $\mathsf{ColorFind}(H, x, c)$ as $\mathsf{ReportFrom}(r)$ where r is the root of H and $\mathsf{ReportFrom}$ is the following recursive procedure.

```
def ReportFrom(v):
    if key(v) ≤ x:
        if color(v) ≠ c:
            E = Find(v.elms, x)
            for e in E:
                report e
        ReportFrom(v.right)
        W = Find(v.lefts, x)
        for w in W:
            ReportFrom(w)
```

The correctness of this implementation is easy to establish. The heap-order ensures that all nodes v with $v.\mathit{key} \leq x$ are visited during the traversal. The definition of $v.\mathit{key}$ implies that any element e with key $k \leq x$ is stored at a node v with $v.\mathit{key} \leq x$, i.e. among the elements returned by $\mathsf{Find}(v.\mathit{elms}, x)$ for some node v visited during the traversal. Together with the test $\mathit{color}(v) \neq c$ this implies that all elements e with key $k \leq x$ and color different from c are reported by $\mathsf{ColorFind}(H, x, c)$.

Now consider the running time of $\mathsf{ColorFind}(H, x, c)$. Since $\mathsf{Find}(v.\mathit{elms}, x)$ and $\mathsf{Find}(v.\mathit{lefts}, x)$ both take time proportional to the number of returned elements, it follows that the running time is dominated by the number of recursive calls plus the number of reported elements. To argue that the running time of $\mathsf{ColorFind}(H, x, c)$ is proportional to the number of reported elements we therefore argue that the number of reported elements dominates the number of recursive calls. We only make recursive calls from a node v if $v.\mathit{key} \leq x$. Let v be such a node and consider two cases. If $\mathit{color}(v) \neq c$, then we report at least one element, namely the element with key $v.\mathit{key}$, and by I_2 and I_3 we report at least as many elements as the number of left-children we call from v, so except for a constant term that we can charge for visiting node v, the number of reported elements at v accounts for the call to v and all calls from v. If $\mathit{color}(v) = c$, then we do not report any elements at v, but I_1 ensures that we reported elements at its parent (unless v is the root) and that we will be reporting elements at all left-children we call from v. The call to v is thus already accounted for by the elements reported at its parent, and except for a constant term that we can charge for visiting node v, all calls from v will be accounted for by elements reported at the children of v. We conclude that the number of reported elements dominates the number of recursive calls, so $\mathsf{ColorFind}(H, x, c)$ takes time proportional to the number of reported elements.

We implement $\mathsf{ColorMeld}(H, H')$ similar to $\mathsf{Meld}(H, H')$ except that we must ensure that the constructed colored heap-tree obeys the three invariants. Let H and H' be colored heap-trees with roots r and r', $r.key \leq r'.key$, respectively. We implement $\mathsf{ColorMeld}(H, H')$ as the following three steps.

1. *Merge.* We merge the backbones of H and H' together such that the resulting heap-ordered tree stores all elements in H and H' with unchanged keys. The merging is done by popping nodes from the backbone-stacks of H and H' until the backbone-stack of H' is empty

2. *Solve conflicts.* A node w on the merged backbone with the same color as its parent v is a violation of invariant I_1. We solve conflicts between neighboring nodes v and w of equal color by melding the elements and left-children of the two nodes and removing node w. We say that parent v swallows the child w.

   $v.elms = \mathsf{Meld}(v.elms, w.elms)$
   $v.lefts = \mathsf{Meld}(v.lefts, w.lefts)$
   $v.right = w.right$

3. *Shorten backbone.* Let v be the node on the merged backbone corresponding to r' or the node that swallowed r' in Step 2. We shorten the backbone by moving the right-child of v to the set of left-children of v.

   $v.lefts = \mathsf{Meld}(v.lefts, \mathsf{Init}(v.right, v.right.key))$
   $v.right = null$

The main difference from the implementation of $\mathsf{Meld}(H, H')$ is Step 2 where the invariant I_1 is restored along the merged backbone. To establish the correctness of the implementation of $\mathsf{ColorMeld}(H, H')$ we consider each of the three steps in more details.

In Step 1 we merge the backbones of H and H' together such that the resulting tree is a heap-ordered tree that stores all elements in H and H' with unchanged keys. Since the merging does not change the left-children or the elements of any node and since H and H' both obey I_2 and I_3, the constructed heap-ordered tree also obeys I_2 and I_3. The merged backbone can however contain neighboring nodes of equal color. These conflicts are a violation of I_1.

In Step 2 we restore I_1. We solve all conflicts on the merged backbone between neighboring nodes v and w of equal color by letting the parent v swallow the child w as illustrated in Figure 4.6. We observe that since H and H' both obey I_1 a conflict must involve a node from both of them. This implies that a conflict can only occur in the part of the merged backbone made of nodes popped off the backbone-stacks in Step 1. We also observe that solving a conflict does not induce a new conflict. Combined with the previous observation this implies that the number of conflicts is bounded by the number of nodes popped off the backbone-stacks in Step 1. Finally, we observe that solving a conflict does not induce violations of I_2 and I_3, so after solving all conflicts on the merged backbone we have a colored heap-tree that stores all elements in H and H' with unchanged keys.

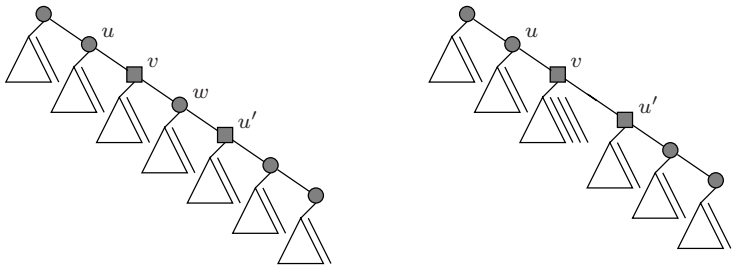


Figure 4.6: This figure illustrates how a conflict on the merged backbone is solved. If $color(v) = color(w)$ then I_1 is violated. The invariant is restored by letting node v swallow node w, i.e. melding the elements and left-children at the two nodes and removing node w. Since $color(u) \neq color(w) = color(v)$ and $color(u') \neq color(v)$, solving a conflict does not induce another conflict.

In Step 3 we shorten the merged backbone. This is done by moving the right-child of r' to its left-children, or in case r' has been swallowed by a node v in Step 2, by moving the right-child of v to its left-children. To argue that this does not induce violations of I_2 and I_3 we start by making two observations. First, we observe that moving the right-child of a node that obeys I_3 to its set of left-children results in a node that obeys I_2. Secondly, we observe that if a node that obeys I_2 (or I_3) swallows a node that obeys I_2 it results in a node that still obeys I_2 (or I_3).

Since r' is the root of H', it obeys I_3 before Step 2. We consider two cases. First, if r' is not swallowed in Step 2, the first observation immediately implies that it obeys I_2 after Step 3. Secondly, if r' is swallowed by a node v in Step 2, we might as well think of Step 2 and Step 3 as occurring in the opposite order as this does not affect the resulting tree. Hence, first we move the right-child of r' to its set of left-children, which by the first observation results in a node that obeys I_2, then we let node v swallow this node, which by the second observation does not affect the invariants obeyed by v.

We conclude that our implementation of $\mathsf{ColorMeld}(H, H')$ constructs a colored heap-tree that obeys all three invariants and stores all elements in H and H' with unchanged keys. It is easy to see that the backbone-stack of the colored heap-tree constructed by $\mathsf{ColorMeld}(H, H')$ is what remains on the backbone-stack of H after popping of nodes in Step 1 with the node r' pushed onto it, unless the node r' is swallowed in Step 2.

Now consider the time it takes to meld n colored heap-trees of size one together by a sequence of $n - 1$ melds. If we ignore the time it takes to meld the heap-trees storing elements and references to left-children when solving conflicts in Step 2 and shortening the backbone in Step 3, then we can bound the time it takes to do the sequence of melds by $O(n)$ exactly as we did in the previous section. It is easy to see that melding n colored heap-trees of size one involves melding at most n heap-trees of size one storing elements, and at most n heap-trees of size one storing references to left-children. Since melding n heap-trees of size one takes time $O(n)$, we have that melding the

heap-trees storing elements and references to left-children also takes time $O(n)$, so melding n colored heap-trees of size one takes time $O(n)$ in the worst case.

4.4.2 Algorithms

In the following we present two algorithms to find pairs with lower bounded gap. First we describe a simple algorithm to find all right-maximal pairs with lower bounded gap using heap-trees, then we extend it to find all maximal pairs with lower bounded gap using colored heap-trees. Both algorithms run in time $O(n + z)$ where z is the number of reported pairs.

Right-maximal pairs with lower bounded gap

We find all right-maximal pairs in S with gap at least $g(|\alpha|)$ by for each node v in the binary suffix tree $T_B(S)$ to consider the leaf-lists at its two children w_1 and w_2. The pair $(p, q, |\alpha|)$, $p \in LL(w_1)$ and $q \in LL(w_2)$, is right-maximal and has gap at least $g(|\alpha|)$ if and only if $q \geq p + |\alpha| + g(|\alpha|)$. If we let p_{min} denote the minimum element in $LL(w_1)$ this implies that every q in

$$Q = \{q \in LL(w_2) \mid q \geq p_{min} + |\alpha| + g(|\alpha|)\}$$

forms a right-maximal pair $(p, q, |\alpha)$ with gap at least $g(|\alpha|)$ with every p in

$$P_q = \{p \in LL(w_1) \mid p \leq q - g(|\alpha|) - |\alpha|\}.$$

By construction P_q contains p_{min} and we have that $(p, q, |\alpha|)$ is a right-maximal pair with gap at least $g(|\alpha|)$ if and only if $q \in Q$ and $p \in P_q$. We can construct Q and P_q using heap-trees. Let H_i and $\bar{H}_i$ be heap-trees that store the elements in $LL(w_i)$ ordered by "$\leq$" and "$\geq$" respectively. By definition of the operations Min and Find we have that $p_{min} = \mathsf{Min}(H_1)$, $Q = \mathsf{Find}(\bar{H}_2, p_{min} + |\alpha| + g(|\alpha|)$ and $P_q = \mathsf{Find}(H_1, q - g(|\alpha|) - |\alpha|)$.

This leads to the formulation of Algorithm 4 in which we at every node v in $T_B(S)$ construct two heap-trees, H and $\bar{H}$, that store the elements in $LL(v)$ ordered by "$\leq$" and "$\geq$" respectively. If v is a leaf, we construct H and $\bar{H}$ directly by creating two heap-trees of size one each storing the index at the leaf. If v is an internal node, we construct H and $\bar{H}$ by melding the corresponding heap-trees at the two children (lines 11–12). Before constructing H and $\bar{H}$ at node v, we report right-maximal pairs of its path-label (lines 1–10).

To argue that Algorithm 4 finds all right-maximal pairs in S with gap at least $g(|\alpha|)$ it is enough to argue that we at each node v in $T_B(S)$ report all pairs $(p, q, |\alpha|)$ and $(q, p, |\alpha|)$, $p \in LL(w_1)$ and $q \in LL(w_2)$, with gap at least $g(|\alpha|)$. The rest follows because we consider every node in $T_B(S)$. Let v be a node in $T_B(S)$ at which the heap-trees H_1, $\bar{H}_1$ and H_2, $\bar{H}_2$ at its two children are available. As explained above $(p, q, |\alpha|)$ is a right-maximal pair with gap at least $g(|\alpha|)$ if and only if $q \in Q$ and $p \in P_q$, which exactly are the pairs reported in lines 1–5. Symmetrically we can argue that $(q, p, |\alpha|)$ is a right-maximal pair with gap at least $g(|\alpha|)$ if and only if $p \in P$ and $q \in Q_p$, which exactly are the pairs reported in lines 6–10.

Algorithm 4 Find all right-maximal pairs in S with lower bounded gap.

1. *Initializing:* Build the binary suffix tree $T_B(S)$. Create at each leaf two heap-trees of size one, H ordered by "$\leq$" and $\bar{H}$ ordered by "$\geq$", that both store the index at the leaf.

2. *Reporting and melding:* When the heap-trees H_1 and $\bar{H}_1$ at the left-child of node v, and the heap-trees H_2 and $\bar{H}_2$ at the right-child of node v are available we report pairs of α, the path-label of v, and construct the heap-trees H and $\bar{H}$ as follows

```
1   Q = Find(H̄_2, Min(H_1) + |α| + g(|α|))
2   for q in Q:
3       P_q = Find(H_1, q − g(|α|) − |α|)
4       for p in P_q:
5           report pair (p, q, |α|)

6   P = Find(H̄_1, Min(H_2) + |α| + g(|α|))
7   for p in P:
8       Q_p = Find(H_2, p − g(|α|) − |α|)
9       for q in Q_p:
10          report pair (q, p, |α|)

11  H = Meld(H_1, H_2)
12  H̄ = Meld(H̄_1, H̄_2)
```

Now consider the running time of the algorithm. We first note that constructing two heap-trees of size one at each of the n leaves in $T_B(S)$ and melding them together according to the structure of $T_B(S)$ takes time $O(n)$ because each of the $n-1$ meld operation takes amortized constant time. We then note that the reporting of pairs at each node, lines 1–10, takes time proportional to the number of reported pairs because the find operation takes time proportional to the number of returned elements and the set P_q (and Q_p) is non-empty for every element q in Q (and p in P). Finally we remember that constructing the binary suffix tree $T_B(S)$ takes time $O(n)$. Now consider the space needed by the algorithm. The binary suffix tree requires space $O(n)$. The heap-trees also requires space $O(n)$ because no element at any time is stored in more than one heap-tree. Finally, since no leaf-list contains more than n elements, storing the elements returned by the find operations during the reporting requires no more than space $O(n)$. In summary we formulate the following theorem.

Theorem 3 *Algorithm 4 finds all right-maximal pairs $(i, j, |\alpha|)$ in a string S with gap at least $g(|\alpha|)$ in space $O(n)$ and time $O(n+z)$, where z is the number of reported pairs and n is the length of S.*

Maximal pairs with lower bounded gap

Essential to the above algorithm is that we in time proportional to its size can construct the set Q that contains all elements q in $LL(w_2)$ that form a right-maximal pair $(p_{min}, q, |\alpha|)$ with gap at least $g(|\alpha|)$. Unfortunately the left-characters $S[q-1]$ and $S[p_{min}-1]$ can be equal, so Q can contain elements that do not form a maximal pair with any element in $LL(w_1)$. Since we aim for the reporting of pairs to take time proportional to the number of reported pairs, this implies that we cannot afford to consider every element in Q if we only want to report maximal pairs.

Fortunately we can efficiently construct the subset of $LL(w_2)$ that contains all the elements that form at least one maximal pair. An element q in $LL(w_2)$ forms a maximal pair if and only if there is an element p in $LL(w_1)$ such that $q \geq p + |\alpha| + g(|\alpha|)$ and $S[q-1] \neq S[p-1]$. We can construct this subset of $LL(w_2)$ using colored heap-trees. We define the color of an element to be its left-character, i.e. the color of p in $LL(w_1)$ and q in $LL(w_2)$ is $S[p-1]$ and $S[q-1]$ respectively. Let H_i and $\bar{H}_i$ be colored heap-trees that store the elements in $LL(w_i)$ ordered by "$\leq$" and "$\geq$" respectively. Using $p_{min} = \mathsf{ColorMin}(H_1)$ and $p_{sec} = \mathsf{ColorSec}(H_1)$ we can characterize the elements in $LL(w_2)$ that form at least one maximal pair with gap at least $g(|\alpha|)$ by considering two cases.

First, if $q \geq p_{sec} + |\alpha| + g(|\alpha|)$ then $(p_{min}, q, |\alpha|)$ and $(p_{sec}, q, |\alpha|)$ both have gap at least $g(|\alpha|)$ and since $S[p_{min}-1] \neq S[p_{sec}-1]$ at least one of them is maximal, so every $q \geq p_{sec} + |\alpha| + g(|\alpha|)$ forms a maximal pair with gap at least $g(|\alpha|)$. If $\#$ is a character not appearing anywhere in S, i.e. no element in $LL(w_2)$ has color $\#$, this is the same as saying that every q in $Q' = \mathsf{ColorFind}(\bar{H}_2, p_{sec} + |\alpha| + g(|\alpha|), \#)$ forms a maximal pair with gap at least $g(|\alpha|)$. Secondly, if $q < p_{sec} + |\alpha| + g(|\alpha|)$ forms a maximal pair $(p, q, |\alpha|)$ with gap at least $g(|\alpha|)$ then $p_{min} \leq p < p_{sec}$. This implies that $S[p-1] = S[p_{min}-1]$, so $(p_{min}, q, |\alpha|)$ is also maximal and has gap at least $g(|\alpha|)$. We thus have that $q < p_{sec} + |\alpha| + g(|\alpha|)$ forms a maximal pairs with gap at least $g(|\alpha|)$ if and only if $(p_{min}, q, |\alpha|)$ is maximal and has gap at least $g(|\alpha|)$, i.e. if and only if $S[q-1] \neq S[p_{min}-1]$ and $q \geq p_{min} + |\alpha| + g(|\alpha|)$. This implies that the set $Q'' = \mathsf{ColorFind}(\bar{H}_2, p_{min} + |\alpha| + g(|\alpha|), S[p_{min}-1])$ contains every $q < p_{sec} + |\alpha| + g(|\alpha|)$ that forms a maximal pair with gap at least $g(|\alpha|)$.

By construction of Q' and Q'' the set $Q' \cup Q''$ contains all elements in $LL(w_2)$ that form a maximal pair with gap at least $g(|\alpha|)$. More precisely, every q in $Q' \cup Q''$ forms a maximal pair $(p, q, |\alpha|)$ with gap at least $g(|\alpha|)$ with every $p \leq q - g(|\alpha|) - |\alpha|$ in $LL(w_1)$ where $S[p-1] \neq S[q-1]$, i.e. every p in $P_q = \mathsf{ColorFind}(H_1, q - g(|\alpha|) - |\alpha|, S[q-1])$ which by construction is non-empty. We can construct $Q' \cup Q''$ efficiently. Every element in Q'' greater than $p_{sec} + |\alpha| + g(|\alpha|)$ is also in Q', so we can construct $Q' \cup Q''$ by concatenating Q' and what remains of Q'' after removing all elements greater than $p_{sec} + |\alpha| + g(|\alpha|)$ from it. This together with the complexity of $\mathsf{ColorFind}$ implies that we can construct $Q' \cup Q''$ in time proportional to $|Q'| + |Q''| \leq 2|Q' \cup Q''|$.

This leads to the formulation of Algorithm 5. The algorithm is similar to Algorithm 4 except that we maintain colored heap-trees during the traversal of

Algorithm 5 Find all maximal pairs in S with lower bounded gap.

1. *Initializing:* Build the binary suffix tree $T_B(S)$. Create at each leaf two colored heap-trees of size one, H ordered by "$\leq$" and $\bar{H}$ ordered by "$\geq$", that both store the index at the leaf with color corresponding to its left-character.

2. *Reporting and melding:* When the colored heap-trees H_1 and $\bar{H}_1$ at the left-child of node v, and the colored heap-trees H_2 and $\bar{H}_2$ at the right-child of node v are available we report pairs of α, the path-label of v, and construct the colored heap-trees H and $\bar{H}$ as follows (remember that # is a character not appearing anywhere in S)

```
1   p_min, p_sec = ColorMin(H_1), ColorSec(H_1)
2   Q' = ColorFind(H̄_2, p_sec + |α| + g(|α|), #)
3   Q'' = ColorFind(H̄_2, p_min + |α| + g(|α|), S[p_min − 1])
4   for q in Q' ∪ Q'':
5       P_q = ColorFind(H_1, q − g(|α|) − |α|, S[q − 1])
6       for p in P_q:
7           report pair (p, q, |α|)

8   q_min, q_sec = ColorMin(H_2), ColorSec(H_2)
9   P' = ColorFind(H̄_1, q_sec + |α| + g(|α|), #)
10  P'' = ColorFind(H̄_1, q_min + |α| + g(|α|), S[q_min − 1])
11  for p in P' ∪ P'':
12      Q_p = ColorFind(H_2, p − g(|α|) − |α|, S[p − 1])
13      for q in Q_p:
14          report pair (q, p, |α|)

15  H = ColorMeld(H_1, H_2)
16  H̄ = ColorMeld(H̄_1, H̄_2)
```

the binary suffix tree. At every node we report maximal pairs of its path-label. In lines 1–7 we report all maximal pairs $(p, q, |\alpha|)$ by constructing and considering the elements in P_q for every q in $Q' \cup Q''$. In lines 8–15 we analogously report all maximal pairs $(q, p, |\alpha|)$. The correctness of the algorithm follows immediately from the above discussion. Since the operations on colored heap-trees have the same complexities as the corresponding operations on heap-tress, the running time and space requirement of the algorithm is exactly as analyzed for Algorithm 4. In summary we can formulate the following theorem.

Theorem 4 *Algorithm 5 finds all maximal pairs $(i, j, |\alpha|)$ in a string S with gap at least $g(|\alpha|)$ in space $O(n)$ and time $O(n + z)$, where z is the number of reported pairs and n is the length of S.*

4.5 Conclusion

We have presented efficient and flexible methods to find all maximal pairs $(i, j, |\alpha|)$ in a string under various constraints on the gap $j - i - |\alpha|$. If the gap is required to be between $g_1(|\alpha|)$ and $g_2(|\alpha|)$, the running time is $O(n \log n + z)$ where n is the length of the string and z is the number of reported pairs. If the gap is only required to be at least $g_1(|\alpha|)$, the running time reduces to $O(n+z)$. In both cases we use space $O(n)$.

In some cases it might be interesting only to find maximal pairs $(i, j, |\alpha|)$ fulfilling additional requirements on $|\alpha|$, e.g. to filter out pairs of short substrings. This is straightforward to do using our methods by only reporting from the nodes in the binary suffix tree whose path-label α fulfills the requirements on $|\alpha|$. In other cases it might be of interest just to find the vocabulary of substrings that occur in maximal pairs. This is also straightforward to do using our methods by just reporting the path-label α of a node if we can report one or more maximal pairs from the node.

Instead of just looking for maximal pairs, it could be interesting to look for an array of occurrences of the same substring in which the gap between consecutive occurrences is bounded by some constants. This problem requires a suitable definition of a maximal array. One definition and approach is presented in [127]. Another definition inspired by the definition of a maximal pair could be to require that every pair of occurrences in the array is a maximal pair. This definition seems very restrictive. A more relaxed definition could be to only require that we cannot extend all the occurrences in the array to the left or to the right without destroying at least one pair of occurrences in the array.

Acknowledgments

This work was initiated while Christian N. S. Pedersen and Jens Stoye were visiting Dan Gusfield at UC Davis. We would like to thank Dan Gusfield, as well as Rob Irwing, for listening to some preliminary results.

Chapter 5

Comparison of coding DNA

> In the grand design, women were definitely drawn from a different set of blueprints.
>
> —Dale Cooper, *Twin Peaks*

This paper describes an algorithm to compare two DNA sequences when considering both the DNA sequences themselves, and the proteins encoded by the DNA sequences. The results were presented at the Ninth Annual Conference on Combinatorial Pattern Matching and published in the proceedings of this conference [116]. Furthermore, a similar paper has been published in the BRICS report series [117]. The algorithm has been implemented and the source code is available at `http://www.brics.dk/~cstorm/combat/index.html`.

Comparison of coding DNA

Christian N. S. Pedersen* Rune B. Lyngsø* Jotun J. Hein†

Abstract

We discuss a model for the evolutionary distance between two coding DNA sequences which specializes to the DNA/protein model proposed in Hein [60]. We discuss the DNA/protein model in details and present a quadratic time algorithm that computes an optimal alignment of two coding DNA sequences in the model under the assumption of affine gap cost. The algorithm solves a conjecture in [60] and we believe that the constant factor of the running time is sufficiently small to make the algorithm feasible in practice.

5.1 Introduction

A straightforward model of the evolutionary distance between two coding DNA sequences is to ignore the encoded proteins and compute the distance in some evolutionary model of DNA. We say that such a model is a DNA level model. The evolutionary distance between two sequences in a DNA level model can most often be formulated as a classical alignment problem and be efficiently computed using a dynamic programming approach [108, 130, 132, 151].

It is well known that proteins evolve slower than its coding DNA, so it is usually more reliable to describe the evolutionary distance based on a comparison of the encoded proteins rather than on a comparison of the coding DNA itself. Hence, most often the evolutionary distance between two coding DNA sequences is modeled in terms of amino acid events, such as substitution of a single amino acid and insertion-deletion of consecutive amino acids, necessary to transform the one encoded protein into the other encoded protein. We say that such a model is a protein level model. The evolutionary distance between two coding DNA sequences in a protein level model can most often be formulated as a classical alignment problem of the two encoded proteins. Even though a protein level model is usually more reliable than a DNA level model, it falls short because it postulates that all insertions and deletions on the underlying DNA occur at codon boundaries and because it ignores similarities on the DNA level.

In this paper we present a model of the evolutionary distance between two coding DNA sequences in which a nucleotide event is penalized by the change it

*Basic Research in Computer Science (BRICS), Centre of the Danish National Research Foundation, Department of Computer Science, University of Aarhus, Ny Munkegade, 8000 Århus C, Denmark. E-mail: {cstorm,rlyngsoe}@brics.dk.

†Department for Ecology and Genetics, University of Aarhus, Ny Munkegade, 8000 Århus C, Denmark. E-mail: jotun@pop.bio.aau.dk

induces on the DNA as well as on the encoded protein. The model is a natural combination of a DNA level model and a protein level model. The DNA/protein model introduced in Hein [60, 62] is a biological reasonable instance of the general model in which the evolution of coding DNA is idealized to involve only substitution of a single nucleotide and insertion-deletion of a multiple of three nucleotides. Hein [60, 62] presents an $O(n^2m^2)$ time algorithm for computing the evolutionary distance in the DNA/protein model between two sequences of length n and m. This algorithm assumes certain properties of the cost function. We discuss these properties and present an $O(nm)$ time algorithm that solves the same problem under the assumption of affine gap cost.

The practicality of an algorithm not only depends on the asymptotic running time but also on the constant factor hidden by the use of O-notation. To determine the distance between two sequences of length n and m our algorithm computes $400nm$ table entries. Each computation involves a few additions, table lookups and comparisons. We believe the constant factor is sufficiently small to make the algorithm feasible in practice.

The problem of comparing coding DNA is also discussed by Arvestad [10] and Hua, Jiang and Wu [67]. The models discussed in these papers are inspired by the DNA/protein model in Hein [60, 62] but differ in the interpretation of gap cost. A heuristic algorithm for solving the alignment problem in the DNA/protein model is described In Hein [61]. A related problem of how to compare a coding DNA sequence with a protein has been discussed in [119].

The rest of this paper is organized as follows: In Sect. 5.2 we introduce and discuss the DNA/protein model. In Sect. 5.3 we describe how to determine the cost of an alignment. In Sect. 5.4 we present the simple alignment algorithm of Hein [60]. In Sect. 5.5 we present a quadratic time alignment algorithm. Finally, in Sect. 5.6 we discuss future work.

5.2 The DNA/protein model

Let $a = a_1a_2a_3 \dots a_{3n-2}a_{3n-1}a_{3n}$ be a coding sequence of DNA of length $3n$ with a reading frame starting at a_1. We introduce the notation $a_1^i a_2^i a_3^i$ to denote the ith codon $a_{3i-2}a_{3i-1}a_{3i}$ and the notation A_i to describe the amino acid coded by the ith codon. The amino acid sequence $A = A_1A_2 \dots A_n$ describes the protein coded by a. Let $b = b_1b_2b_3 \dots b_{3m-2}b_{3m-1}b_{3m}$, $b_1^i b_2^i b_3^i$ and $B = B_1B_2 \dots B_m$ be defined similarly.

5.2.1 The general model

An evolutionary event e on the DNA that transforms a to a' will also change the encoded protein from A to A'. As some amino acids are coded by several codons, the proteins A and A' might be identical. The cost of e should reflect the changes on the DNA as well as the changes on the encoded protein.

$$cost(a \xrightarrow{e} a') = cost_d(a \xrightarrow{e} a') + cost_p(A \xrightarrow{e} A') \tag{5.1}$$

We say that $cost_d(a \xrightarrow{e} a')$ is the DNA level cost of e and that $cost_p(A \xrightarrow{e} A')$ is the protein level cost of e. In this paper we assume that the DNA level cost and

the protein level cost are combined by addition but other combination functions $f : \mathbf{R} \times \mathbf{R} \to \mathbf{R}$ could of course also be considered.

The cost of a sequence E of evolutionary events $e_1, e_2, \ldots, e_k$ transforming $a^{(0)}$ to $a^{(k)}$ as $a^{(0)} \xrightarrow{e_1} a^{(1)} \xrightarrow{e_2} a^{(2)} \xrightarrow{e_3} \cdots \xrightarrow{e_k} a^{(k)}$ is defined as some function of the costs of each event. In the rest of this paper we will assume that this function is the sum of the costs of each event.

$$cost(a^{(0)} \xrightarrow{E} a^{(k)}) = \sum_{i=1}^{k} cost(a^{(i-1)} \xrightarrow{e_i} a^{(i)}) \tag{5.2}$$

We define the distance between two coding sequences of DNA a and b according to the parsimony principle as the minimum cost of a sequence of evolutionary events which transforms a to b.

$$dist(a, b) = \min\{cost(a \xrightarrow{E} b) \mid E \text{ is a sequence of events}\} \tag{5.3}$$

In order to compute $dist(a, b)$ we have to specify the set of allowed evolutionary events and define the cost of each event on the DNA level as well as on the protein level. The choice of evolutionary events and cost function influences both the biological relevance of the distance measure and the computational complexity of computing the distance.

5.2.2 The specific model

The DNA/protein model introduced in [60] can be described as an instance of the general model where the evolution of coding DNA is idealized to involve only substitutions of single nucleotide and insertion-deletions of a multiple of three consecutive nucleotides. The DNA level cost of an event is defined in the classical way by specifying a substitution cost and a gap cost. The protein level cost of an event that changes the encoded protein from A to A' is defined to reflect the difference between protein A and protein A'.

- The DNA level cost $cost_d(a \xrightarrow{e} a')$ depends on e. The cost of substituting a nucleotide σ with σ' is $c_d(\sigma, \sigma')$ for some metric c_d on nucleotides. The cost of inserting or deleting $3k$ consecutive nucleotides is $g_d(3k)$ for some subadditive[1] function $g_d : \mathbf{N} \to \mathbf{R}^+$.

- The protein level cost $cost_p(A \xrightarrow{e} A')$ is defined as the distance $dist_p(A, A')$ between A and A'. The distance $dist_p(A, A')$ is the minimum cost of a distance alignment of A and A' where we allow substitution of a single amino acid and insertion-deletion of consecutive amino acids. The substitution cost is given by a metric c_p on amino acids and the gap cost is given by a subadditive function $g_p : \mathbf{N} \to \mathbf{R}^+$. Additional restrictions will be given in Sect. 5.2.3.

[1] A function is subadditive if $f(i+j) \leq f(i)+f(j)$. A subadditive gap cost function implies that an insertion-deletion of a consecutive block of nucleotides is best explained as a single event.

The reason why gap lengths are restricted to a multiple of three is that an insertion or deletion of length not divisible by three changes the reading frame. This is called a frame shift and it may change the entire remaining amino acid sequence as illustrated in Fig. 5.1. Frame shifts are believed to be rare biological events, so it is not unreasonable to leave them out of the model.

Figure 5.1: An insertion-deletion of length not divisible by three changes the reading frame.

Except for the restriction on insertion-deletion length the DNA/protein model allows the traditional set of symbol based nucleotide events. This allows us to use the notion of an alignment. An alignment of two sequences describes a set of substitution or insertion-deletion events necessary to transform one of the sequences into the other sequence. The set of events is usually described by a matrix or a path in a graph as illustrated in Fig. 5.2. The cost of an alignment is the optimal cost of any sequence of the events described by the alignment. Hence, the evolutionary distance $dist(a, b)$ in the DNA/protein model between two coding DNA sequences a and b is the cost of an optimal alignment of a and b in the DNA/protein model. In the rest of this paper we will address the problem of computing the cost of an optimal alignment in the DNA/protein model.

$$\begin{bmatrix} \mathrm{T} & \mathrm{T} & \mathrm{G} & \mathrm{C} & \mathrm{T} & - & - & - & \mathrm{C} \\ \mathrm{T} & - & - & - & \mathrm{C} & \mathrm{A} & \mathrm{T} & \mathrm{G} & \mathrm{C} \end{bmatrix}$$

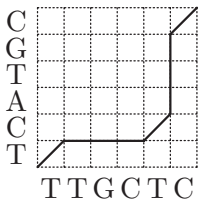


Figure 5.2: An alignment can be described by a matrix or a path in the alignment graph. The above alignment describes three matches and two gaps of combined length six.

If the cost of any sequence of events is independent of the order but only depends on the set of events, then an optimal alignment can be computed efficiently using dynamic programming [108, 130, 132, 151]. In the DNA/protein model the cost of an event is the sum of the DNA level cost and the protein level cost. We observe that the DNA level cost of a sequence of events is independent of the order but that the protein level cost is not [60, Fig. 2]. This implies that we cannot use a classical alignment algorithm to compute an optimal alignment in the DNA/protein model. In order to formulate an efficient alignment algorithm in the DNA/protein model we must examine the protein level cost further.

5.2.3 Restrictions on the cost function

A single nucleotide event affects nucleotides in one or more consecutive codons. Since a nucleotide event in the DNA/protein model cannot change the reading frame then only the amino acids encoded by the affected codons are affected by the nucleotide event. A nucleotide event thus changes protein $A = UXV$ to protein $A' = UX'V$ where X and X' are the amino acids affected by nucleotide event. Let us consider the protein level cost $dist_p(A, A')$ of the event.

Hein [60] implicitly assumes that $dist_p(A, A')$ is the cost of a distance alignment of X and X' with the minimum number of insertion-deletions. This assumption implies that $dist_p(A, A')$ is the cost of one of the alignments of A and A' shown in Fig. 5.3. This assumption is essential to the formulation of our alignment algorithms in Sect. 5.4 and 5.5.

If the cost of alignment of A and A' implied by Fig. 5.3 is not minimal then the assumption conflicts with our previous definition of $dist_p(A, A')$ as being the minimum cost of an alignment of A and A'. Lemma 9 states restrictions on c_p and g_p that prevent this conflict. If we assume an affine gap cost function $g_p(k) = \alpha_p + \beta_p k$ for some $\alpha_p, \beta_p \geq 0$ (and define $g_p(0)$ to be zero), then the restrictions on c_p and g_p becomes $c_p(\sigma, \tau) + \alpha_p + \beta_p k \leq 2\alpha_p + \beta_p(k + 2l)$ for any amino acids σ, τ and all lengths $0 < l \leq n - k$. This simplifies to $c_p(\sigma, \tau) \leq \alpha_p + 2\beta_p$ for all amino acids σ, τ which is a biological reasonable restriction as insertions and deletions are rare events compared to substitutions.

$$\begin{bmatrix} A_1 & A_2 & \cdots & A_{i-1} & A_i & A_{i+1} & \cdots & A_n \\ A_1 & A_2 & \cdots & A_{i-1} & A'_i & A_{i+1} & \cdots & A_n \end{bmatrix}$$

(a) A substitution in the ith codon. The cost is $c_p(A_i, A'_i)$.

$$\begin{bmatrix} A_1 & A_2 & \cdots & A_{i-1} & A_i & A_{i+1} & \cdots & A_{i+k} & A_{i+k+1} & \cdots & A_n \\ A_1 & A_2 & \cdots & A_{i-1} & A_i & - & \cdots & - & A_{i+k+1} & \cdots & A_n \end{bmatrix}$$

(b) An insertion-deletion of $3k$ nucleotides affecting exactly k codons. The cost is $g_p(k)$.

$$\begin{bmatrix} A_1 & A_2 & \cdots & A_{i-1} & A_i & \cdots & A_{j-1} & A_j & A_{j+1} & \cdots & A_{i+k} & A_{i+k+1} & \cdots & A_n \\ A_1 & A_2 & \cdots & A_{i-1} & - & \cdots & - & \upsilon & - & \cdots & - & A_{i+k+1} & \cdots & A_n \end{bmatrix}$$

(c) An insertion-deletion of $3k$ nucleotides affecting $k + 1$ codons. The remaining amino acid υ is matched with one of the amino acids affected by the deletion. The cost is $\min_{j=0,1,\ldots,k} \{g_p(j) + c_p(A_{i+j}, \upsilon) + g_p(k - j)\}$.

Figure 5.3: The protein level cost of a nucleotide event can be determined by considering only the amino acids affected by the event.

Lemma 9 *Assume a nucleotide event changes $A = UXV$ to $A' = UX'V$. Let $n = |A|$ and $k = ||A| - |A'||$. If there for any amino acids σ, τ and for all $0 < l \leq n-k$ exists $0 \leq j \leq k$ such that $c_p(\sigma,\tau)+g_p(j)+g_p(k-j) \leq g_p(l)+g_p(l+k)$, then $dist_p(A, A')$ is the cost of an alignment describing exactly k insertions or deletions. Furthermore $dist_p(A, A')$ only depends on X and X'.*

Proof. We note that the alignments in Fig. 5.3 all describe the minimum number of insertion-deletions and that only the sub-alignment of X and X', as illustrated by the shaded parts, contributes to the cost. We will argue that the assumption on c_p and g_p stated in the lemma ensures that $dist_p(A, A')$ is the cost of one of the alignments in Fig. 5.3. We split the argumentation depending on the event. Since $dist_p(A, A')$ is equal to $dist_p(A', A)$ the cost of an insertion transforming A to A' is equal to the cost of a deletion transforming A' to A. We thus only consider substitutions and deletions.

A substitution of a nucleotide in the ith codon of A transforms A_i to A'_i. The alignment in Fig. 5.3(a) describes no insertion-deletions and has cost $c_p(A_i, A'_i)$. Any other alignment of A and A' must describe an equal number of insertions and deletions, so by subadditivity of g_p the cost is at least $2g_p(l)$ for some $0 < l \leq n$. The assumption in the lemma implies that $c_p(A_i, A'_i) \leq 2g_p(l)$ for any $0 < l \leq n$. The alignment in Fig. 5.3(a) is thus optimal and the protein level cost of the substitution is $c_p(A_i, A'_i)$.

A deletion of $3k$ nucleotides affecting k codons transforms $A = A_1A_2 \cdots A_n$ to $A' = A_1A_2 \cdots A_iA_{i+k+1}A_{i+k+2} \cdots A_n$. Any alignment of A and A' must describe l insertions and $l+k$ deletions for some $0 \leq l \leq n-k$, so the cost is at least $g_p(l)+g_p(l+k)$. The alignment in Fig. 5.3(b) describes k deletions and has cost $g_p(k)$. The assumption in the lemma and the sub-additivity of g_p implies that $g_p(k) \leq g_p(j) + g_p(k-j) \leq g_p(l) + g_p(l+k)$ for all $l > 0$. The alignment in Fig. 5.3(b) is thus optimal and the protein level cost of the deletion is $g_p(k)$.

A deletion of $3k$ nucleotides affecting $k+1$ codons, say a deletion of the $3k$ nucleotides $a_3^i a_1^{i+1} a_2^{i+1} a_3^{i+1} \cdots a_1^{i+k} a_2^{i+k}$, transforms $A = A_1A_2 \cdots A_n$ to $A' = A_1A_2 \cdots A_{i-1}\upsilon A_{i+k+1} \cdots A_n$ where υ is the amino acid coded by $a_1^i a_2^i a_3^{i+k}$. We say that υ is the remaining amino acid and $a_1^i a_2^i a_3^{i+k}$ is the remaining codon. Any alignment of A and A' describing exactly k deletions must align υ with A_{i+j} for some $0 \leq j \leq k$, so by subadditivity of g_p the cost is at least $g_p(j) + c_p(A_{i+j}, \upsilon) + g_p(k-j)$. The alignment in Fig. 5.3(c) illustrates one of the $k+1$ alignments of A and A' where υ is aligned with an affected amino acid and all non-affected amino acids are aligned. Such an alignment describes exactly k deletions and the cost of the optimal alignment among them has cost

$$\min_{j=0,1,\ldots,k} \{g_p(j) + c_p(A_{i+j}, \upsilon) + g_p(k-j)\}, \tag{5.4}$$

and is thus optimal for any alignment describing exactly k deletions. Any other alignment of A and A' must describe l insertions and $l+k$ deletions for some $0 < l \leq n-k$, so the cost is at least $g_p(l) + g_p(l+k)$. The assumption in the lemma implies that the cost given by (5.4) is less than or equal to $g_p(l)+g_p(l+k)$. The protein level cost of the deletion is thus given by (5.4). □

The assumption in Lemma 9 is sufficient to ensure that we can compute the protein level cost of a nucleotide event efficiently, but the formulation of the lemma is too general to make the assumption necessary. The following example however suggests when the assumption is necessary. Consider a deletion of three nucleotides that transforms the six amino acids ABEFCD to ABGCD, i.e. $X = \mathrm{EF}$ and $X' = \mathrm{G}$. Consider the two alignments shown in Fig. 5.4.

$$\begin{bmatrix} \mathrm{A} & \mathrm{B} & \mathrm{E} & \mathrm{F} & \mathrm{C} & \mathrm{D} \\ \mathrm{A} & \mathrm{B} & \mathrm{G} & - & \mathrm{C} & \mathrm{D} \end{bmatrix} \qquad \begin{bmatrix} \mathrm{A} & \mathrm{B} & \mathrm{E} & \mathrm{F} & - & \mathrm{C} & \mathrm{D} \\ \mathrm{A} & \mathrm{B} & - & - & \mathrm{G} & \mathrm{C} & \mathrm{D} \end{bmatrix}$$

Figure 5.4: Two alignments of the amino acids ABEFCD and ABGCD.

If we assume that $c_p(\mathrm{E}, \mathrm{G}) \leq c_p(\mathrm{F}, \mathrm{G})$ then the cost of the alignment in Fig. 5.4 (left) is $c_p(\mathrm{E}, \mathrm{G}) + g_p(1)$ while the cost of the alignment in Fig. 5.4 (right) is $g_p(2) + g_p(1)$. If the assumption in lemma 9 does not hold then $g_p(2) + g_p(1)$ might be less than $c_p(\mathrm{E}, \mathrm{G}) + g_p(1)$ because $c_p(\mathrm{E}, \mathrm{G})$ can be arbitrary large. Hence, the protein level cost of the deletion would not be the cost of an alignment describing the minimum number of insertion-deletions.

5.3 The cost of an alignment

Before we can describe how to compute the cost of an optimal alignment of two sequence in the DNA/protein model, we need to know how to compute the cost of a given alignment in the model.

An alignment of two sequences describes a set of events necessary to transform one of the sequences into the other sequence but it does not describe the order of the events. As observed in Sect. 5.2.2 the DNA level cost of an alignment is independent of the order of the events, while the protein level cost of an alignment depends on the order of the events. This implies that the DNA level cost of an alignment is just the sum of the DNA level cost of the events described by the alignment, while the protein level cost of the same alignment is somewhat harder to determine. An obvious way to determine the protein level cost of an alignment is to minimize over all possible sequences of the events described by the alignment. This method is however not feasible in practice due to the factorial number of possible sequences one has to consider.

If Lemma 9 is fulfilled then we know that the protein level cost of a nucleotide event only depends on the affected codons. We can use this property to decompose the computation of the protein level cost of an alignment into smaller subproblems. The idea is to decompose the alignment into *codon alignments*. A codon alignment is a minimal part of the alignment that corresponds to a path connecting two nodes $(3i', 3j')$ and $(3i, 3j)$ in the alignment graph. We can decompose an alignment uniquely into codon alignments as illustrated in Fig. 5.5.

The assumption that the length of an insertion or deletion is a multiple of three implies that a codon alignment either describes no substitutions (see Type 2 and 3 in Fig. 5.7) or exactly three substitutions. If a codon alignment describes exactly three substitutions then it can also describe one or more

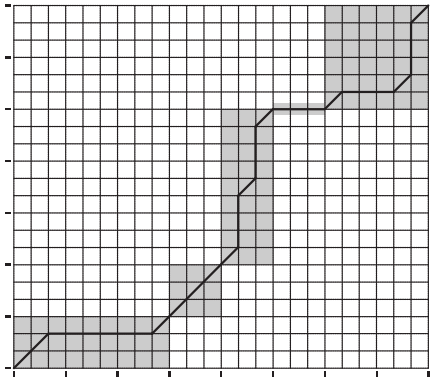


Figure 5.5: An alignment of two sequences decomposed into codon alignments.

insertion-deletions. More precisely, between any two consecutive substitutions in the codon alignment there can be an alternating sequence of insertions and deletions each with length a multiple of three. Such a sequence of insertion-deletions corresponds to a "staircase" in the alignment graph. The set of codon alignments that describe three substitutions and at most one insertion and deletion between two consecutive substitutions, i.e. the "staircase" is limited to at most one "step", is illustrated in Fig. 5.6. A particular codon alignment in this set corresponds to a choice of which sides of the two rectangles to traverse.

Now consider the decomposition of an alignment into codon alignments (this could be as illustrated in Fig. 5.5). We observe that nucleotide events described by two different codon alignments in the decomposition do not affect the same codons. Hence, the protein level cost of a codon alignment can be computed independently of the other codon alignments in the decomposition. We can thus compute the protein level cost of an alignment as the sum of the protein level cost of each of the codon alignments in the decomposition.

Since a codon alignment can describe an alternating sequence of insertion-deletions between two consecutive substitutions it is possible that a decomposition of an alignment of two sequences of length n and m contains codon alignments describing $\Theta(n+m)$ events. This implies that the problem of computing the cost of the codon alignments in a decompsition of an alignment is, in the worst case, not any easier than computing the cost of the alignment itself. One way to circumvent this problem is to only consider alignments that can be decomposed into (or built of) codon alignments with at most some maximum number of insertion-deletions between two consecutives substitutions. This upper bounds the number of events described by any codon alignment by some constant. Hence, we can determine the cost of a codon alignment in constant time simply by minimizing over all possible sequences of the events described by the codon alignment. The protein level cost of an alignment can thus be determined in time proportional to the number of codon alignments in the decomposition of the alignment.

In Hein [60] at most one insertion *or* one deletion is allowed between two consecutive substitutions in a codon alignment. Besides the two codon alignments

that describe no substitutions, this corresponds to the set of codon alignments we obtain from Fig. 5.6 when we require that the width and/or the height of each of the two rectangles must be zero. This implies that there are eleven different types of codon alignments. The eleven types are shown in Fig. 5.7.

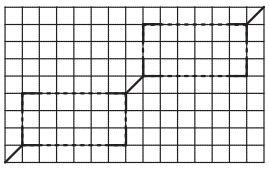


Figure 5.6: A summary of all possible codon alignments with at most one insertion and one deletion between two consecutive substitutions.

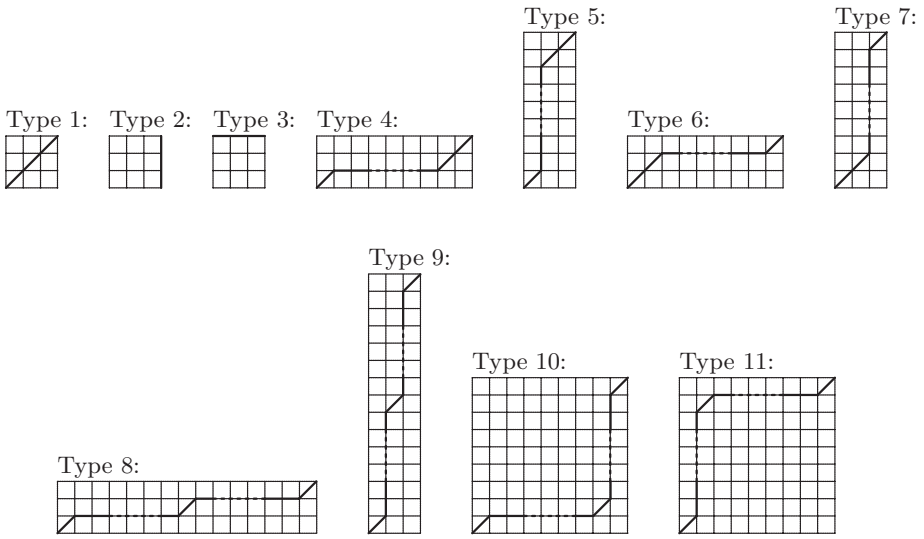


Figure 5.7: The eleven types of codon alignments with at most one inseriton or one deletion between two consecutive substitutions.

5.4 A simple alignment algorithm

Let $a_1a_2\cdots a_{3n}$ and $b_1b_2\cdots b_{3m}$ be two coding sequences of DNA. Hein [60] describes how the decomposition into codon alignments makes it possible to compute the cost of an optimal alignment of a and b in the DNA/protein model in time $\mathrm{O}(n^2m^2)$. The algorithm assumes that Lemma 9 is fulfilled and that we only allow codon alignments with some maximum number of insertion-deletions between two consecutive substitutions, e.g. the eleven types of codon alignments in Fig. 5.7. The algorithm can be summarized as follows.

Let $D(i,j)$ denote the cost of an optimal alignment of $a_1a_2\cdots a_{3i}$ and $b_1b_2\cdots b_{3j}$. We define $D(0,0)$ to be zero and $D(i,j)$ to be infinity for $i<0$ or $j<0$. An optimal alignment of $a_1a_2\cdots a_{3i}$ and $b_1b_2\cdots b_{3j}$ can be decom-

posed into codon alignments $ca_1, ca_2, \ldots, ca_k$. We say that ca_k is the last codon alignment and that $ca_1, ca_2, \ldots, ca_{k-1}$ is the remaining alignment.

If the last codon alignment ca_k in an optimal alignment is an alignment of $a_{3i'+1}a_{3i'+2}\cdots a_{3i}$ and $b_{3j'+1}b_{3j'+2}\cdots b_{3j}$ for some $(i', j') < (i, j)$[2], then $D(i, j)$ is equal to $D(i', j') + cost(ca_k)$. This is the cost of the last codon alignment plus the cost of the remaining alignment. We can compute $D(i, j)$ by minimizing the expression $D(i', j') + cost(ca)$ over all $(i', j') < (i, j)$ and all possible codon alignments ca of $a_{3i'+1}a_{3i'+2}\cdots a_{3i}$ and $b_{3j'+1}b_{3j'+2}\cdots b_{3j}$.

The upper bound on the number of insertion-deletions in a codon alignment implies that the number of possible codon alignments of $a_{3i'+1}a_{3i'+2}\cdots a_{3i}$ and $b_{3j'+1}b_{3j'+2}\cdots b_{3j}$, for all $(i', j') < (i, j)$, is upper bounded by some constant. For example, if we only consider the eleven types of codon alignments in Fig. 5.7 then there are at most three possible codon alignments of $a_{3i'+1}a_{3i'+2}\cdots a_{3i}$ and $b_{3j'+1}b_{3j'+2}\cdots b_{3j}$. Hence, if we assume that $D(i', j')$ is known for all $(i', j') < (i, j)$ then we can compute $D(i, j)$ in time $\mathrm{O}(ij)$. By dynamic programming this implies that we can compute $D(n, m)$ in time $\mathrm{O}(n^2m^2)$ and space $\mathrm{O}(nm)$. By back-tracking we can also get the optimal alignment (and not only the cost) within the same time and space bound.

5.5 An improved alignment algorithm

Let a and b be coding sequences of DNA as introduced in the previous section. We will describe how to compute the cost of an optimal alignment of a and b in the DNA/protein model in time $\mathrm{O}(nm)$ and space $\mathrm{O}(n)$. Besides the assumptions of the simple algorithm described in the previous section we also assume that the function $g(k) = g_d(3k)+g_p(k)$ is affine $\alpha+\beta k$ for some $\alpha, \beta \geq 0$. We say that g is the *combined gap cost function*.

The idea behind the improved algorithm is similar to the idea behind the simple algorithm in the sense that we compute the cost of an optimal alignment by minimizing the cost over all possible last codon alignments. We define $D^t(i, j)$ to be the cost of an optimal alignment of $a_1a_2\cdots a_{3i}$ and $b_1b_2\cdots b_{3j}$ under the assumption that the last codon alignment is of type t. Remember that we only allow codon alignments with some maximum number of insertion-deletions between two consecutive substitutions. This implies that the number of possible codon alignment, i.e. types of codon alignments, is upper bounded by some constant. We define $D^t(0, 0)$ to be zero and $D^t(i, j)$ to be infinity for $i < 0$ or $j < 0$. We can compute $D(i, j)$ as

$$D(i, j) = \min_t D^t(i, j). \tag{5.5}$$

Lemma 9 ensures that $D^t(i, j)$ is the cost of some last codon alignment (of type t) plus the cost of the corresponding remaining alignment. This allows us to compute $D^t(i, j)$ by minimizing the cost over all possible last codon alignments of type t. The assumption that g is affine makes it possible to compute $D^t(i, j)$ in constant time if $D^t(k, l)$, for all t, is known for some $(k, l) < (i, j)$. Since we

[2]We say that $(i', j') < (i, j)$ iff $i' \leq i \wedge j' \leq j \wedge (i' \neq i \vee j' \neq j)$.

only consider a constant number of possible codon alignments (e.g. the types in Fig. 5.7) this implies that we can compute $D(n, m)$ in time $O(nm)$ and space $O(n)$. By adapting the technique in Hirschberg [63] or the variant described in Durbin et al. [35, Page 35–36] we can also get the optimal alignment (and not only the cost) within the same time and space bound.

The method we use to compute $D^t(i, j)$ can be used for any type of last codon alignment but the bookkeeping and thereby the constant overhead increases with the number of gaps described by the last codon alignment. By restricting ourselves to the eleven types of codon alignments shown in Fig. 5.7 we can compute the cost of an optimal alignment with a reasonable constant overhead. In the rest of this paper we therefore focus on these eleven types of codon alignments. We divide the explanation of how to compute $D^t(i, j)$ for $t = 1, 2, \ldots, 11$ according to the number of gaps within a codon (denoted *internal gaps*) described by a codon alignment of type t. Codon alignments of type 1–3 describe no internal gaps, codon alignments of type 4–7 describe one internal gap and codon alignments of type 8–11 describe two internal gaps.

We introduce $c_p^* : \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}^3 \times \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}^3 \to \mathbf{R}$ for use in the explanation. We define $c_p^*(\sigma_1\sigma_2\sigma_3, \tau_1\tau_2\tau_3)$ as the distance between $\sigma_1\sigma_2\sigma_3$ and $\tau_1\tau_2\tau_3$ in the DNA/protein model. This is the minimum over the cost[3] of the six possible sequences of the three substitutions $\sigma_1 \to \tau_1$, $\sigma_2 \to \tau_2$ and $\sigma_3 \to \tau_3$.

5.5.1 Codon alignments with no internal gaps

The cost $D^1(i, j)$ is the cost of the last codon alignment of type 1 plus the cost of the remaining alignment. The last codon alignment is an alignment of $a_1^i a_2^i a_3^i$ and $b_1^j b_2^j b_3^j$. By definition of c_p^* the cost is $c_p^*(a_1^i a_2^i a_3^i, b_1^j b_2^j b_3^j)$. The cost of the remaining alignment is $D(i-1, j-1)$.

$$D^1(i, j) = D(i-1, j-1) + c_p^*(a_1^i a_2^i a_3^i, b_1^j b_2^j b_3^j) \tag{5.6}$$

A codon alignment of type 2 or type 3 describes a gap between codons. Since the combined gap cost function is affine we can use the technique introduced in [46] saying that a gap ending in (i, j) is either a continuation of an existing gap ending in $(i-1, j)$ or $(i, j-1)$, or the start of a new gap.

$$D^2(i, j) = \min\{D(i, j-1) + \alpha + \beta, D^2(i, j-1) + \beta\} \tag{5.7}$$

$$D^3(i, j) = \min\{D(i-1, j) + \alpha + \beta, D^3(i-1, j) + \beta\} \tag{5.8}$$

5.5.2 Codon alignments with one internal gap

We describe how to compute $D^6(i, j)$. The other cases where the last codon alignment describes one internal gap are handled similarly. The cost $D^6(i, j)$ is the cost of the last codon alignment of type 6 plus the cost of the remaining alignment. The last codon alignment of type 6 describes three substitutions and one deletion. If the deletion has length k (a deletion of $3k$ nucleotides) then the cost of the remaining alignment is $D(i-k-1, j-1)$ and the last codon

[3] We use the term *cost* to denote the DNA level cost plus the protein level cost.

alignment is an alignment of $a_1^{i'}a_2^{i'}a_3^{i'}\cdots a_1^ia_2^ia_3^i$ and $b_1^jb_2^jb_3^j$ where $i' = i - k$. This is illustrated in Fig. 5.8.

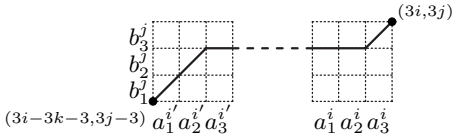


Figure 5.8: The last codon alignment of type 6.

The cost of the last codon alignment is the minimum cost of a sequence of the three substitutions and the deletion. Any sequence of these four events can be divided into three steps: The substitutions occurring before the deletion, the deletion and the substitutions occurring after the deletion. Figure 5.9 illustrates the three steps of the evolution of $a_1^{i'}a_2^{i'}a_3^{i'}\cdots a_1^ia_2^ia_3^i$ to $b_1^jb_2^jb_3^j$. The nucleotides x_1, x_2 and x_3 are the result of the up to three substitutions before the deletion. For example, if the substitution $a_1^{i'} \to b_1^j$ occurs before the deletion, then x_1 is b_1^j, otherwise it is $a_1^{i'}$. We say that $x_1 \in \{a_1^{i'}, b_1^j\}$, $x_2 \in \{a_2^{i'}, b_2^j\}$ and $x_3 \in \{a_3^i, b_3^j\}$ are the status of the three substitutions before the deletion. We observe that $x_1x_2x_3$ is the remaining codon of the deletion.

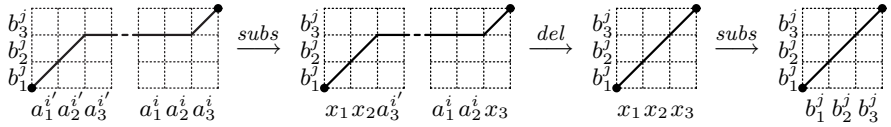


Figure 5.9: The evolution of $a_1^{i'}a_2^{i'}a_3^{i'}\cdots a_1^ia_2^ia_3^i$ to $b_1^jb_2^jb_3^j$ described by the last codon alignment.

The substitutions occurring before the deletion change codon $a_1^{i'}a_2^{i'}a_3^{i'}$ to $x_1x_2a_3^{i'}$ and codon $a_1^ia_2^ia_3^i$ to $a_1^ia_2^ix_3$. The substitutions occurring after the deletion change codon $x_1x_2x_3$ to $b_1^jb_2^jb_3^j$. We recall that the cost of changing codon $\sigma_1\sigma_2\sigma_3$ to codon $\tau_1\tau_2\tau_3$ by a sequence of the substitutions $\sigma_1 \to \tau_1$, $\sigma_2 \to \tau_2$ and $\sigma_3 \to \tau_3$ is $c_p^*(\sigma_1\sigma_2\sigma_3, \tau_1\tau_2\tau_3)$. Since an identical substitution has cost zero then the cost of the three substitutions in the last codon alignment is equal to the cost of the induced codon changes. The cost is

$$cost(subs) = c_p^*(a_1^{i'}a_2^{i'}a_3^{i'}, x_1x_2a_3^{i'}) + c_p^*(a_1^ia_2^ia_3^i, a_1^ia_2^ix_3) + c_p^*(x_1x_2x_3, b_1^jb_2^jb_3^j). \quad (5.9)$$

The cost of the deletion of $3k$ nucleotides in the last codon alignment is the sum of the DNA level cost $g_d(3k)$ and the protein level cost as given by (5.4). By using the combined gap cost function $g(k) = g_d(3k) + g_p(k) = \alpha + \beta k$ and our knowledge of the remaining codon $x_1x_2x_3$ of the deletion, we can formulate this sum as

$$cost(del) = \min \begin{cases} \alpha + \beta k + c_p(a_1^ia_2^ix_3, x_1x_2x_3)^4 \\ \alpha_p + \alpha + \beta k + \min_{0<l<k} c_p(a_1^{i-l}a_2^{i-l}a_3^{i-l}, x_1x_2x_3) \\ \alpha + \beta k + c_p(x_1x_2a_3^{i'}, x_1x_2x_3) \end{cases} . \quad (5.10)$$

The cost of the deletion depends on the deletion length, the remaining codon $x_1x_2x_3$ and a witness. The witness can be the start-codon $x_1x_2a_3^{i'}$, the end-codon $a_1^ia_2^ix_3$ or one of the internal codons $a_1^{i-l}a_2^{i-l}a_3^{i-l}$ for some $0 < l < k$. The witness encodes the amino acid aligned with the remaining amino acid.

We can now compute $D^6(i,j)$ under the assumption of a certain deletion length k and remaining codon $x_1x_2x_3$ of the deletion in the last codon alignment as the sum $cost(subs) + cost(del) + D(i-k-1, j-1)$. We can thus compute $D^6(i,j)$ by minimizing this sum over all possible combinations of deletion length k and remaining codon $x_1x_2x_3$. A combination of deletion length k and remaining codon $x_1x_2x_3$ is possible if $x_1 \in \{a_1^{i'}, b_1^j\}$, $x_2 \in \{a_2^{i'}, b_2^j\}$ and $x_3 \in \{a_3^i, b_3^j\}$ where $i' = i - k$. The terms $c_p^*(a_1^ia_2^ia_3^i, a_1^ia_2^ix_3)$ and $c_p^*(x_1x_2x_3, b_1^jb_2^jb_3^j)$ of $cost(subs)$ do not depend on the deletion length, so we can split the minimization as

$$D^6(i,j) = \min_{x_1x_2x_3}\{c_p^*(a_1^ia_2^ia_3^i, a_1^ia_2^ix_3) + c_p^*(x_1x_2x_3, b_1^jb_2^jb_3^j) + D^6_{x_1x_2x_3}(i,j)\} \quad (5.11)$$

where

$$D^6_{x_1x_2x_3}(i,j) = \min_{0<k<i}\{D(i-k-1, j-1) + c_p^*(a_1^{i-k}a_2^{i-k}a_3^{i-k}, x_1x_2a_3^{i-k}) + cost(del)\} \quad (5.12)$$

is the minimum cost of the terms that depend on both the deletion length and the remaining codon under the assumption that the remaining codon is $x_1x_2x_3$. If we expand the term $cost(del)$ we get

$$D^6_{x_1x_2x_3}(i,j) = \min_{0<k<i}\{\mathrm{len}^6_{x_1x_2}(i,j,k) + \min\begin{cases} c_p(a_1^ia_2^ix_3, x_1x_2x_3) \\ \alpha_p + \min_{0<l<k} c_p(a_1^{i-l}a_2^{i-l}a_3^{i-l}, x_1x_2x_3) \\ c_p(x_1x_2a_3^{i-k}, x_1x_2x_3) \end{cases} \quad (5.13)$$

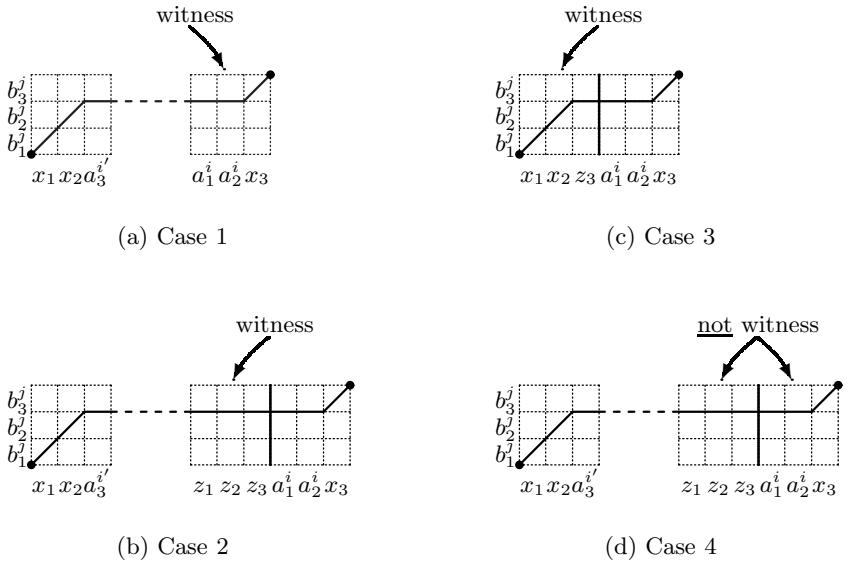
where

$$\mathrm{len}^6_{x_1x_2}(i,j,k) = D(i-k-1, j-1) + c_p^*(a_1^{i-k}a_2^{i-k}a_3^{i-k}, x_1x_2a_3^{i-k}) + \alpha + \beta k \quad (5.14)$$

is the cost of the remaining alignment plus the part of the cost of the last codon alignment that does not depend on the codon $a_1^ia_2^ia_3^i$ and the witness. The cost $\mathrm{len}^6_{x_1x_2}(i,j,k)$ is defined if $x_1 \in \{a_1^{i-k}, b_1^j\}$ and $x_2 \in \{a_2^{i-k}, b_2^j\}$. The cost $D^6_{x_1x_2x_3}(i,j)$ is defined if there exists a deletion length k such that k and $x_1x_2x_3$ is a possible combination of deletion length and remaining codon.

We observe that there are at most 32 possible remaining codons $x_1x_2x_3$. The observation follows because we known that x_3 must be one of the two

[4]We use $c_p(\sigma_1\sigma_2\sigma_3, \tau_1\tau_2\tau_3)$ as a convenient notation for $c_p(\sigma, \tau)$ where σ and τ are the amino acids coded by the codons $\sigma_1\sigma_2\sigma_3$ and $\tau_1\tau_2\tau_3$ respectively.



(a) Case 1

(c) Case 3

(b) Case 2

(d) Case 4

Figure 5.10: The four cases in the computation of $D^6_{x_1x_2x_3}(i,j)$. We use $z_1z_2z_3$ as notation for $a_1^{i-1}a_2^{i-1}a_3^{i-1}$.

known nucleotides a_3^i or b_3^j. If we can compute $D^6_{x_1x_2x_3}(i,j)$ in constant time for each of the possible remaining codons then we can also compute $D^6(i,j)$ in constant time. To compute $D^6_{x_1x_2x_3}(i,j)$ we must determine a combination of witness and deletion length that minimizes the cost. We say that we must determine the witness and deletion length of $D^6_{x_1x_2x_3}(i,j)$.

It is easy to see that witness and deletion lenghth of $D^6_{x_1x_2x_3}(i,j)$ must be one of the four combinations illustrated in Fig. 5.10. We can thus compute $D^6_{x_1x_2x_3}(i,j)$ as the minimum over the cost of the four cases illustrated in Fig. 5.10. The cost of case 1–3 is computed by simplifying (5.13) for a particular witness and deletion length. The cost of case 4 cannot be computed this way because both the witness and the deletion length are unknown.

Case 1. The end-codon is the witness and the deletion length is at least one. The cost is $\min_{0<k<i} \text{len}^6_{x_1x_2}(i,j,k) + c_p(a_1^i a_2^i x_3, x_1x_2x_3)$.

Case 2. The last internal codon is the witness and the deletion length is at least two. The cost is $\min_{1<k<i} \text{len}^6_{x_1x_2}(i,j,k) + \alpha_p + c_p(a_1^{i-1}a_2^{i-1}a_3^{i-1}, x_1x_2x_3)$.

Case 3. The start-codon is the witness and the deletion length is one. The cost is $\text{len}^6_{x_1x_2}(i,j,1) + c_p(x_1x_2a_3^{i-1}, x_1x_2x_3)$.

Case 4. The witness is neither the end-codon nor the last internal codon and the deletion length is at least two. We observe that if the witness of $D^6_{x_1x_2x_3}(i-1,j)$ is not the end-codon $a_1^{i-1}a_2^{i-1}x_3$ then by optimality of $D^6_{x_1x_2x_3}(i-1,j)$ this witness must also be the witness of case 4. If this is the case then the cost of case 4 is $D^6_{x_1x_2x_3}(i-1,j) + \beta$.

The observation in case 4 suggests that we can use dynamic programming to keep track of $D^6_{x_1x_2x_3}(i,j)$ under the assumption that the end-codon is not the witness, i.e. use dynamic programming to keep track of the minimum cost of case 2–4. We introduce tables $F^6_{x_1x_2x_3}$ corresponding to the 64 combinations of $x_1x_2x_3$. We maintain that if $x_1x_2x_3$ is a possible remaining codon and the end-codon $a_1^i a_2^i x_3$ is not the witness of $D^6_{x_1x_2x_3}(i,j)$, then $F^6_{x_1x_2x_3}(i,j)$ is equal to $D^6_{x_1x_2x_3}(i,j)$. If we define $F^6_{x_1x_2x_3}(0,j)$ to infinity, then

$$F^6_{x_1x_2x_3}(i,j) = \min \begin{cases} \text{cost of } \textit{Case 2} \\ \text{cost of } \textit{Case 3} \\ F^6_{x_1x_2x_3}(i-1,j) + \beta \end{cases} \tag{5.15}$$

In order to compute the cost of case 1 and 2 in constant time we maintain the minimum of $\text{len}^6_{x_1x_2}(i,j,k)$ over k by dynamic programming. We introduce tables $L^6_{x_1x_2}$ corresponding to the 16 combinations of x_1x_2 such that $L^6_{x_1x_2}(i,j)$ is equal to $\min_{0<k<i} \text{len}^6_{x_1x_2}(i,j,k)$. If we define $L^6_{x_1x_2}(0,j)$ to infinity, then

$$L^6_{x_1x_2}(i,j) = \min \begin{cases} \text{len}^6_{x_1x_2}(i,j,1) \\ L^6_{x_1x_2}(i-1,j) + \beta \end{cases} \tag{5.16}$$

We can now compute $D^6_{x_1x_2x_3}(i,j)$ in constant time as the minimum cost of case 1–4. The cost of case 1 is $L^6_{x_1x_2}(i,j) + c_p(a_1^i a_2^i x_3, x_1x_2x_3)$ and the minimum cost of case 2–4 is $F^6_{x_1x_2x_3}(i,j)$, so

$$D^6_{x_1x_2x_3}(i,j) = \min \begin{cases} L^6_{x_1x_2}(i,j) + c_p(a_1^i a_2^i x_3, x_1x_2x_3) \\ F^6_{x_1x_2x_3}(i,j) \end{cases} \tag{5.17}$$

The computation of $D^6(i,j)$ by (5.11) requires us to compute $D^6_{x_1x_2x_3}(i,j)$ for each of the 32 possible remaining codons. To do this we must compute entry (i,j) in the 16 tables $L^6_{x_1x_2}$ and entry (i,j) in the 64 tables $F^6_{x_1x_2x_3}$. As explained in this section all this can be done in constant time.

The other three cases where the last codon alignment describes one internal gap (type 4, 5 and 7) are handled similarly. However, if the last codon alignment is of type 4 or 5, then only the first nucleotide x_1 in the remaining codon depends on the deletion (or insertion) length. This limits the number of possible remaining codons to 16 and implies that only four tables are needed to keep track of $\min_{0<k<i} \text{len}^t_{x_1}(i,j,k)$ for $t = 4,5$. Hence, to compute $D^t(i,j)$ for $t = 4,5,6,7$, we compute $2 \cdot 4 + 2 \cdot 16 + 4 \cdot 64 = 296$ table entries in total.

5.5.3 Codon alignments with two internal gaps

We describe how to compute $D^8(i,j)$. The other cases where the last codon alignment describes two internal gaps are handled similarly. The cost $D^8(i,j)$ is the cost of the last codon alignment of type 8 plus the cost of the remaining alignment. The last codon alignment of type 8 describes three substitutions and two deletions. If the first deletion has length k' and the second deletion has length k then the cost of the remaining alignment is $D(i-k-k'-1, j-1)$ and the last codon alignment is an alignment of $a_1^{i''} a_2^{i''} a_3^{i''} \cdots a_1^{i'} a_2^{i'} a_3^{i'} \cdots a_1^i a_2^i a_3^i$ and $b_1^j b_2^j b_3^j$ where $i' = i-k$ and $i'' = i'-k'$. This is illustrated in Fig. 5.11.

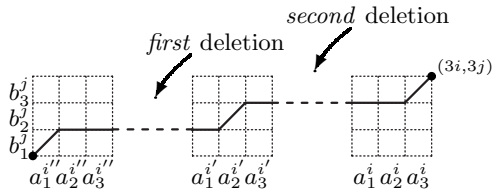


Figure 5.11: The last codon alignment of type 8

We will compute $D^8(i,j)$ as we computed $D^6(i,j)$ by minimizing the cost over all possible combinations of deletion length k and remaining codon $x_1x_2x_3$ of the second deletion. This reduces the problem of computing $D^8(i,j)$ to computing $D^8_{x_1x_2x_3}(i,j)$, the cost under the assumption of a certain remaining codon of the second deletion, for each of the 32 possible remaining codons of the second deletion. We can compute $D^8_{x_1x_2x_3}(i,j)$ similar to the way we computed $D^6_{x_1x_2x_3}(i,j)$. An inspection of (5.15), (5.16) and (5.17) reveals that all we essentially have to do is to replace $\text{len}^6_{x_1x_2}(i,j,1)$ with the corresponding part of $D^8(i,j)$. We denote this part of the cost $\text{len}^8_{x_1x_2x_3}(i,j,1)$.

More precisely, if we assume that the second deletion has length k and remaining codon $x_1x_2x_3$ then $\text{len}^8_{x_1x_2x_3}(i,j,k)$ is the part of $D^8(i,j)$ that does not depend on the codon $a^i_1a^i_2a^i_3$ and the witness of the second deletion. This cost depends on the order of the two deletions in the last codon alignment. Hence, we introduce $\text{len}^{8'}_{x_1x_2x_3}(i,j,k)$ and $\text{len}^{8''}_{x_1x_2x_3}(i,j,k)$ to denote the cost when the first deletion occurs before the second deletion and vice versa. We define $\text{len}^8_{x_1x_2x_3}(i,j,k)$ as $\min\{\text{len}^{8'}_{x_1x_2x_3}(i,j,k), \text{len}^{8''}_{x_1x_2x_3}(i,j,k)\}$. Since we only have to compute $\text{len}^8_{x_1x_2x_3}(i,j,1)$ we can restrict ourselves to the case where the second deletion has length one and the first deletion has length k'. In the rest of this section we use the notation that $i' = i-1$ and $i'' = i'-k'$. We split into two cases depending on the order of the first and second deletion.

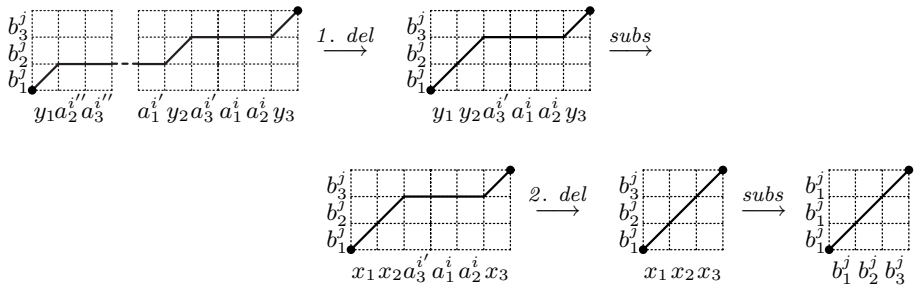


Figure 5.12: The first deletion occurs before the second deletion and the second deletion has length one.

Figure 5.12 illustrates the evolution of the last codon alignment (of type 8) when the second deletion has length one and occurs after the first deletion. The nucleotides y_1, y_2 and y_3 are the status of the substitutions before the first deletion and the nucleotides x_1, x_2 and x_3 are the status of the substitutions before

the second deletion. Similar to (5.9) we compute the cost of the three substitutions in the last codon alignment as the cost of the induced codon changes. An inspection of Fig. 5.12 reveals that the cost of the three substitutions is

$$\begin{aligned} cost(subs) = c_p^*(a_1^{i''}a_2^{i''}a_3^{i''}, y_1a_2^{i''}a_3^{i''}) + c_p^*(a_1^{i'}a_2^{i'}a_3^{i'}, a_1^{i'}y_2a_3^{i'}) + \\ c_p^*(a_1^ia_2^ia_3^i, a_1^ia_2^iy_3) + c_p^*(y_1y_2a_3^{i'}, x_1x_2a_3^{i'}) + \\ c_p^*(a_1^ia_2^iy_3, a_1^ia_2^ix_3) + c_p^*(x_1x_2x_3, b_1^jb_2^jb_3^j). \end{aligned} \quad (5.18)$$

We can compute the cost of the two deletions similar to (5.10). Remember that the first deletion has length k' and the second deletion has length one and that we use the notation $i' = i - 1$ and $i'' = i' - k'$.

$$cost(del_1) = \min \begin{cases} \alpha + \beta k' + c_p(a_1^{i'}y_2a_3^{i'}, y_1y_2a_3^{i'}) \\ \alpha_p + \alpha + \beta k' + \min_{0<l<k'} c_p(a_1^{i'-l}a_2^{i'-l}a_3^{i'-l}, y_1y_2a_3^{i'}) \\ \alpha + \beta k' + c_p(y_1a_2^{i''}a_3^{i''}, y_1y_2a_3^{i'}) \end{cases} \quad (5.19)$$

$$cost(del_2) = \alpha + \beta + \min \begin{cases} c_p(x_1x_2a_3^{i'}, x_1x_2x_3) \\ c_p(a_1^ia_2^ix_3, x_1x_2x_3) \end{cases} \quad (5.20)$$

If we assume that the first deletion occurs before the second deletion and that the second deletion has length one and remaining codon $x_1x_2x_3$ then $D^8(i,j)$ is given by the sum $cost(subs) + cost(del_1) + cost(del_2) + D(i' - k' - 1, j - 1)$ minimized over all possible combinations of codon $y_1y_2y_3$ and length k'. Remember that $\text{len}^{8'}_{x_1x_2x_3}(i,j,1)$ is the part of this minimum that does not depend on $a_1^ia_2^ia_3^i$ or the witness of the second deletion. By inspection of the above expressions we observe that $\text{len}^{8'}_{x_1x_2x_3}(i,j,1)$ includes everything but the terms $c_p^*(a_1^ia_2^ia_3^i, a_1^ia_2^iy_3)$ and $c_p^*(a_1^ia_2^iy_3, a_1^ia_2^ix_3)$ of $cost(subs)$ and everything but the term $\min\{c_p(x_1x_2a_3^{i'}, x_1x_2x_3), c_p(a_1^ia_2^ix_3, x_1x_2x_3)\}$ of $cost(del_2)$. It is easy to verify that $D^4_{y_1y_2a_3^{i'}}(i',j)$ is equal to the sum $D(i' - k' - 1, j - 1) + c_p^*(a_1^{i''}a_2^{i''}a_3^{i''}, y_1a_2^{i''}a_3^{i''}) + cost(del_1)$ minimized over the deletion length k' of the first deletion. This observation makes it possible to compute $\text{len}^{8'}_{x_1x_2x_3}(i,j,1)$ as

$$\begin{aligned} \text{len}^{8'}_{x_1x_2x_3}(i,j,1) = \alpha + \beta + c_p^*(x_1x_2x_3, b_1^jb_2^jb_3^j) + \\ \min_{y_1y_2}\{c_p^*(a_1^{i'}a_2^{i'}a_3^{i'}, a_1^{i'}y_2a_3^{i'}) + D^4_{y_1y_2a_3^{i'}}(i',j) + c_p^*(y_1y_2a_3^{i'}, x_1x_2a_3^{i'})\} \end{aligned} \quad (5.21)$$

where we minimize over $y_1 \in \{a_1^{i''}, x_1\}$ and $y_2 \in \{a_2^{i'}, x_2\}$. The cost $\text{len}^{8'}_{x_1x_2x_3}(i,j,1)$ is defined if $x_1x_2x_3$ allows the second deletion to have length one, i.e. if $x_1 \in \{a_1^{i''}, b_1^j\}$, $x_2 \in \{a_2^{i'}, b_2^j\}$ and $x_3 \in \{a_3^i, b_3^j\}$. The nucleotide $a_1^{i''}$ depends on the unknown length of the first deletion, so we must assume that it can be any of the four nucleotides.

Figure 5.13 illustrates the evolution of the last codon alignment when the second deletion has length one and occurs before the first deletion. The nucleotides z_1, x_2 and x_3 are the status of the substitutions before the second deletion and the nucleotides y_1, y_2 and y_3 are the status of the substitutions

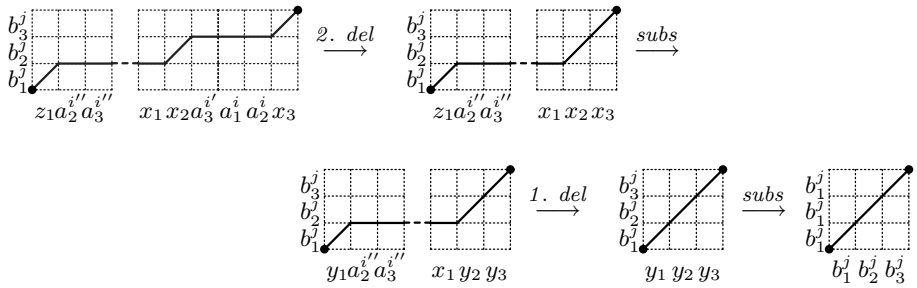


Figure 5.13: The second deletion occurs before the first deletion and the second deletion has length one.

before the first deletion. Observe that x_1 is just $a_1^{i'}$. The cost of this case can be described as above. This would reveal that $\text{len}^{8''}_{x_1x_2x_3}(i,j,1)$ includes everything but $c_p^*(a_1^ia_2^ia_3^i, a_1^ia_2^ix_3) + c_p^*(a_1^ia_2^ix_3, a_1^ia_2^iy_3) + c_p(x_1x_2x_3, w_1w_2w_3)$ where $w_1w_2w_3$ is the witness of the second deletion. Furthermore it would reveal that $D^4_{y_1y_2y_3}(i',j)$ is equal to the sum of the cost of the remaining alignment, the cost of the first deletion and the cost of changing codon $a_1^{i''}a_2^{i''}a_3^{i''}$ to $z_1a_2^{i''}a_3^{i''}$ to $y_1a_2^{i''}a_3^{i''}$ minimized over the deletion length k' of the first deletion. This makes it possible to compute $\text{len}^{8''}_{x_1x_2x_3}(i,j,1)$ as

$$\begin{aligned}\text{len}^{8''}_{x_1x_2x_3}(i,j,1) &= \alpha + \beta + c_p^*(a_1^{i'}a_2^{i'}a_3^{i'}, x_1x_2a_3^{i'}) + \\ &\min_{y_1y_2y_3}\{c_p^*(a_1^{i'}x_2x_3, a_1^{i'}y_2y_3) + D^4_{y_1y_2y_3}(i',j) + c_p^*(y_1y_2y_3, b_1^jb_2^jb_3^j)\} \quad (5.22)\end{aligned}$$

where we minimize over $y_1 \in \{z_1, b_1^j\}$, $y_2 \in \{x_2, b_2^j\}$ and $y_3 \in \{x_3, b_3^j\}$. The nucleotide z_1 depends on the unknown length of the first deletion, so we must assume that z_1 can be any of the four nucleotides. The cost $\text{len}^{8''}_{x_1x_2x_3}(i,j,1)$ is defined if $x_1x_2x_3$ allows the second deletion to have length one, i.e. if $x_1 = a_1^{i'}$, $x_2 \in \{a_2^{i'}, b_2^j\}$ and $x_3 \in \{a_3^i, b_3^j\}$.

We are finally in a position where we can describe how to use the method from the previous section to compute $D^8(i,j)$. The cost $\text{len}^8_{x_1x_2x_3}(i,j,k)$ depends on x_1, x_2 and x_3, so instead of 16 tables we need 64 tables $L^8_{x_1x_2x_3}$ to keep track of $\min_{0<k<i}\text{len}^8_{x_1x_2x_3}(i,j,k)$. We still need 64 tables $F^8_{x_1x_2x_3}$ to keep track of the cost under the assumption that the end-codon $a_1^ia_2^ix_3$ is not the witness (of the second deletion). We compute table entry (i,j) in these tables as

$$L^8_{x_1x_2x_3}(i,j) = \min\begin{cases}\text{len}^8_{x_1x_2x_3}(i,j,1) \\ L^8_{x_1x_2x_3}(i-1,j) + \beta\end{cases} \quad (5.23)$$

$$F^8_{x_1x_2x_3}(i,j) = \min\begin{cases}L^8_{x_1x_2x_3}(i-1,j) + \beta + \alpha_p + c_p(a_1^{i-1}a_2^{i-1}a_3^{i-1}, x_1x_2x_3) \\ \text{len}^8_{x_1x_2x_3}(i,j,1) + c_p(x_1x_2a_3^{i-1}, x_1x_2x_3) \\ F^8_{x_1x_2x_3}(i-1,j) + \beta\end{cases} \quad (5.24)$$

We compute $D^8_{x_1x_2x_3}(i,j)$ using the above tables and we compute $D^8(i,j)$ by minimizing over the 32 possible remaining codons of the second deletion.

$$D^8_{x_1x_2x_3}(i,j) = \min \begin{cases} L^8_{x_1x_2x_3}(i,j) + c_p(a_1^i a_2^i x_3, x_1x_2x_3) \\ F^8_{x_1x_2x_3}(i,j) \end{cases} \tag{5.25}$$

$$D^8(i,j) = \min_{x_1x_2x_3} \{c_p^*(a_1^i a_2^i a_3^i, a_1^i a_2^i x_3) + D^8_{x_1x_2x_3}(i,j)\} \tag{5.26}$$

This constant time computation of $D^8(i,j)$ requires us to compute entry (i,j) in 128 tables. The other three cases where the last codon alignment describes two internal gaps are handled similarly. Hence, to compute $D^t(i,j)$ for $t = 8,9,10,11$ we compute $4 \cdot 128 = 512$ table entries in total.

5.5.4 Combining the computation

We observe that the only real difference between the computation of $D^6(i,j)$ and $D^8(i,j)$ is between $\text{len}^6_{x_1x_2}(i,j,1)$ and $\text{len}^8_{x_1x_2x_3}(i,j,1)$. The similarity stems from the fact that a codon alignment of type 6 and type 8 ends in the same way. By "end in the same way" we mean that the events described on the codon $a_1^i a_2^i a_3^i$ are the same. Figure 5.7 reveals that a codon alignment of type 11 also ends in the same way as codon alignments of type 6 and 8.

The similarity between the computation of $D^t(i,j)$ for $t = 6,8,11$ makes it possible to combine the computation of the three costs and thereby reduce the number of tables. We can replace the three tables $L^6_{x_1x_2}$, $L^8_{x_1x_2x_3}$ and $L^{11}_{x_1x_2x_3}$ with one table $L^{6,8,11}_{x_1x_2x_3}$ where $L^{6,8,11}_{x_1x_2x_3}(i,j)$ is the minimum of entry (i,j) in the three tables it replaces. Similarly we can replace $F^6_{x_1x_2x_3}$, $F^8_{x_1x_2x_3}$ and $F^{11}_{x_1x_2x_3}$ with $F^{6,8,11}_{x_1x_2x_3}$. We can compute $L^{6,8,11}_{x_1x_2x_3}(i,j)$ and $F^{6,8,11}_{x_1x_2x_3}(i,j)$ by expressions similar to (5.23) and (5.24). All we essentially have to do is to replace $\text{len}^8_{x_1x_2x_3}(i,j,1)$ by

$$\text{len}^{6,8,11}_{x_1x_2x_3}(i,j,1) = \min_{t=6,8,11} \text{len}^t_{x_1x_2x_3}(i,j,1) \tag{5.27}$$

where we in order to ensure that $\text{len}^t_{x_1x_2x_3}(i,j,1)$ for $t = 6,8,11$ describes the same part of the total cost must redefine $\text{len}^6_{x_1x_2x_3}(i,j,1)$ as $\text{len}^6_{x_1x_2}(i,j,1) + c_p^*(x_1x_2x_3, b_1^j b_2^j b_3^j)$.

We introduce $D^{6,8,11}(i,j)$ as the minimum of $D^t(i,j)$ over $t = 6,8,11$. We can compute $D^{6,8,11}(i,j)$ by using $L^{6,8,11}_{x_1x_2x_3}$ and $F^{6,8,11}_{x_1x_2x_3}$ in expressions similar to (5.25) and (5.26). The computation of $D^{6,8,11}(i,j)$ requires us to compute only $64+64 = 128$ table entries while the individual computation of $D^t(i,j)$ for $t = 6,8,10$ requires us to compute $80+128+128 = 336$ table entries. Figure 5.7 also reveals that codon alignments of type 7, 9 and 10 end in the same way. Hence, we can also combine the computation of $D^t(i,j)$ for $t = 7,9,10$.

Finally, to compute $D(i,j)$ by (5.5) we must minimize over $D^1(i,j)$, $D^2(i,j)$, $D^3(i,j)$, $D^4(i,j)$, $D^5(i,j)$, $D^{6,8,11}(i,j)$ and $D^{7,9,10}(i,j)$. In total this computation requires us to compute $1+7+68+68+128+128 = 400$ table entries.

5.6 Future work

We are working on implementing the alignment algorithm described in the previous section in order to compare it to the heuristic alignment algorithm described in [61]. The heuristic algorithm allows frame shifts, so an obvious extension of our exact algorithm would be to allow frame shifts, e.g. to allow insertion-deletions of arbitrary length. This however makes it difficult to split the evaluation of the alignment cost into small independent subproblems (codon alignments) of known size.

Another interesting extension would be to annotate the DNA sequence with more information. For example, if the DNA sequence codes in more than one reading frame (overlapping reading frames) then the DNA sequence should be annotated with all the amino acid sequences encoded and the combined cost of a nucleotide event should summarize the cost of changes induced on all the amino acid sequences encoded by the DNA sequence. This extension also makes it difficult to split the evaluation of the alignment cost into small independent subproblems. To implement these extensions efficiently it might be fruitful to investigate reasonable restrictions of the cost functions.

Chapter 6

Measures on hidden Markov models

> a day will come when I
> shall take the hidden paths that run
> west of the moon, east of the sun.
>
> —J. R. R. Tolkien, *Lord of the Rings*

This paper describes methods to compute metrics and similarity measures comparing hidden Markov models. The results were presented at the Seventh International Conference on Intelligent Systems for Molecular Biology and a short version of the paper, not describing the methods for handling models that are not of the left/right type, is published in the proceedings of this conference [92]. Furthermore, the paper has been published in the BRICS report series [91]. The version included here has been slightly modified compared to the version in the original dissertation presented to the Faculty of Science, University of Aarhus. More precisely, the original illustration of the transition structure of a profile hidden Markov model in figure 6.1 has been replaced with the profile hidden Markov model also shown in figure 2.6(d). And section 6.5.1 has been rewritten as the original version contained some errors. The method for left/right models has been implemented and source code is available at `http://www.brics.dk/~cstorm/hmmcomp/index.html`.

Measures on hidden Markov models

Rune B. Lyngsø* Christian N. S. Pedersen† Henrik Nielsen‡

Abstract

Hidden Markov models were introduced in the beginning of the 1970's as a tool in speech recognition. During the last decade they have been found useful in addressing problems in computational biology such as characterising sequence families, gene finding, structure prediction and phylogenetic analysis. In this paper we propose several measures between hidden Markov models. We give an efficient algorithm that computes the measures for left-right models, e.g. profile hidden Markov models, and discuss how to extend the algorithm to other types of models. We present an experiment using the measures to compare hidden Markov models for three classes of signal peptides.

6.1 Introduction

A hidden Markov model describes a probability distribution over a potentially infinite set of sequences. It is convenient to think of a hidden Markov model as generating a sequence according to some probability distribution by following a first order Markov chain of states, called the path, from a specific start-state to a specific end-state and emitting a symbol according to some probability distribution each time a state is entered. One strength of hidden Markov models is the ability efficiently to compute the probability of a given sequence as well as the most probable path that generates a given sequence. Hidden Markov models were introduced in the beginning of the 1970's as a tool in speech recognition. In speech recognition the set of sequences might correspond to digitised sequences of human speech and the most likely path for a given sequence is the corresponding sequence of words. Rabiner [124] gives a good introduction to the theory of hidden Markov models and their applications to speech recognition.

Hidden Markov models were introduced in computational biology in 1989 by Churchill [27]. Durbin et al. [35] and Eddy [36, 37] are good overviews of the use of hidden Markov models in computational biology. One of the most popular applications is to use them to characterise sequence families by using so called profile hidden Markov models introduced by Krogh et al. [80]. For a profile hidden Markov model the probability of a given sequence indicates how

*Department of Computer Science, University of Aarhus, Denmark. E-mail: rlyngsoe@daimi.au.dk. Work done in part while visiting the Institute for Biomedical Computing at Washington University, St. Louis.

†Basic Research In Computer Science, Centre of the Danish National Research Foundation, University of Aarhus, Denmark. E-mail: cstorm@brics.dk.

‡Center for Biological Sequence Analysis, Centre of the Danish National Research Foundation, Technical University of Denmark, Denmark. E-mail: hnielsen@cbs.dtu.dk

likely it is that the sequence is a member of the modelled sequence family, and the most likely path for a given sequence corresponds to an alignment of the sequence against the modelled sequence family.

An important advance in the use of hidden Markov models in computational biology within the last two years, is the fact that several large libraries of profile hidden Markov models have become available [37]. These libraries not only make it possible to classify new sequences, but also open up the possibility of comparing sequence families by comparing the profiles of the families instead of comparing the individual members of the families, or of comparing entire sequence families instead of the individual members of the family to a hidden Markov model constructed to model a particular feature. To our knowledge little work has been published in this area, except for alignment of profiles [47].

In this paper we propose measures for hidden Markov models that can be used to address this problem. The measures are based on what we call the co-emission probability of two hidden Markov models. We present an efficient algorithm that computes the measures for profile hidden Markov models and observe that the left-right architecture is the only special property of profile hidden Markov models required by the algorithm. We describe how to extend the algorithm to broader classes of models and how to approximate the measures for general hidden Markov models. The method can easily be adapted to various special cases, e.g. if it is required that paths pass through certain states.

As the algorithm we present is not limited to profile hidden Markov models, we have chosen to emphasise this generality by presenting an application to a set of hidden Markov models for signal peptides. These models do not strictly follow the profile architecture and consequently cannot be compared using profile alignment [47].

The rest of the paper is organised as follows. In section 6.2 we discuss hidden Markov models in more detail. In section 6.3 we introduce the co-emission probability of two hidden Markov models and formulate an algorithm for computing this probability of two profile hidden Markov models. In section 6.4 we use the co-emission probability to formulate several measures between hidden Markov models. In section 6.5 we discuss extensions to more general models. In section 6.6 we present an experiment using the method to compare three classes of signal peptides. Finally in section 6.7 we briefly discuss how to compute relaxed versions of the co-emission probability.

6.2 Hidden Markov models

Let M be a hidden Markov model that generates sequences over some finite alphabet Σ with probability distribution P_M, i.e. $P_M(s)$ denotes the probability of $s \in \Sigma^*$ under model M. Like a classical Markov model, a hidden Markov model consists of a set of interconnected states. We use $P_q(q')$ to denote the probability of a transition from state q to state q'. These probabilities are usually called *state transition probabilities*. The transition structure of a hidden Markov model is often shown as a directed graph with a node for each state, and an edge between two nodes if the corresponding state transition probability

is non-zero. Figure 6.1 shows an example of a transition structure. Unlike a classical Markov model, a state in a hidden Markov model can generate or emit a symbol according to a local probability distribution over all possible symbols. We use $P_q(\sigma)$ to denote the probability of generating or emitting symbol $\sigma \in \Sigma$ in state q. These probabilities are usually called *symbol emission probabilities*. If a state does not have symbol emission probabilities we say that the state is a silent state.

It is often convenient to think of a hidden Markov model as a generative model, in which a run generates or emits a sequence $s \in \Sigma^*$ with probability $P_M(s)$. A run of a hidden Markov model begins in a special start-state and continues from state to state according to the state transition probabilities until a special end-state is reached. Each time a non-silent state is entered, a symbol is emitted according to the symbol emission probabilities of the state. A run thus results in a Markovian sequence of states as well as a generated sequence of symbols. The name "hidden Markov model" comes from the fact that the Markovian sequence of states, also called the path, is hidden, while only the generated sequence of symbols is observable.

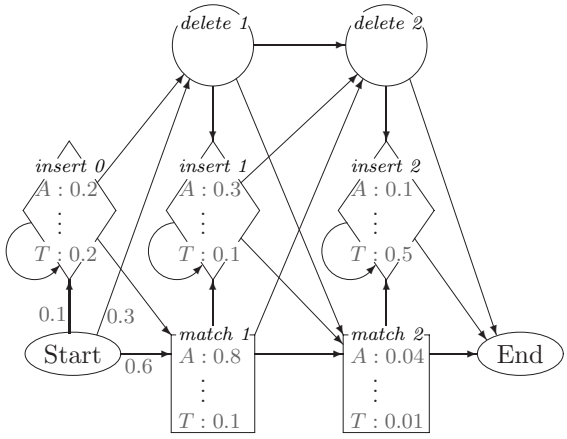


Figure 6.1: The structure of a profile hidden Markov model. The squares are the match-states, the diamonds are the insert-states and the circles are the silent delete-states.

Hidden Markov models have found applications in many areas of computational biology, e.g. gene finding [79] and protein structure prediction [134], but probably the most popular use is as *profiles* for sequence families. A profile is a position-dependent scoring scheme that captures the characteristics of a sequence family, in the sense that the score peaks around members of the family. Profiles are useful when searching for unknown members of a sequence family and several methods have been used to construct and use profiles [48, 87, 138]. Krogh et al. [80] realized that simple hidden Markov models, which they called profile hidden Markov models, were able to capture all other profile methods.

The states of a profile hidden Markov model are divided into match-, insert- and delete-states. Figure 6.1 illustrates the transition structure of a simple profile hidden Markov model. Note the highly repetitive transition structure.

Each of the repeated elements consisting of a match-, insert- and delete-state models a position in the consensus sequence for the sequence family. The silent delete-state makes it possible to skip a position while the self-loop on the insert-state makes it possible to insert one or more symbols between two positions. Another distinctive feature of the structure of profile hidden Markov models is the absence of cycles, except for the self-loops on the insert-states. Hidden Markov models with this property are generally referred to as left-right [73] (or sometimes Bakis [13]) models, as they can be drawn such that all transitions go from left to right.

The state transition and symbol emission probabilities of a profile hidden Markov model (the parameters of the model) should be such that $P_M(s)$ is significant if s is a member of the sequence family. These probabilities can be derived from a multiple alignment of the sequence family, but more importantly, several methods exist to estimate them (or train the model) if a multiple alignment is not available [14, 35, 37].

6.3 Co-emission probability of two models

When using a profile hidden Markov model, it is sometimes sufficient just to focus on the most probable path through the model, e.g. when using a profile hidden Markov model to generate alignments. It is, however, well known that profile hidden Markov models possess a lot more information than the most probable paths, as they allow the generation of an infinity of sequences, each by a multitude of paths. Thus, when comparing two profile hidden Markov models, one should look at the entire spectrum of sequences and probabilities.

In this section we will describe how to compute the probability that two profile hidden Markov models independently generate the same sequence, that is for models M_1 and M_2 generating sequences over an alphabet Σ we compute

$$\sum_{s \in \Sigma^*} P_{M_1}(s) P_{M_2}(s). \tag{6.1}$$

We will call this the *co-emission probability* of the two models. It is also often called the *collision probability* of two probability distributions, as it is the probability that a pair of elements drawn at random from the two distributions 'collide', i.e. are identical. The algorithm we present to compute the co-emission probability is a dynamic programming algorithm similar to the algorithm for computing the probability that a hidden Markov model will generate a specific sequence [35, Chapter 3]. We will describe how to handle the extra complications arising when exchanging the sequence with a profile hidden Markov model.

When computing the probability that a hidden Markov model M generates a sequence $s = s_1 \dots s_n$, a table indexed by a state from M and an index from s is usually built. An entry (q, i) in this table holds the probability of being in the state q in M and having generated the prefix $s_1 \dots s_i$ of s. We will use a similar approach to compute the co-emission probability. Given two hidden Markov models M_1 and M_2, we will describe how to build a table A indexed by states

from the two hidden Markov models, such that the entry $A(q, q')$ – where q is a state of M_1 and q' is a state of M_2 – holds the probability of being in state q in M_1 and q' in M_2 and having independently generated identical sequences on the paths to q and q'. The entry indexed by the two end-states will then hold the probability of being in the end-states and having generated identical sequences, that is the co-emission probability.

To build the table, A, we have to specify how to fill out all entries of A. For a specific entry $A(q, q')$ this depends on the types of states q and q'. As explained in the previous section, a profile hidden Markov model has three types of states (insert-, match- and delete-states) and two special states (start and end). We postpone the treatment of the special states until we have described how to handle the other types of states. For reasons of succinctness we will treat insert- and match-states as special cases of a more general type, which we will call a *generate*-state; this type encompasses all non-silent states of the profile hidden Markov models.

The generate-state will be a merging of match- and insert-states, thus both allowing a transition to itself and having a transition from the previous insert-state; a match-state can be viewed as a generate-state with probability zero of choosing the transition to itself, and an insert-state can be viewed as a generate-state with probability zero of choosing the transition from the previous insert-state. Note that this merging of match- and insert-states is only conceptual; we do not physically merge any states, but just handle the two types of states in a uniform way. This leaves two types of states and thus four different pairs of types. This number can be reduced to three, by observing that the two cases of a generate/delete-pair are symmetric, and thus can be handled the same way.

The rationale behind the algorithm is to split paths up in the last transition(s)[1] and all that preceded this. We will thus need to be able to refer to the states with transitions to q and q'. In the following, m, i and d will refer to the match-, insert- and delete-state with a transition to q, and m', i' and d' to those with a transition to q'. Observe that if q (or q') is an insert-state, then i (or i') is the *previous* insert-state which, by the generate-state generalisation, has a transition to q (or q') with probability zero.

delete/delete entry Assume that q and q' are both delete-states. As these states don't emit symbols, we just have to sum over all possible combinations of immediate predecessors of q and q', of the probability of being in these states and having independently generated identical sequences, multiplied by the joint probability of independently choosing the transitions to q and q'. For the calculation of $A(q, q')$ we thus get the equation

$$\begin{aligned} A(q,q') = & \\ & A(m,m')P_m(q)P_{m'}(q') + A(m,i')P_m(q)P_{i'}(q') + A(m,d')P_m(q)P_{d'}(q') \\ & + A(i,m')P_i(q)P_{m'}(q') + A(i,i')P_i(q)P_{i'}(q') + A(i,d')P_i(q)P_{d'}(q') \\ & + A(d,m')P_d(q)P_{m'}(q') + A(d,i')P_d(q)P_{i'}(q') + A(d,d')P_d(q)P_{d'}(q'). \end{aligned} \tag{6.2}$$

[1] In some of the cases explained below, we will only extend the path in one of the models with an extra transition, hence the unspecificity.

delete/generate entry Assume that q is a delete-state and q' is a generate-state. Envision paths leading to q and q' respectively while independently generating the same sequence. As q does not emit symbols while q' does, the path to q's immediate predecessor (that is, the path to q with the actual transition to q removed) must also have generated the same sequence as the path to q'. We thus have to sum over all immediate predecessors of q, of the probability of being in this state and in q' and having generated identical sequences, multiplied by the probability of choosing the transition to q. For the calculation of $A(q, q')$ in this case we thus get the following equation

$$A(q, q') = A(m, q')P_m(q) + A(i, q')P_i(q) + A(d, q')P_d(q). \tag{6.3}$$

generate/generate entry Assume that q and q' are both generate-states. The last character in sequences generated on the paths to q and q' are generated by q and q' respectively. We will denote the probability that these two states independently generate the same symbol by p, and it is an easy observation that

$$p = \sum_{\sigma \in \Sigma} P_q(\sigma)P_{q'}(\sigma). \tag{6.4}$$

The problem with generate/generate entries is that the last transitions on paths to q and q' might actually come from q and q' themselves, due to the self-loops of generate states. It thus seems that we need $A(q, q')$ to be able to compute $A(q, q')$!

So let us start out by assuming that at most one of the paths to q and q' has a self-loop transition as the last transition. Then we can easily compute the probability of being in q and q' and having independently generated the same sequence on the paths to q and q', by summing over all combinations of states with transitions to q and q' (including combinations with either q or q' but not both) the probabilities of these combinations, multiplied by p (for independently generating the same symbol at q and q') and the joint probability of independently choosing the transitions to q and q'. We denote this probability by $A_0(q, q')$, and by the above argument the equation for computing it is

$$\begin{aligned} A_0(q, q') =& p(A(m, m')P_m(q)P_{m'}(q') + A(m, i')P_m(q)P_{i'}(q') \\ &+ A(m, d')P_m(q)P_{d'}(q') + A(m, q')P_m(q)P_{q'}(q') \\ &+ A(i, m')P_i(q)P_{m'}(q') + A(i, i')P_i(q)P_{i'}(q') \\ &+ A(i, d')P_i(q)P_{d'}(q') + A(i, q')P_i(q)P_{q'}(q') \\ &+ A(d, m')P_d(q)P_{m'}(q') + A(d, i')P_d(q)P_{i'}(q') \\ &+ A(d, d')P_d(q)P_{d'}(q') + A(d, q')P_d(q)P_{q'}(q') \\ &+ A(q, m')P_q(q)P_{m'}(q') + A(q, i')P_q(q)P_{i'}(q') \\ &+ A(q, d')P_q(q)P_{d'}(q')). \end{aligned} \tag{6.5}$$

Now let us cautiously proceed, by considering a pair of paths where one of the paths has exactly one self-loop transition in the end, and the other path has at least one self-loop transition in the end. The probability – that we surprisingly call $A_1(q, q')$ – of getting to q and q' along such paths while generating the

same sequences is the probability of getting to q and q' along paths that do not both have a self-loop transition in the end, multiplied by the joint probability of independently choosing the self-loop transitions, and the probability of q and q' emitting the same symbols. But this is just

$$A_1(q, q') = rA_0(q, q'), \tag{6.6}$$

where

$$r = pP_q(q)P_{q'}(q') \tag{6.7}$$

is the probability of independently choosing the self-loop transitions and emitting the same symbols in q and q'. Similarly we can define $A_k(q, q')$, and by induction it is easily proven that

$$A_k(q, q') \;=\; rA_{k-1}(q, q') \;=\; r^k A_0(q, q'). \tag{6.8}$$

As any finite path ending in q or q' must have a finite number of self-loop transitions in the end, we get

$$\begin{aligned} A(q, q') \;&=\; \sum_{k=0}^{\infty} A_k(q, q') \\ &= \sum_{k=0}^{\infty} r^k A_0(q, q') \\ &= \frac{1}{1-r} A_0(q, q'). \end{aligned} \tag{6.9}$$

Despite the fact that there is an infinite number of cases to consider, we observe that the sum over the probabilities of all these cases comes out as a geometric series that can easily be computed.

Based on equations 6.2, 6.3, 6.5 and 6.9 we can compute each of the entries of A pertaining to match- insert- and delete-states in constant time. As for the start-states (denoted by s and s') we initialise $A(s, s')$ to 1 (as we have not started generating anything and the empty sequence is identical to itself). Otherwise, even though they do not generate any symbols, we will treat the start-states as generate states; this allows for choosing an initial sequence of delete-states in one of the models. The start-states are the only possible immediate predecessors for the first insert-states, and together with the first insert-states the only immediate predecessors of the first match- and delete-states; the equations for the entries indexed by any of these states can trivially be modified according to this. The end-states (denoted by e and e') do not emit any symbols and are thus akin to delete-states, and can be treated the same way.

The co-emission probability of M_1 and M_2 is the probability of being in the states e and e' and having independently generated the same sequences. This probability can be found by looking up $A(e, e')$. In the rest of this paper we will use $A(M_1, M_2)$ to denote the co-emission probability of M_1 and M_2.

As all entries of A can be computed in constant time, we can compute the co-emission probability of M_1 and M_2 in time $\mathrm{O}(n_1 n_2)$ where n_i denotes the number of states in M_i. The straightforward space requirement is also $\mathrm{O}(n_1 n_2)$ but can be reduced to $\mathrm{O}(n_1)$ by a standard trick [52, Chapter 11].

6.4 Measures on hidden Markov Models

Based on the co-emission probability we define two metrics that hopefully, to some extent, express how similar the families of sequences represented by two hidden Markov models are. A problem with the co-emission probability is that the models having the largest co-emission probability with a specific model, M, usually will not include M itself, as shown by the following proposition.

Proposition 2 *Let M be a hidden Markov model and $p = \max\{P_M(s) \mid s \in \Sigma^*\}$. The maximum co-emission probability with M attainable for any hidden Markov model is p. Furthermore, the hidden Markov models attaining this co-emission probability with M, are exactly those models, M', for which $P_{M'}(s) > 0 \Leftrightarrow P_M(s) = p$ for all $s \in \Sigma^*$.*

Proof. Let M' be a hidden Markov model with $P_{M'}(s) > 0 \Leftrightarrow P_M(s) = p$. Then

$$\sum_{s\in\Sigma^*, P_M(s)=p} P_{M'}(s) = 1 \tag{6.10}$$

and thus the co-emission probability of M and M' is

$$\sum_{s\in\Sigma^*} P_M(s)P_{M'}(s) \;=\; \sum_{s\in\Sigma^*, P_M(s)=p} P_M(s)P_{M'}(s) \;=\; p. \tag{6.11}$$

Now let M' be a hidden Markov model with $P_{M'}(s') = p' > 0$ for some $s' \in \Sigma^*$ with $P_M(s') = p'' < p$. Then the co-emission probability of M and M' is

$$\begin{aligned} \sum_{s\in\Sigma^*} P_M(s)P_{M'}(s) &= p'p'' + \sum_{s\in\Sigma^*\setminus\{s'\}} P_M(s)P_{M'}(s) \\ &\leq p'p'' + (1-p')p \\ &< p. \end{aligned} \tag{6.12}$$

This proves that a hidden Markov model, M', has maximum co-emission probability, p, with M, if and only if the assertion of the proposition is fulfilled. □

Proposition 2 indicates that the co-emission probability of two models not only depends on how alike they are, but also on how 'self-confident' the models are, that is, to what extent the probabilities are concentrated to a small subset of all possible sequences.

Another way to explain this undesirable property of the co-emission probability, is to interpret hidden Markov models – or rather the probability distribution over finite sequences of hidden Markov models – as vectors in the infinite dimensional space spanned by all finite sequences over the alphabet. With this interpretation the co-emission probability, $A(M_1, M_2)$, of two hidden Markov models, M_1 and M_2, simply becomes the inner product,

$$\langle M_1, M_2\rangle = |M_1||M_2|\cos v, \tag{6.13}$$

of the models. In the expression on the right hand side, v is the angle between the models – or vectors – and $|M_i| = \sqrt{\langle M_i, M_i \rangle}$ is the length of M_i. One observes the direct proportionality between the co-emission probability and the length (or 'self-confidence') of the models being compared. If the length is to be completely ignored, a good measure of the distance between two hidden Markov models would be the angle between them – two models are orthogonal, if and only if they can not generate identical sequences, and parallel (actually identical as the probabilities have to sum to 1) if they express the same probability distribution. This leads to the definition of our first metric on hidden Markov models.

Definition 9 *Let M_1 and M_2 be two hidden Markov models, and let $A(M, M')$ denote the co-emission probability of two hidden Markov models M and M'. We define the* angle *between M_1 and M_2 as*

$$D_{angle}(M_1, M_2) = \arccos\left(A(M_1, M_2) \Big/ \sqrt{A(M_1, M_1)A(M_2, M_2)}\right).$$

Having introduced the vector interpretation of hidden Markov models, another obvious metric to consider is the standard metric on vector spaces, that is, the (euclidian) norm of the difference between the two vectors

$$|M_1 - M_2| = \sqrt{\langle M_1 - M_2, M_1 - M_2 \rangle}. \tag{6.14}$$

Considering the square of this, we obtain

$$\begin{aligned} |M_1 - M_2|^2 &= \langle M_1 - M_2, M_1 - M_2 \rangle \\ &= \sum_{s \in \Sigma^*} \left(P_{M_1}(s) - P_{M_2}(s)\right)^2 \\ &= \sum_{s \in \Sigma^*} \left(P_{M_1}(s)^2 + P_{M_2}(s)^2 - 2P_{M_1}(s)P_{M_2}(s)\right) \\ &= A(M_1, M_1) + A(M_2, M_2) - 2A(M_1, M_2). \end{aligned} \tag{6.15}$$

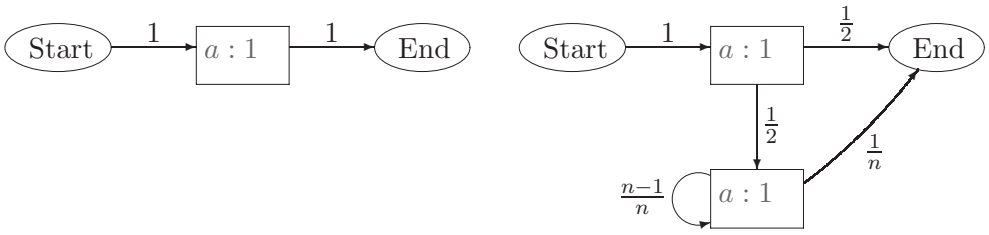
Thus this norm can be computed based on co-emission probabilities, and we propose it as a second choice for a metric on hidden Markov models.

Definition 10 *Let M_1 and M_2 be two hidden Markov models, and $A(M, M')$ be the co-emission probability of M and M'. We define the* difference *between M_1 and M_2 as*

$$D_{diff}(M_1, M_2) = \sqrt{A(M_1, M_1) + A(M_2, M_2) - 2A(M_1, M_2)}.$$

One problem with the D_{diff} metric is that $|M_1| - |M_2| \leq D_{\text{diff}}(M_1, M_2) \leq |M_1| + |M_2|$. If $|M_1| \gg |M_2|$ we therefore get that $D_{\text{diff}}(M_1, M_2) \approx |M_1|$, and we basically only get information about the length of M_1 from D_{diff}.

The metric D_{angle} is not prone to this weakness, as it ignores the length of the vectors and focuses on the sets of most probable sequences in the two models and their relative probabilities. But this metric can also lead to undesirable situations, as can be seen from figure 6.2 which shows that D_{angle} might not



(a) Hidden Markov model M_1 with $P_{M_1}(a) = 1$.

(b) Hidden Markov model M_2 with $P_{M_2}(a) = 1/2$ and $P_{M_2}(a^k) = \frac{1}{2n}(\frac{n-1}{n})^{k-2}$ for $k > 1$.

Figure 6.2: Two distinctly different models can have an arbitrarily small distance in the D_{angle} metric. It is an easy observation that $A(M_1, M_1) = 1$, $A(M_1, M_2) = 1/2$ and $A(M_2, M_2) = 1/4 + 1/(8n - 4)$; for $n \to \infty$ one thus obtains $D_{\text{angle}}(M_1, M_2) \to 0$ but $D_{\text{diff}}(M_1, M_2) \to 1/2$.

be able to discern two clearly different models. Choosing what metric to use, depends on what kind of differences one wants to highlight.

For some applications one might want a similarity measure instead of a distance measure. Based on the above metrics or the co-emission probability one can define a variety of similarity measures. We decided to examine the following two similarity measures.

Definition 11 *Let M_1 and M_2 be two hidden Markov models and $A(M, M')$ be the co-emission probability of M and M'. We define the* similarity *between M_1 and M_2 as*

$$\begin{aligned} S_1(M_1, M_2) &= \cos\left(D_{angle}(M_1, M_2)\right) \\ &= A(M_1, M_2) \Big/ \sqrt{A(M_1, M_1)A(M_2, M_2)} \end{aligned}$$

and

$$S_2(M_1, M_2) = 2A(M_1, M_2) \,/ \left(A(M_1, M_1) + A(M_2, M_2)\right) .$$

One can easily prove that these two similarity measures possess the following nice properties.

1. $0 \leq S_i(M_1, M_2) \leq 1$.
2. $S_i(M_1, M_2) = 1$ if and only if $\forall s \in \Sigma^* : P_{M_1}(s) = P_{M_2}(s)$.
3. $S_i(M_1, M_2) = 0$ if and only if $\forall s \in \Sigma^* : P_{M_i}(s) > 0 \Rightarrow P_{M_{3-i}}(s) = 0$, that is, there are no sequences that can be generated by both M_1 and M_2.

The only things that might not be immediately clear are that S_2 satisfies properties 1 and 2. This however follows from

$$A(M_1, M_1) + A(M_2, M_2) - 2A(M_1, M_2) = \sum_{s \in \Sigma^*} (P_{M_1}(s) - P_{M_2}(s))^2, \tag{6.16}$$

cf. equation 6.15, wherefore $2A(M_1, M_2) \leq A(M_1, M_1) + A(M_2, M_2)$, and equality only holds if for all sequences their probabilities in the two models are equal.

6.5 Other types of hidden Markov models

Profile hidden Markov models are not by far the only type of hidden Markov models used in computational biology. Other types of hidden Markov models have been constructed for e.g. gene prediction [79] and recognition of transmembrane proteins [134].We observe that the properties of the metrics and similarity measures introduced in the previous section do not depend on the structure of the underlying models, so once we can compute the co-emission probability of two models, we can also compute the distance between and similarity of the two models. The question thus is, can our method be extended to compute the co-emission probability for other types of hidden Markov models too?

The first thing one can observe, is that the only feature of the underlying structure of profile hidden Markov models we use, is that they are left-right models, i.e. we can number the states such that if there is a transition from state i to state j then $i \leq j$ (if the inequality is strict, that is $i < j$, then we do not even need the geometric sequence calculation, and the calculation of the co-emission probability reduces to a calculation similar to the forward/backward calculations [35, Chapter 3]). For all left-right hidden Markov models, e.g. profile hidden Markov models extended with free insertion modules [15, 69], we can thus use recursions similar to those specified in section 6.3 to compute the co-emission probability.

With some work the method can even be extended to all hidden Markov models where each state is part of at most one cycle, even if this cycle consists of more than the one state of the self-loop case. We will denote such models as hidden Markov models with only simple cycles. This extension can be useful when comparing models of coding DNA, that will often contain cycles with three states, or models describing a variable number of small domains. For general hidden Markov models we will have to resort to approximating the co-emission probability. In the rest of this section we will describe these two generalisations.

6.5.1 Hidden Markov models with only simple cycles

Assume that we can split M and M' into a number of disjoint cycles and single states, $\{C_i\}_{i \leq k}$ and $\{C'_i\}_{i \leq k'}$, such that $\{C_i\}$ and $\{C'_i\}$ are topologically sorted, i.e. for $p \in C_i$ ($p' \in C'_i$) and $q \in C_j$ ($q' \in C'_j$) and $i < j$ there is no path from q to p in M (from q' to p' in M'). To compute the co-emission probability of M and M', we will go from considering pairs of single states to considering pairs of cycles, i.e. we look at all states in a cycle at the same time.

Let C_i and $C'_{i'}$ be cycles[2] in M and M' respectively. Assume that we have already computed the co-emission probability, $A(q, q')$, for all pairs of states, q, q', where $q \in C_j$, $q' \in C'_{j'}$, $j \leq i$, $j' \leq i'$ and $(i, i') \neq (j, j')$. We will now describe how to compute the co-emission probability, $A(p, p')$, for all pairs of states, p, p', with $p \in C_i$ and $p' \in C'_{i'}$.

[2]If C_i or $C'_{i'}$ is not a cycle but a single state, the calculations of the co-emission probabilities pertaining to pairs of states from C_i and $C'_{i'}$ trivialises to calculations similar to equation 6.17 below.

As with the profile hidden Markov models, cf. section 6.3, we will proceed in a step by step fashion. We start by restricting the types of paths we consider, to get some intermediate results; we then expand the types of paths allowed, using the intermediate results, until we have covered all possible paths.

The first types of paths we consider are paths, π and π', generating identical sequences that ends in p and p', but where the immediate predecessor of p on π is not in C_i, or the immediate predecessor of p' on π' is not in $C'_{i'}$. We will denote the co-emission probability at p, p' of paths of this type as $A_e(p, p')$, as it covers the co-emission probability of paths entering the pair of cycles, $C_i, C'_{i'}$, at p, p'; it can easily be computed as

$$A_e(q, q') = \sum_{\substack{r \to q, r' \to q' \\ (r, r') \notin C_i \times C'_{i'}}} P_r(q) P_{r'}(q') A(r, r') \sum_{\sigma \in \Sigma} P_q(\sigma) P_{q'}(\sigma), \tag{6.17}$$

where $r \to q$ ($r' \to q'$) denotes that there is a transition from r to q in M (from r' to q' in M'). Here we assume that both q and q' are non-silent states; if both are silent, the sum over all symbols factor, $\sum_{\sigma \in \Sigma} P_q(\sigma) P_{q'}(\sigma)$ (the probability that q and q' generates identical symbols), should be omitted, and if one is silent and the other non-silent, the sum should furthermore only be over non-C_i (or non-$C'_{i'}$) predecessors of the silent state.

Before we proceed further, we will need some definitions that allow us to talk about predecessors of states and predecessors of pairs of states in $C_i, C'_{i'}$, and some related probabilities.

Definition 12 *Let $q \in C_i$ ($q' \in C'_{i'}$). The* predecessor *of q in C_i (q' in $C'_{i'}$) is the unique state $r \in C_i$ ($r' \in C'_{i'}$) for which there is a transition from r to q (from r' to q').*

The uniqueness of the predecessor follows from the requirement that the models only have simple cycles. For predecessors of pairs of states things are a little bit more complicated, as we want the predecessor of a pair to be the unique pair from which we can come, generating the same number of symbols (zero or one) using one transition in one or both models. This is captured by definition 13.

Definition 13 *Let $q \in C_i$ and $q' \in C'_{i'}$. The* predecessor *of q, q' in $C_i, C'_{i'}$,* $\mathrm{pre}(q, q')$, *is the pair of states r, r' where*

- *if q is silent or q' is non-silent, then r is the predecessor of q; otherwise $r = q$.*
- *if q' is silent or q is non-silent, then r' is the predecessor of q'; otherwise $r' = q'$.*

By this definition the predecessor of a pair of states, q, q', is the pair of predecessors of q and q' if both states are silent or both states are non-silent states. If q is a non-silent state and q' is a silent state then the predecessor of q, q' is the pair consisting of q and the predecessor of q'. Note that though any pair of states has a unique predecessor, the same pair of states can be the predecessor of more than one pair of states. Hence, the structure of the predecessor relationships

for pair of states will consist of a number of independent cycles with tree-like (i.e. acyclic) structures branching off at some nodes in the cycles, cf. figure 6.3. Once we have computed the co-emission probability for all pairs of states in one of these cycles, it is easy to compute the co-emission probabilities at the pairs of states in the attached tree-like structures as these computations do not involve cyclic dependencies. Therefore we will only consider the pairs of states in the cycles in the following.

We will use $P_{\mathrm{pre}(q,q')}(q,q')$ to denote the probability of getting from $r, r' = \mathrm{pre}(q,q')$ to q, q' generating identical symbols. If q and q' are both non-silent, then $P_{r,r'}(q,q') = P_r(q)P_{r'}(q')\sum_{\sigma\in\Sigma} P_q(\sigma)P_{q'}(\sigma)$; if one or both are silent, the sum over all symbols factor, $\sum_{\sigma\in\Sigma} P_q(\sigma)P_{q'}(\sigma)$, should be omitted, and if only q (q') is silent, the $P_{r'}(q')$ factor ($P_r(q)$ factor) should furthermore be omitted as $r' = q'$ (as $r = q$).

More generally we will use $P_{r,r'}(q,q')$, where $q, r \in C_i$ and $q', r' \in C'_{i'}$, to denote the probability of getting from q, q' to r, r' generating identical sequences without cycling, i.e. by just starting in q, q' and backtracking through predecessors until we reach r, r' the first time. We resolve the ambiguity of the meaning of $P_{q,q'}(q,q')$ by setting $P_{q,q'}(q,q') = 1$. To ease notation in the following, we furthermore define $P'_{r,r'}(q,q') = P_{\mathrm{pre}(q,q')}(q,q')P_{r,r'}(\mathrm{pre}(q,q'))$. The probability $P'_{q,q'}(q,q')$ is thus the probability of going through one full cycle of predecessors to q, q' until we are back at q, q'; for all $r, r' \neq q, q'$ one observes that $P'_{q,q'}(r,r') = P_{q,q'}(r,r')$.

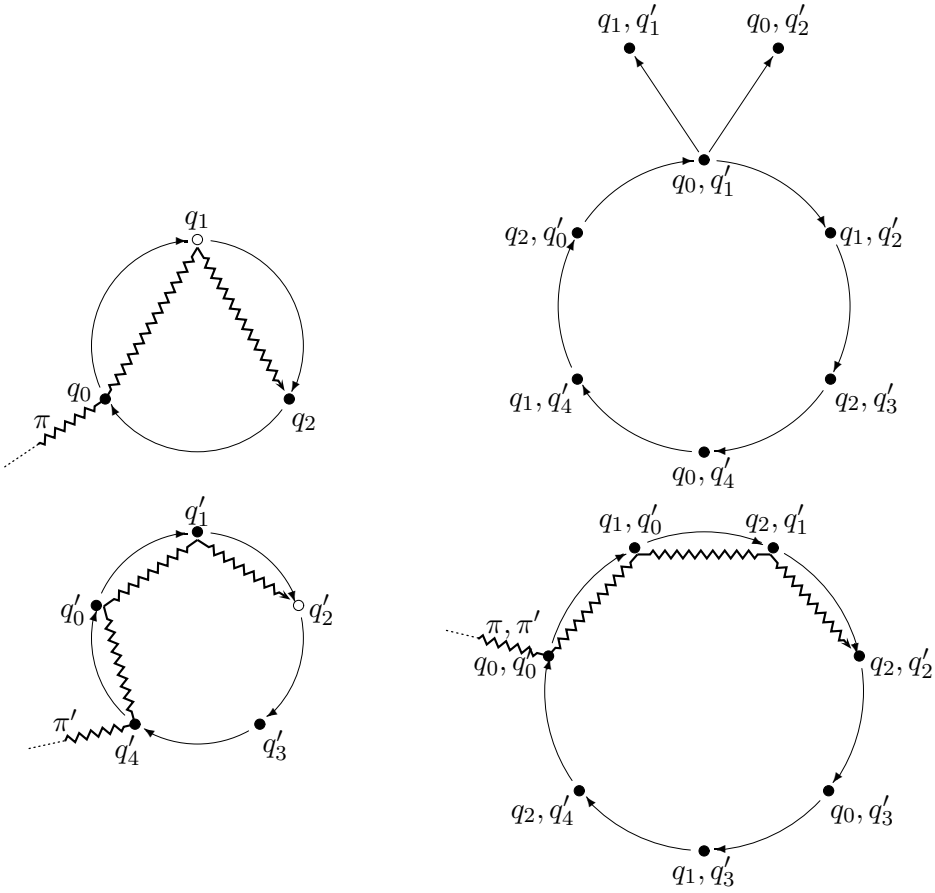
We are now ready to compute the probability of getting simultaneously to p and p' having generated identical sequences, without having been simultaneously in p and p' previously on the paths. This is

$$A_0(p,p') = \sum_{\substack{q,q' \text{ belongs to the} \\ \text{same cycle of pairs as } p,p'}} A_e(q,q')P_{q,q'}(p,p') \tag{6.18}$$

as we sum over all possible pairs, q, q', where paths ending in p, p' can have entered $C_i, C'_{i'}$. It is similar to $A_0(p,p')$ for profile hidden Markov models in the sense, that it is the probability of reaching p and p' having generated identical sequences without having looped through p, p' previously.

To compute the A_0 entries efficiently for all pairs of states in a cycle of pairs, we exploit the fact that $P_{\mathrm{pre}(q,q')}(p,p') = P_{\mathrm{pre}(q,q')}(q,q')P_{q,q'}(p,p')$ (for $\mathrm{pre}(q,q') \neq p,p'$); we can thus compute $A_0(p,p')$ in an incremental way, starting at the pair of states on the cycle having p, p' as predecessor and working our way through the cycle of pairs of states, adding the A_e values and multiplying by the probability of getting to the next successor, until we get back to p, p'. Furthermore, as

$$\begin{aligned} &A_0(\mathrm{pre}(p,p'))P_{\mathrm{pre}(p,p')}(p,p') + A_e(p,p') \\ &\quad= \sum_{\substack{q,q' \text{ belongs to the} \\ \text{same cycle of pairs as } p,p'}} A_e(q,q')P_{q,q'}(\mathrm{pre}(p,p'))P_{\mathrm{pre}(p,p')}(p,p') \\ &\qquad\qquad + A_e(p,p')P'_{p,p'}(p,p') \\ &\quad= A_0(p,p') + A_e(p,p')P'_{p,p'}(p,p') \end{aligned} \tag{6.19}$$



(a) Two example cycles, $C_i = \{q_0, q_1, q_2\}$ and $C'_{i'} = \{q'_0, q'_1, q'_2, q'_3, q'_4\}$. Hollow circles denote silent states and filled circles denote non-silent states.

(b) The structure of predecessor relations between pairs of states from C_i and $C'_{i'}$. There is an edge from one pair of states to another pair of states if the first pair of states is a predecessor of the second pair of states.

Figure 6.3: An example of a pair of cycles in M and M' and the induced predecessor structures for pairs of states from the two cycles. A path, π, ending in q_2 in C_i and a path, π', ending in q'_2 in $C'_{i'}$ are shown with zigzagged lines. If we assume that the two paths generate identical sequences, then the co-emission path, π, π', ends in q_2, q'_2 in $C_i, C'_{i'}$. Though π' enters $C'_{i'}$ at q'_4, the co-emission path, π, π', enters $C_i, C'_{i'}$ at q_0, q'_0, as the first symbol in the sequence generated by π and π' that is generated by states in both C_i and $C'_{i'}$, the second last symbol of the sequence, is generated by q_0 and q'_0 respectively.

Algorithm 6 Computation of the co-emission probabilities at all pairs of states that are in the same cycle of pairs as p, p'.

$q, q' = p, p'$
$AccumulatedP = A_e(p, p')$
$r = 1$
while $q, q' \neq \text{pre}(p, p')$ **do**
  Set q, q' to the pair of states on the cycle of pairs having the current q, q' pair as predecessor
  $AccumulatedP = AccumulatedP \cdot P_{\text{pre}(q,q')}(q, q') + A_e(q, q')$
  $r = r \cdot P_{\text{pre}(q,q')}(q, q')$
$r = r \cdot P_{q,q'}(p, p')$
repeat /* $AccumulatedP = A_0(q, q')$ *and* $r = P'_{q,q'}(q, q')$ */
  $A(q, q') = AccumulatedP \cdot \frac{1}{1-r}$
  Set q, q' to the pair of states on the cycle of pairs having the current q, q' pair as predecessor
  $AccumulatedP = AccumulatedP \cdot P_{\text{pre}(q,q')}(q, q') + (1 - r) \cdot A_e(q, q')$
until $q, q' = \text{pre}(p, p')$

we do not need to start from scratch when computing A_0 for the other pairs that belong to the same cycle of pairs as p, p' – which would require time proportional to the square of the number of pairs in the cycle – but can reuse $A_0(\text{pre}(p, p'))$ to compute $A_0(p, p')$ in constant time. Finally we observe that

$$A(p, p') = \sum_{i=0}^{\infty} P'_{p,p'}(p, p')^i A_0(p, p') = \frac{1}{1 - P'_{p,p'}(p, p')} A_0(p, p') \tag{6.20}$$

and

$$P'_{p,p'}(p, p') = P'_{q,q'}(q, q') \tag{6.21}$$

for all q, q' that belong to the same cycle of pairs as p, p'. This allows us to formulate algorithm 6 for computing the co-emission probability at all pairs in a cycle.

It is an easy observation that we run through all pairs of the cycle of pairs twice – once in the *while*-loop and once in the *repeat*-loop – thus using time proportional to the number of pairs in the cycle to compute the co-emission probabilities at each pair. Therefore, the overall time for handling the entries pertaining to the pair of cycles, $C_i, C'_{i'}$, is $\text{O}(|C_i||C'_{i'}|)$ once we have computed the A_e entries; thus the time used to compute the co-emission probability of two hidden Markov models with only simple cycles is proportional to the product of the number of transitions in the two models. This is comparable to the complexity of $\text{O}(n_1 n_2)$ for profile hidden Markov models, as this result relied on there only being a constant number of transitions to each state. In general we can compute the co-emission probability of two hidden Markov models, M_1 and M_2, with only simple cycles – including left-right hidden Markov models – in time $\text{O}(m_1 m_2)$, where m_i denotes the number of transitions in M_i.

6.5.2 General hidden Markov models

For more complex hidden Markov models, let us examine what is obtained by iterating the calculations. Let $A'_i(q, q')$ be the value computed for entry (q, q') in the i'th iteration. If we assume that q and q' are either both silent or both non-silent states, then we can compute the new entry for (q, q') as

$$A'_{i+1}(q, q') = p \sum_{\substack{r \to q \\ r' \to q'}} A'_i(r, r') P_r(q) P_{r'}(q'), \tag{6.22}$$

where p is as defined in equation 6.4 if q and q' are non-silent states, and is 1 if q and q' are silent states. If q and q' are of different types, the summation should only be over the predecessors of the silent state as in equation 6.3. In each iteration we thus extend co-emission paths with one pair of states, and $A'_i(q, q')$ is the probability of getting to q, q' having generated identical sequences on a co-emission path of length i.

The resemblance of this iterated computation to the previous calculation of A_i is evident, but a well-known mathematical sequence is not easily recognisable in equation 6.22. We can observe, though, that if we can find an array A' with $A'(s, s') = 1$ that for all entries is a fixed point (i.e. no entry changes if we recompute it using equation 6.22), it will provide us with a solution to the problem of determining the co-emission probability. This fixed point can be determined by solving the set of linear equations induced by equation 6.22.

Solving this set of linear equations might be too time consuming in many cases. Furthermore, in most cases a good estimate of the co-emission probability will be sufficient. It is thus of interest to examine how fast we can assume the iterated computation to converge. We observe that $A'_i(q, q')$ holds the probability of being in states q and q' and having generated identical prefixes in the two models after i iterations. If we assume that the only transitions from the end-states are self-loops with probability 1 (this makes the $A'_i(e, e')$ entry accumulate the probabilities of generating identical sequences after at most i iterations), then

$$A'_i(e, e') \;\leq\; A(M_1, M_2) \;\leq \sum_{q \in M_1,\, q' \in M_2} A'_i(q, q') \tag{6.23}$$

where $A(M_1, M_2)$ is the true co-emission probability of M_1 and M_2. This follows from the fact, that to generate identical sequences we must either already have done so, or at least have generated identical prefixes so far.

Now assume that for any two states, we can choose transitions to non-silent states (or the end-states) and emit different symbols with probability at least $1 - c$ where $c < 1$. Then the total weight with which $A'_i(q, q')$ contributes to the entries – not counting the special (e, e') entry – of A'_{i+1} is at most c. Thus

$$\sum_{\substack{q \in M_1,\, q' \in M_2 \\ (q,q') \neq (e,e')}} A'_{i+1}(q, q') \leq c \sum_{\substack{q \in M_1,\, q' \in M_2 \\ (q,q') \neq (e,e')}} A'_i(q, q') \tag{6.24}$$

and by induction we get

$$\sum_{q \in M_1,\, q' \in M_2} A'_i(q,q') - A'_i(e,e') \;=\; \sum_{\substack{q \in M_1,\, q' \in M_2 \\ (q,q') \neq (e,e')}} A'_i(q,q') \;\leq\; c^i, \tag{6.25}$$

which shows that the iteration method approximates the co-emission probability exponentially fast.

Though our assumption about the non-zero probability of choosing transitions and emissions such that we generate different symbols in the two models is valid for most, if not all, hidden Markov models used in practice, it is not even necessary. If d is the minimum number of paired transitions we have to follow from q and q' to get to the end-states[3] or states where we can emit different symbols after having generated identical prefixes, and c' is the probability of staying on this path and emit different symbols, we still get the exponential approximation of equation 6.25 with $c = (c')^{1/d}$. By these arguments we can approximate the co-emission probabilities and thus the metrics and similarity measures presented in section 6.4 of arbitrary hidden Markov models exponentially fast.

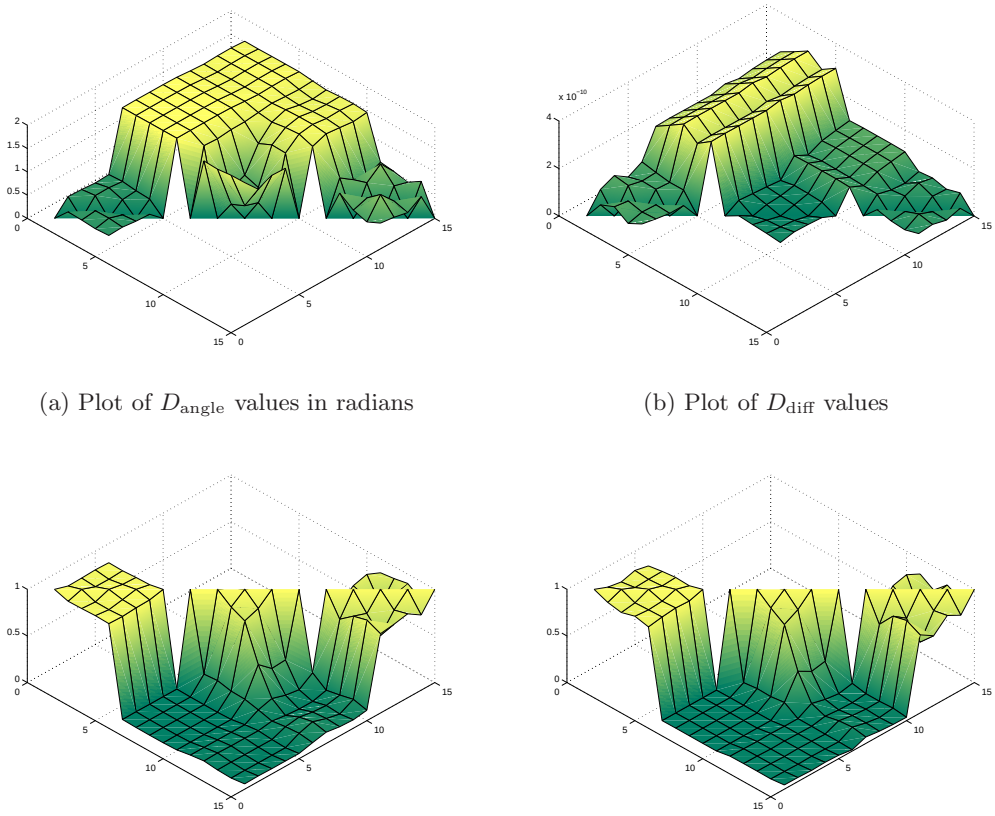
6.6 Results

We have implemented the method described in the previous sections for computing the co-emission probabilities of two left-right models. The program, which is currently available at `www.brics.dk/~cstorm/hmmcomp`, furthermore computes the derived measures. The program was used to test the four measures in a comparison of hidden Markov models for three classes of secretory signal peptides – cleavable N-terminal sequences which target secretory proteins for translocation over a membrane.

Signal peptides do not have a well-defined consensus motif, but they do share a common structure: an N-terminal region with a positive charge, a stretch of hydrophobic residues, and a region of more polar regions containing the cleavage site, where two positions are partially conserved [148]. There are statistical differences between prokaryotic and eukaryotic signal peptides concerning the length and composition of these regions [149, 110], but the distributions overlap, and in some cases, eukaryotic and prokaryotic signal peptides are found to be functionally interchangeable [16].

The hidden Markov model used here is not a profile HMM, since signal peptides of different proteins are not necessarily related, and therefore do not constitute a sequence family that can be aligned in a meaningful way. Instead, the signal peptide model is composed of three region models, each having a characteristic amino acid composition and length distribution, plus seven states modelling the cleavage site – see Nielsen and Krogh [111] for a detailed description. A combined model with three branches was used to distinguish between

[3]The end-states ensures that d exists – if we can not get to e and e', then we can not pass through q and q' and generate identical sequences. Therefore we may just as well ignore the (q,q') entry.



(a) Plot of D_{angle} values in radians

(b) Plot of D_{diff} values

(c) Plot of S_1 values

(d) Plot of S_2 values

Figure 6.4: Plots of the results obtained with the different measures. Models 1 through 5 are the models trained on eukaryotic sequences, models 6 through 10 are the models trained on Gram-positive bacterial sequences, and models 11 through 15 are the models trained on Gram-negative bacterial sequences. This gives 9 blocks, each of 25 entries, of different pairs of groups of organisms compared, but as all the measures are symmetric we have left out half the blocks showing comparisons between different groups of organisms. This should increase clarity, as no parts of the plots are hidden behind peaks.

signal peptides, signal anchors (a subset of transmembrane proteins), and non-secretory proteins; but only the part modelling the signal peptide plus the first few positions after the cleavage site has been used in the comparisons reported here.

The same architecture was used to train models of three different signal peptide data sets: eukaryotes, Gram-negative bacteria (with a double membrane), and Gram-positive bacteria (with a single membrane). For cross-validation of the predictive performance, each model was trained on five different training/test set partitions, with each training set comprising 80% of the data – i.e., any two training sets have 75% of the sequences in common.

The comparisons of the models are shown in figures 6.4 and 6.5. In general,

	Euk	G_{pos}	G_{neg}
Euk	0.231	1.56	1.52
G_{pos}		0.864	1.47
G_{neg}			0.461

(a) Table of D_{angle} values

	Euk	G_{pos}	G_{neg}
Euk	$6.77 \cdot 10^{-11}$	$2.56 \cdot 10^{-10}$	$2.67 \cdot 10^{-10}$
G_{pos}		$1.95 \cdot 10^{-11}$	$9.09 \cdot 10^{-11}$
G_{neg}			$4.43 \cdot 10^{-11}$

(b) Table of D_{diff} values

	Euk	G_{pos}	G_{neg}
Euk	0.967		
G_{pos}	$1.06 \cdot 10^{-2}$	0.547	
G_{neg}	$4.74 \cdot 10^{-2}$	0.102	0.866

(c) Table of S_1 values

	Euk	G_{pos}	G_{neg}
Euk	0.955		
G_{pos}	$1.78 \cdot 10^{-3}$	0.511	
G_{neg}	$2.93 \cdot 10^{-2}$	$4.78 \cdot 10^{-2}$	0.839

(d) Table of S_2 values

Figure 6.5: Tables of the average values of each block plotted in figure 6.4. The empty entries corresponds to the blocks left out in the plots.

models trained on cross-validation sets of the same group are more similar than models trained on data from different groups, and the two groups of bacteria are more similar to one another than to the eukaryotes. However, there are some remarkable differences between the measures. According to D_{diff}, the two bacterial groups are almost as similar as the cross-validation sets, but according to D_{angle} and the similarity measures, they are almost as dissimilar as the bacterial/eukaryotic comparisons.

This difference actually reflects the problem with the D_{diff} measure discussed in section 6.4. The distribution of sequences for models trained on eukaryotic data are longer in the vector interpretation, i.e. the probabilities are more concentrated, than the distributions for models trained on bacterial data. What we mainly see in the D_{diff} values for bacterial/eukaryotic comparisons is thus the length of the eukaryotic models. This reflects two properties of eukaryotic signal peptides: they have a more biased amino acid composition in the hydrophobic region that comprises a large part of the signal peptide sequence; and they are actually *shorter* than their bacterial counterparts, thus raising the probability of the most probable sequences generated by this model.

D_{angle} also shows that the differences within groups are larger in the Gram-positive group than in the others. This may simply reflect the smaller sample size in this group (172 sequences vs. 356 for the Gram-negative bacteria and 1137 for the eukaryotes).

The values of D_{angle} in between-group comparisons are quite close to the maximal $\pi/2$. Thus the distributions over sequences for models of different groups are close to being orthogonal. This might seem surprising in the light of the reported examples of functionally interchangeable signal peptides; but it does not mean that no sequences can be generated by both eukaryotic and bacterial models, only that these sequences have low probabilities compared to those that are unique for one group. In other words: if a random sequence is

generated from one of these models, it may with a high probability be identified which group of organisms it belongs to.

6.7 Discussion

Recall that the co-emission probability is defined as the probability that two hidden Markov models, M_1 and M_2, generate *completely identical* sequences, i.e. as $\sum_{s_1,s_2 \in \Sigma} P_{M_1}(s_1)P_{M_2}(s_2)$ where $s_1 = s_2$. One problem with the co-emission probability – and measures based on it – is that it can be desirable to allow sequences to be slightly different. One might thus want to loosen the restriction of "$s_1 = s_2$" to, e.g., "s_1 is a substring (or subsequence) of s_2," or even "$|s_1| = |s_2|$" ignoring the symbols of the sequences and just comparing the length distributions of the two models.

Another approach is to take the view that the two hidden Markov models do not generate independent sequences, but instead generates alignments with two sequences. Inspecting the equations for computing the co-emission probability, one observes that we require that when one model emits a symbol the other model should emit an identical symbol. This corresponds to only allowing columns with identical symbols in the produced alignments. A less restrictive approach would be to allow other types of columns, i.e. columns with two different symbols or a symbol in only one of the sequences, and weighting a column according to the difference it expresses. The modifications proposed in the previous paragraph can actually be considered special cases of this approach. Our method for computing the co-emission probability can easily be modified to encompass these types of modifications.

Acknowledgements

This work was inspired by a talk by Xiaobing Shi on his work with David States on aligning profile hidden Markov models. The authors would like to thank Bjarne Knudsen and Jotun Hein for their valuable suggestions. Finally we would like to thank Anders Krogh for his help with the software used to train and parse the models. Christian N. S. Pedersen and Henrik Nielsen are supported by the Danish National Research Foundation.

Chapter 7

Prediction of RNA secondary structure

> It shut like a box.
>
> —Terence Hanbury White, *The Once and Future King*

This paper describes a method for efficiently evaluating internal loops in RNA secondary structure prediction. The method is used to investigate the soundness of a commonly used heuristic. The results were presented at the Third Annual International Conference on Computational Molecular Biology and a short version of the paper, not describing the application to calculating partition functions and the timing experiments comparing the method to the commonly used method, is published in the proceedings of this conference [95]. Another short version, not describing the investigation of the heuristic, has been published in Bioinformatics [93], and the full version has been published as a technical report in the BRICS report series [94]. The method has been implemented and source code is available at `http://www.daimi.au.dk/~rlyngsoe/zuker/index.html`.

An improved algorithm for RNA secondary structure prediction

Rune B. Lyngsø* Michael Zuker† C. N. S. Pedersen‡

Abstract

Though not as abundant in known biological processes as proteins, RNA molecules serve as more than mere intermediaries between DNA and proteins, e.g. as catalytic molecules. Furthermore, RNA secondary structure prediction based on free energy rules for stacking and loop formation remains one of the few major breakthroughs in the field of structure prediction. We present a new method to evaluate all possible internal loops of size at most k in an RNA sequence, s, in time $O(k|s|^2)$; this is an improvement from the previously used method that uses time $O(k^2|s|^2)$. For unlimited loop size this improves the overall complexity of evaluating RNA secondary structures from $O(|s|^4)$ to $O(|s|^3)$ and the method applies equally well to finding the optimal structure and calculating the equilibrium partition function. We use our method to examine the soundness of setting $k = 30$, a commonly used heuristic.

7.1 Introduction

Structure prediction remains one of the most compelling, yet elusive areas of computational biology. Not yielding to overwhelming numbers and resources this area still poses a lot of interesting questions for future research. For RNA, if one restricts attention to the prediction of unknotted secondary structures, much progress has been achieved. Dynamic programming algorithms combined with the nearest neighbour model and experimentally determined free energy parameters give rigorous solutions to the problems of computing minimum free energy structures, structures that are usually close to real world optimal foldings, and partition functions that yield exact base pair probabilities.

Secondary structure in RNA is the list of base pairs that occur in a three dimensional RNA structure. According to the theory of thermodynamics the optimal foldings of an RNA sequence are those of minimum free energy, and thus the native foldings, i.e. the foldings encountered in the real world, should correspond to the optimal foldings. Furthermore, thermodynamics tells us that

*Department of Computer Science, University of Aarhus, Denmark. E-mail: rlyngsoe@daimi.au.dk. Work done in part while visiting the Institute for Biomedical Computing at Washington University, St. Louis.

†Institute for Biomedical Computing, Washington University, USA. E-mail: zuker@ibc.wustl.edu

‡Basic Research In Computer Science, Centre of the Danish National Research Foundation, University of Aarhus, Denmark. E-mail: cstorm@brics.dk.

the folding of an RNA sequence in the real world is actually a probability distribution over all possible structures, where the probability of a specific structure is proportional to an exponential of the free energy of the structure. For a set of structures, the partition function is the sum over all structures of the set of the exponentials of the free energies.

Information on the secondary structure of an RNA molecule can be used as a stepping-stone to modelling the full structure of the molecule, which in turn relates to the biological function. As recent experiments have shown that RNA molecules can undertake a wide range of different functions [66], the prediction of RNA secondary structure should continue to be important for biomolecule engineering.

A model was proposed in [142, 141] to calculate the stability (in terms of free energy) of a folded RNA molecule by adding independent contributions from base pair stacking and loop destabilising terms from the secondary structure. This model has proven a good approximation of the forces governing RNA structure formation, thus allowing fair predictions of real structures by determining the most stable structures in the model of a given sequence.

Based on this model, algorithms for computing the most stable structures have been proposed e.g. in [167, 112]. Zuker [164] proposes a method to determine all base pairs that can participate in structures with a free energy within a specified range from the optimal. McCaskill [101] demonstrates how a related dynamic programming algorithm can be used to calculate equilibrium partition functions, which lead to exact calculations of base pair probabilities in the model.

A major problem for these algorithms is the time required to evaluate possible internal loops. In general, this requires time $O(|s|^4)$ which is often circumvented by assuming that only 'small' loops need to be considered (e.g. [101]). This risks missing some optimal large internal loops, especially when folding at high temperatures, but the time required for evaluating internal loops is reduced to $O(|s|^2)$ thus reducing the overall complexity to $O(|s|^3)$. If the stability of an internal loop can be assumed only to depend on the size of the internal loop, Waterman et. al. [153] describes how to reduce the time requirement to $O(|s|^3)$[1]. This is further improved to $O(|s|^2 \log^2 |s|)$ for convex free energy functions by Eppstein et.al. [39]. Affine free energy functions (i.e. of the form $a + bn$, where n is the size of the loop) allows for $O(|s|^2)$ computation time by borrowing a simple method used in sequence alignment [46].

Unfortunately the currently used free energy functions for internal loops are not convex, let alone affine. Furthermore, the technique described in [39] hinges on the objective being to find a structure of maximum stability, and thus does not translate to the calculation of the partition function of [101] where a Boltzmann weighted sum of contributions to the partition function is calculated.

In this paper we will describe a method based on a property of current free energy functions for internal loops that allows all internal loops to be evaluated

[1]This method is also referred to by [101] where a combination of the above methods is proposed – a free energy function only dependent on loop size is used for large loops, while small loops are treated specially.

in time $O(|s|^3)$. This method is applicable both to determining the most stable structure and to calculating the partition function.

The rest of this paper is structured as follows. In section 7.2 we briefly review the basic dynamic programming algorithm for RNA secondary structure prediction and introduce the notation we will be using. In section 7.3 we present a method yielding cubic time algorithms for evaluating internal loops for certain free energy functions. We argue that this method can be used with currently used free energy functions in section 7.3.2, and describe how the same technique can be used to calculate the contributions to the partition function from structures with internal loops in section 7.3.3. In section 7.4 we compare our method to the previously used method, and in section 7.5 we present an experiment using the new algorithm to analyse a hitherto commonly used heuristic. In section 7.6 we discuss some future directions for improvements.

7.2 Basic dynamic programming algorithm

A secondary structure of an RNA sequence s is a set S of base pairs $i \cdot j$ with $1 \leq i < j \leq |s|$, such that $\forall i \cdot j, i' \cdot j' \in S : i = i' \Leftrightarrow j = j'$. Thus, any base can take part in at most one base pair. We will further assume that the structure does not contain pseudo-knots. A pseudo-knot is two "overlapping" base pairs, that is, base pairs $i \cdot j$ and $i' \cdot j'$ with $i < i' < j < j'$.

One can view a pseudo-knot free secondary structure S as a collection of *loops* together with some *external* unpaired bases (see figure 7.1). Let $i < k < j$ with $i \cdot j \in S$. Then k is said to be *accessible* from $i \cdot j$ if for all $i' \cdot j' \in S$ it is not the case that $i < i' < k < j' < j$. The base pair $i \cdot j$ is said to be the *exterior* base pair of (or *closing*) the loop consisting of $i \cdot j$ and all bases accessible from it. If i' and j' are accessible from $i \cdot j$ and $i' \cdot j' \in S$ – observe that for a structure without pseudo-knots either both or none of i' and j' will be accessible from $i \cdot j$ if $i' \cdot j' \in S$ – then $i' \cdot j'$ is called an *interior* base pair of the loop and is said to be accessible from $i \cdot j$. If there are no interior base pairs the loop is called a *hairpin* loop. With one interior base pair it is called a *stacked pair* if $i' = i + 1$ and $j' = j - 1$, and otherwise it is called an *internal* loop (*bulges* are a special kind of internal loops with either $i' = i + 1$ or $j' = j - 1$). Loops with more than one interior base pair are called *multibranched* loops. Unpaired bases and base pairs not accessible from any base pair are called external.

RNA secondary structure prediction is the problem of determining the most stable structure for a given sequence. We measure stability in terms of the free energy of the structure. Thus we want to find a structure of minimal free energy which we will also call an optimal structure. The energy of a secondary structure is assumed to be the sum of the energies of the loops of the structure and furthermore the loops are assumed to be independent, that is, the energy of a loop only depends on the loop and not on the rest of the structure [141].

Based on these assumptions one can specify a recursion to calculate the energy of the optimal structure for a sequence s [167, 112]. Before presenting our improvement to the part of the algorithm dealing with internal loops, we will briefly review the hitherto used method. We use the same notation as

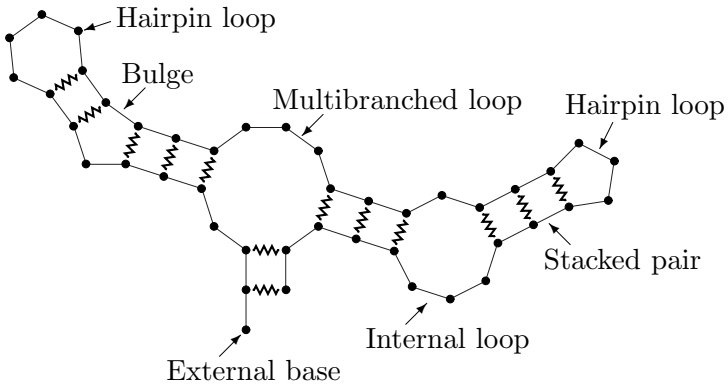


Figure 7.1: An example RNA structure. Bases are depicted by circles, the RNA backbone by straight lines and base pairings by zigzagged lines.

in [143]. Four arrays[2] – W, V, VBI and VM – are used to hold the minimal free energy of certain restricted structures of subsequences of s. The entries of these arrays are interdependent and can be calculated recursively using prespecified free energy functions – eS, eH, eL and eM – for the contributions from the various types of loops as follows.

- The energy of an optimal structure of the subsequence from 1 through i:

$$W(i) = \min\{W(i-1), \min_{1<j\leq i}\{W(j-1) + V(j,i)\}\}.$$

- The energy of an optimal structure of the subsequence from i through j closed by $i \cdot j$:

$$V(i,j) = \min\{eH(i,j), eS(i,j) + V(i+1,j-1), VBI(i,j), VM(i,j)\}$$

  where $eH(i,j)$ is the energy of a hairpin loop closed by $i \cdot j$ and $eS(i,j)$ is the energy of stacking base pair $i \cdot j$ with $i+1 \cdot j-1$.

- The energy of an optimal structure of the subsequence from i through j where $i \cdot j$ closes a bulge or an internal loop:

$$VBI(i,j) = \min_{\substack{i<i'<j'<j \\ i'-i+j-j'>2}} \{eL(i,j,i',j') + V(i',j')\}$$

  where $eL(i,j,i',j')$ is the energy of a bulge or internal loop with exterior base pair $i \cdot j$ and interior base pair $i' \cdot j'$.

[2]Actually two arrays – V and W – suffices, but we will use four arrays to simplify the description. Below we will introduce a fifth array WM that will also be needed in an efficient implementation.

- The energy of an optimal structure of the subsequence from i through j where $i \cdot j$ closes a multibranched loop:

$$VM(i,j) = \min_{\substack{i<i_1<j_1< \\ \cdots \\ <i_k<j_k<j}} \{eM(i,j,i_1,j_1,\ldots,i_k,j_k) + \sum_{l=1}^{k} V(i_l,j_l)\}$$

  where $k > 1$ and $eM(i,j,i_1,j_1,\ldots,i_k,j_k)$ is the energy of a multibranched loop with exterior base pair $i \cdot j$ and interior base pairs $i_1 \cdot j_1$, $\ldots$, $i_k \cdot j_k$.

When all entries of these arrays have been filled out, $W(|s|)$ contains the free energy for optimal structures and an optimal structure can be determined by backtracking the calculations that led to this free energy.

To make the problem of determining the optimal secondary structure tractable the following simplifying assumption is often made. The energy of multibranched loops can be decomposed into linear contributions from the number of unpaired bases in the loop, the number of branches in the loop and a constant [166][3], that is

$$eM(i,j,i_1,j_1,\ldots,i_k,j_k) = a + bk + c\Big(i_1 - i - 1 + j - j_k - 1 + \sum_{l=1}^{k-1}(i_{l+1} - j_l - 1) \Big). \quad (7.1)$$

We introduce an extra array

- The energy of an optimal structure of the subsequence from i through j that constitutes part of a multibranched loop structure, that is, where unpaired bases and external base pairs are penalised according to equation 7.1:

$$WM(i,j) = \min\{V(i,j) + b, WM(i,j-1) + c, WM(i+1,j) + c, \min_{i<k\leq j}\{WM(i,k-1) + WM(k,j)\}\}$$

which enables us to restate the calculation of the energy of the optimal multibranched loop as

$$VM(i,j) = \min_{i+1<k\leq j-1}\{WM(i+1,k-1) + WM(k,j-1) + a\}.$$

Based on these recurrence relations we can by dynamic programming calculate the energy of the optimal structure in time $O(|s|^3)$ – assuming that the free energy functions can be evaluated in constant time – except for the calculation of the entries of VBI which requires $O(|s|^4)$ in total. The bottleneck of finding the optimal structures is thus the evaluation of internal loops. In the following section we will present a method to reduce the time used calculating the entries of VBI from $O(|s|^4)$ to $O(|s|^3)$, thereby improving the time complexity of the overall RNA secondary structure prediction algorithm from $O(|s|^4)$ to $O(|s|^3)$.

[3]It is known that the stability of a multibranched loop also depends on the stacking effects of the base pairs in the loop and their neighbouring unpaired bases. These effects can also be handled efficiently, but for simplicity we have omitted the details here.

7.3 Efficient evaluation of internal loops

Examining the recursion for internal loops one observes that two base pairs, $i \cdot j$ and $i' \cdot j'$, may be compared as candidates for the interior base pair for numerous exterior base pairs. If $V(i,j) \ll V(i',j')$, it is evident that we would not have to consider $i' \cdot j'$ as a candidate interior base pair for any entry of VBI where $i \cdot j$ would also be a candidate interior base pair.

Though it would often in practice be the case that we could a priori discard many candidate interior base pairs by the above observation, we can not in general guarantee this to be the case. To get an improvement in the worst case performance of the evaluation of internal loops, we thus have to examine properties of the energy functions for internal loop stability that will allow us to group base pairs and entries of VBI, such that we only have to make one comparison between $i \cdot j$ and $i' \cdot j'$ to determine which one would yield the more stable structure for the entire group of entries. In this section we will exploit such properties of currently used energy functions leading to an algorithm for evaluating internal loops requiring worst case time $\mathrm{O}(|s|^3)$.

Currently used energy rules for internal loop stability (cf. [163]) split the contributions into three parts:

- An entropic term that depends on the size of the loop.
- Stacking energies for the mismatched base pairs adjacent to the enclosing (exterior *and* interior) base pairs.
- An asymmetry penalty for asymmetric loops.

With this separation we can rewrite the internal loop energy function as

$$\begin{aligned} eL(i,j,i',j') = &\,\mathrm{size}(i'-i+j-j'-2)+ \\ &\,\mathrm{stacking}(i \cdot j) + \mathrm{stacking}(i' \cdot j')+ \\ &\,\mathrm{asymmetry}(i'-i-1, j-j'-1). \end{aligned} \tag{7.2}$$

Figure 7.2 gives a graphical representation of these components of the internal loop energy function. In the following we will further assume that the lopsidedness and the size dependence of the asymmetry function can be separated out, or more specifically that

$$\mathrm{asymmetry}(k+1, l+1) = \mathrm{asymmetry}(k,l) + g(k+l) \tag{7.3}$$

holds. The change of the asymmetry function when varying the size while maintaining lopsidedness thus only depends on the size of the loop. This is equivalent to assuming that

$$\mathrm{asymmetry}(k,l) = \mathrm{lopsidedness}(|k-l|) + \mathrm{size}'(k+l), \tag{7.4}$$

where one can observe that the g term in equation 7.3 corresponds to changes in the size$'$ term in equation 7.4. This size-dependence of the asymmetry function

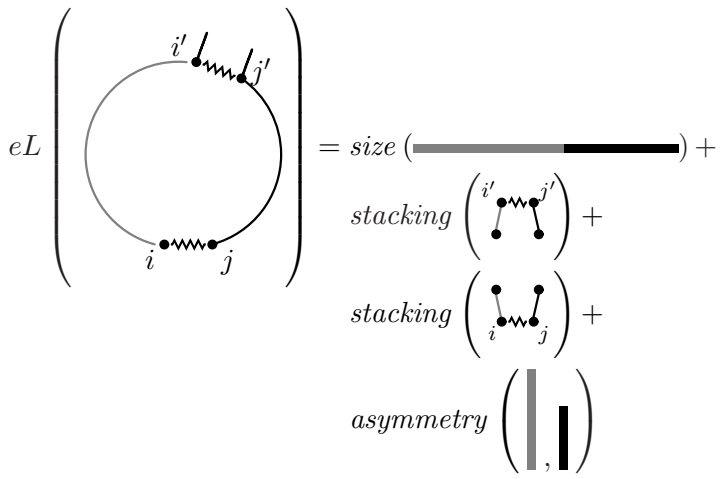


Figure 7.2: The energy function for internal loops can be split into a sum of independent contributions.

can be moved to the size-function of the overall internal loop energy function, thus allowing us to restate the assumption of equation 7.3 as

$$\text{asymmetry}(k+1, l+1) = \text{asymmetry}(k, l). \tag{7.5}$$

In the rest of this paper we will therefore omit the g term, but the formulation of equation 7.3 might be useful when specifying or recognising an asymmetry function obeying the assumption.

7.3.1 Finding optimal internal loops

If the assumption of equation 7.3 holds, we propose algorithm 7 as an efficient alternative to compute the $VBI(i, j)$ entries in the dynamic programming algorithm for predicting RNA secondary structure. The algorithm is an extension of the ideas in [153] where an $O(n^3)$ method for calculating the entries of VBI, assuming that the stability of an internal loop only depends on the size of the loop, was presented. The rationale behind the algorithm is, that when we extend loops while retaining lopsidedness we can reuse comparisons as depicted in figure 7.3. Thus for a pair of indices, i and j, the algorithm *does not* compute the $VBI(i, j)$ entry. Instead, if we denote all internal loops with a specific size and exterior base pair as a *class* of internal loops, the algorithm evaluates all classes of internal loops where $i \cdot j$ is the middle candidate base pair, that is, choosing $i \cdot j$ as the interior base pair results in a symmetric loop (or almost symmetric – loops of odd size will always have a lopsidedness of at least one).

Proposition 3 *Algorithm 7 computes VBI correctly under the assumption of equation 7.3. Furthermore, the time required to compute the entire table is* $O(n^3)$.

The time complexity of $O(n^3)$ is easy to see, since the algorithm for each of the $O(n^2)$ pairs of indices, i and j, uses time $O(n)$. To prove the correctness

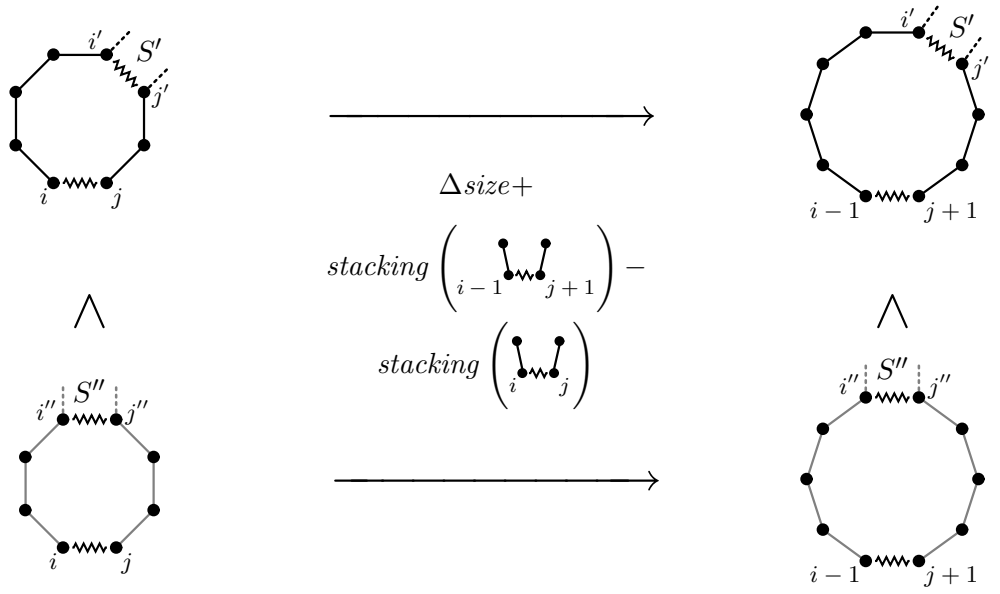


Figure 7.3: The difference in destabilising energy when extending a loop from being closed by $i \cdot j$ to being closed by $i-1 \cdot j+1$ is determined solely by the size of the loop and the change in stacking stability of the closing base pair. We can thus reuse comparisons between different choices of interior base pairs, e.g. $i' \cdot j'$ and $i'' \cdot j''$.

of the algorithm, we will start by sketching a simpler algorithm for which the correctness is obvious, but that has the drawback of using space $\mathrm{O}(n^3)$. Then we will argue that algorithm 7 is similar to this algorithm except for the order in which the computations are carried out, that is, the order in which the different candidate interior loops for a specific entry of VBI are evaluated. Hence, the correctness of the simpler algorithm implies the correctness of algorithm 7.

We define a new array VBI' such that $VBI'(i,j,l)$ is the minimal energy of an internal loop of size l with exterior base pair $i \cdot j$. The following lemma establishes a useful relationship between the entries of VBI'.

Lemma 10 *If equation 7.3 holds, then for* $l > 2$

$$VBI'(i,j,l) = \min \begin{cases} VBI'(i+1,j-1,l-2)+ \\ \quad \text{size}(l) - \text{size}(l-2)+ \\ \quad \text{stacking}(i \cdot j) - \text{stacking}(i+1 \cdot j-1) \\ V(i+1,j-l-1) + eL(i,j,i+1,j-l-1) \\ V(i+l+1,j-1) + eL(i,j,i+l+1,j-1). \end{cases} \tag{7.6}$$

Proof. By definition

$$VBI'(i,j,l) = \min_{\substack{i<i'<j'<j \\ i'-i+j-j'-2=l}} \{eL(i,j,i',j') + V(i',j')\}. \tag{7.7}$$

The last two entries of equation 7.6 handle the cases where this minimum is obtained by a bulge, that is at $i' = i+1$ or $j' = j-1$. Otherwise the minimum is the minimum over

$$\begin{aligned}
&eL(i,j,i',j') + V(i',j') \\
&= \text{size}(l) + \text{asymmetry}(i'-i-1, j-j'-1) \\
&\qquad + \text{stacking}(i \cdot j) + \text{stacking}(i' \cdot j') + V(i',j') \\
&= \text{size}(l) + \text{asymmetry}(i'-i-2, j-j'-2) \\
&\qquad + \text{stacking}(i \cdot j) + \text{stacking}(i' \cdot j') + V(i',j') \\
&= \text{size}(l-2) + \text{asymmetry}(i'-i-2, j-j'-2) \\
&\qquad + \text{stacking}(i+1 \cdot j-1) + \text{stacking}(i' \cdot j') + V(i',j') \\
&\qquad + \text{size}(l) - \text{size}(l-2) \\
&\qquad + \text{stacking}(i \cdot j) - \text{stacking}(i+1 \cdot j-1)
\end{aligned}$$

for all $i' < j'$ with $i' > i+1$, $j' < j-1$ and $i' - (i+1) + (j-1) - j' - 2 = l-2$. The last two lines of the last equation are independent of i' and j', and can thus be moved out of the minimum. The minimum of the first two lines over i' and j' satisfying the above constraints is exactly $VBI'(i+1, j-1, l-2)$, thus proving the lemma. □

Lemma 10 yields the basic recursion needed to compute each entry of VBI' in constant time[4]. It is easily observed that VBI' contains $\mathrm{O}(n^3)$ entries and that VBI can be calculated from VBI' as

$$VBI(i,j) = \min_l \{VBI'(i,j,l)\}, \tag{7.8}$$

each of the $\mathrm{O}(n^2)$ entries being computable in time $\mathrm{O}(n)$. Thus VBI can be computed in time $\mathrm{O}(n^3)$ including the time used to compute VBI'. Unfortunately the table VBI' requires space $\mathrm{O}(n^3)$, thus rendering this method somewhat impractical. However, it can be observed that we only need $VBI'(i,j,l)$ at most twice, namely when

- determining whether it is a candidate for $VBI(i,j)$.
- calculating the value of $VBI'(i-1, j+1, l+2)$.

This is used in algorithm 7 to avoid maintaining the VBI' table. Instead we use E to hold the value[5] that should otherwise be stored in one of the entries of VBI'. We use this value to check it as a candidate for the relevant entry of VBI, according to equation 7.8, in the second minimum of the **for**-loop in algorithm 7. After this check we only need the value to calculate the value corresponding to another entry of VBI'; this is done in the first minimum in the next iteration of the **for**-loop. Now the value can safely be discarded as it is no longer needed.

[4]This is of course assuming that entries of V are ready at hand when we need them. The cost of computing the entries of V can however be charged to V, and thus we don't have to consider it here.

[5]To avoid having to keep adding and subtracting the size and external stacking terms in algorithm 7 we defer adding these terms until the value is considered as a candidate for one of the VBI entries.

Algorithm 7 Evaluation of classes of internal loops with size $2l+a$ and exterior base pair $i - l \cdot j + l + a$.

/ When $a = 0$ loops of even size are handled and when $a = 1$ loops of odd size are handled; this is necessary as we increase the loop size by two in each iteration. */*
for $a = 0$ **to** 1 **do**
  / E maintains the energy of the optimal loop except for* size *and external stacking contributions. */*
  $E = \infty$
  / Iterate through the exterior base pairs. For even sized loops we skip $l = 1$ as this yields a stacked base pair. */*
  for $l = 2 - a$ **to** $\min\{i - 1, |s| - j - a\}$ **do**
    / Examine the two new candidate interior base pairs, i.e. the interior base pairs next to the currently considered exterior base pair. */*
    $E = \min\{E, V(i - l + 1, j - l + 1) +$
      $\text{asymmetry}(0, 2l + a - 2) +$
      $\text{stacking}(i - l + 1, j - l + 1),$
      $V(i + a + l - 1, j + a + l - 1) +$
      $\text{asymmetry}(2l + a - 2, 0) +$
      $\text{stacking}(i + a + l - 1, j + a + l - 1)\}$
    / Update VBI for the currently considered exterior base pair. */*
    $VBI(i - l, j + a + l) = \min\{VBI(i - l, j + a + l),$
      $E + \text{size}(2l + a - 2) + \text{stacking}(i - l, j + a + l)\}$
  end for
end for

It is straightforward to verify that the value that should otherwise have been stored in $VBI'(i', j', l)$ is handled when algorithm 7 is invoked with $i = i' + \lfloor \frac{l}{2} \rfloor$ and $j = j' - \lceil \frac{l}{2} \rceil$. The correctness of the value maintained in E can easily be proved by induction, using lemma 10.

7.3.2 The Asymmetry Function Assumption

The assumption of equation 7.3 might seem somewhat unrealistic as, for one thing, we treat bulges just as if they were normal internal loops. If equation 7.3 only holds for $\min(k, l) \geq c - 1$ we can however modify the algorithm to handle this situation, a modification that does lead to an increase in time complexity by a factor of c, for a total time complexity of $O(cn^3)$.

This is done simply by examining all the $O(cn^3)$ loops with a stem of unpaired bases shorter than c separately, and then applying the technique of extending loops while retaining lopsidedness to the rest of the loops, starting the iteration at $l = c$ and adding or subtracting $c - 1$ from the indices of the interior base pairs considered, including where they partake in the parameters of the asymmetry function. Thus bulges can be treated specially while only doubling the time complexity.

Papanicolaou et. al. [114] propose an asymmetry penalty function on the

form

$$\text{asymmetry}(k,l) = \min\{K, N_{k,l} f(M_{k,l})\}, \tag{7.9}$$

usually called Ninio type asymmetry penalty functions, with $N_{k,l} = |k-l|$ and $M_{k,l} = \min\{k,l,c\}$. The constants K and c and the function f are parameters of the penalty function. We observe that $N_{k+1,l+1} = N_{k,l}$ and that $M_{k+1,l+1} = M_{k,l}$ if $\min\{k,l\} \geq c$. For $\min\{k,l\} \geq c$ it thus follows that asymmetry$(k+1,l+1)$ = asymmetry(k,l), and thus asymmetry functions on this form adheres to the above relaxed assumption, allowing us to solve the RNA secondary structure prediction problem using Ninio type asymmetry penalty functions in time $\mathrm{O}(cn^3)$. In [114] an asymmetry function with $c = 5$ was proposed. A modification of the parameters based on thermodynamic studies was proposed in [120]. With these parameters $c = 1$ thus allowing us to treat only bulges specially[6].

7.3.3 Computing the partition function

In [101] it is described how to compute the full equilibrium partition functions and thus the probabilities of all base pairs. The method used closely mimics the free energy calculation described above, and thus it should be of no surprise that the method presented in this paper also applies to the calculation of the partition functions. In this section we will briefly sketch how to compute the internal loops' contribution to the partition functions. The reader is refered to [101] for the full details on how to calculate the partition functions.

In [101] $Q_{i,j}$ denotes the partition function on the segment from base i through base j, while $Q^b_{i,j}$ denotes the *restricted* partition function for the same sequence segment with the added constraint that bases i and j form a base pair[7]. We will specify how to calculate the contributions from structures with an internal loop closed by $i \cdot j$.

From [101, equations 4 and 7] it is seen that the contributions from these structures – if we consider a stacked pair to be an internal loop of size 0 – are

$$\sum_{i<h<l<j} e^{-eL(i,j,h,l)/kT} Q^b_{h,l}\,, \tag{7.10}$$

where [101, equation 7] uses $F_2(i,j,h,l)$ to gather the energies of all structures with an internal loop with base pairs $i \cdot j$ and $h \cdot l$, thus reducing the terms of the sum to $e^{-F_2(i,j,h,l)/kT}$.

Similar to the approach in section 7.3.1 we define $Q^{il}_{i,j,l}$ to be the partition function for all structures with an internal loop of size l closed by $i \cdot j$, thus corresponding to $VBI'(i,j,l)$ in the energy calculations in section 7.3.1. Now it can be proved that

[6]Sequence dependent destabilising energies are available for internal loops of size three. These – and similar specific energy functions for small loops – can be handled as a special case without affecting the general method for calculating internal loop stability though.

[7]Thus $Q^b_{i,j}$ corresponds to $V(i,j)$ in energy calculations.

Algorithm 8 Evaluation of classes of internal loops with size $2l+a$ and exterior base pair $i-l\cdot j+l+a$.

/ Make sure to handle both even sized and odd sized loops. */*
for $a = 0$ **to** 1 **do**
  / Q maintains the partition function contribution for the current class of internal loops except for* size *and external stacking factors. */*
  $Q = 0$
  / Iterate through the exterior base pairs. For even sized loops we skip $l = 1$ as this yields a stacked base pair. */*
  for $l = 2-a$ **to** $\min\{i-1, |s|-j-a\}$ **do**
    / Add contributions from the two new interior base pairs, i.e. the interior base pairs next to the currently considered exterior base pair. */*
    $Q = Q + Q^b_{i-l+1,j-l+1} e^{-(\text{asymmetry}(0,2l+a-2)+\text{stacking}(i-l+1\cdot j-l+1))/kT}$
      $+ Q^b_{i+a+l-1,j+a+l-1} e^{-(\text{asymmetry}(2l+a-2,0)+\text{stacking}(i+a+l-1\cdot j+a+l-1))/kT}$
    / Update Q^b with contributions from the currently considered class of internal loops. */*
    $Q^b_{i-l,j+a+l} = Q^b_{i-l,j+a+l} + Q e^{-(\text{size}(2l+a-2)+\text{stacking}(i-l\cdot j+a+l))/kT}$
  end for
end for

$$Q^{il}_{i,j,l} = Q^{il}_{i+1,j-1,l-2} e^{(\text{size}(l-2)-\text{size}(l)+\text{stacking}(i+1\cdot j-1)-\text{stacking}(i\cdot j))/kT} + Q^b_{i+1,j-l-1} e^{-eL(i,j,i+1,j-l-1)/kT} + Q^b_{i+l+1,j-1} e^{-eL(i,j,i+l+1,j-1)/kT} \quad (7.11)$$

by similar arguments as in the proof of lemma 10. There is a slight problem if $\text{stacking}(i\cdot j) = \infty$ or $\text{stacking}(i+1\cdot j-1) = \infty$ – that is, if bases i and j or bases $i+1$ and $j-1$ does not form a base pair – but in the proof of equation 7.11 this can be handled by assuming that all stacking energies are finite. In the algorithm we handle it by postponing the multiplication with the exponential of the stacking energies until adding the contribution of $Q^{il}_{i,j,l}$ to $Q^b_{i,j}$. We can now rewrite equation 7.10 as

$$\sum_{l=0}^{j-i-2} Q^{il}_{i,j,l}\,, \quad (7.12)$$

and based on equations 7.11 and 7.12 we can now proceed to present algorithm 8 to handle internal loop contributions to the partition function; the observant reader will notice the close similarity between algorithms 7 and 8. Again it is an easy observation that the time complexity is $\mathrm{O}(n^3)$, and the correctness of algorithm 8 can be proven by arguments similar to the proof of the correctness of algorithm 7.

7.4 Implementation

The method described in this paper has been implemented in *ZUKER*[8], a C program to find the optimal structure of an RNA sequence based on energy rules. To be able to compare the performance of this method to previously used methods, compiler directives determines whether the compiled code will use complete enumeration of all internal loops or the method described here, and whether only to consider loops smaller than a specified size. By this we hope to have eliminated most of the noise due to differences in implementations so as to get a comparison of the underlying methods.

We decided to test our method against the complete enumeration method, both when using a cutoff size of 30 for internal loops (a commonly used cutoff size) and when allowing loops of any size. All four methods were tested with random sequences of length 500 and 1000, respectively, and the results are summarised in Table 7.1. As expected a huge increase in performance is obtained when allowing internal loops of any size, but even when limiting internal loops to size at most 30, our method obtains a speedup of 30 – 40 % compared to the complete enumeration method.

Sequence length	500	1000
Complete enumeration, unlimited loop size	2,119 s	35,988 s
Our method, unlimited loop size	127 s	1,123 s
Complete enumeration, loop size ≤ 30	48 s	264 s
Our method, loop size ≤ 30	30 s	182 s

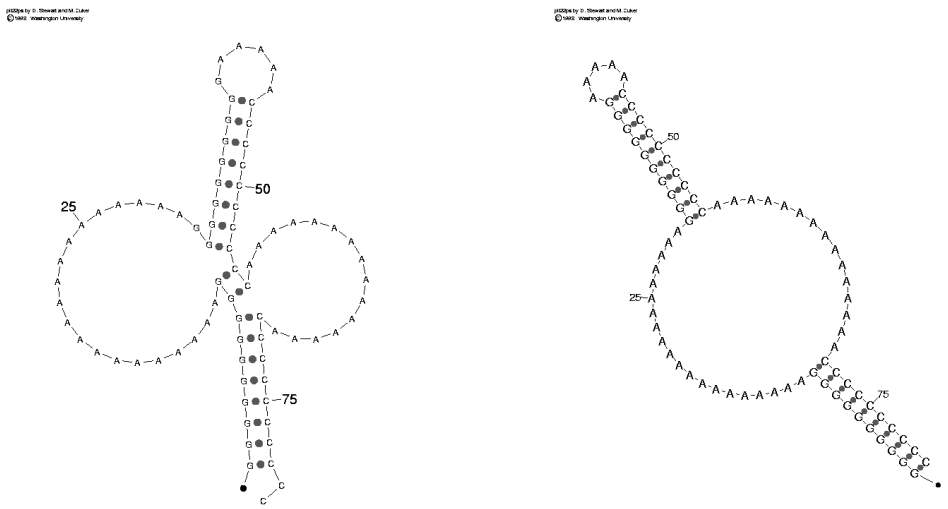
Table 7.1: Comparison of different methods to evaluate internal loops. The running times are as reported by the Unix `time` command on a Silicon Graphics Indigo 2.

The current implementation encompasses the method for calculating the optimal substructure on the parts of the sequence *excluding* the substring from i through j, thus allowing the prediction of suboptimal structures as described in [164] and calculation of base pair probabilities based on partition functions as described in [101]. We are currently working on adding coaxial stacking modifications to the multibranched loop evaluations, and on extending the program to take other parameters, e.g. mutual information or base pair confidences obtained from alignments, into account.

7.5 Experiments

To make the problem of determining the optimal secondary structure for an RNA sequence more tractable it has hitherto been common practice to limit the size of internal loops. The `mfold` server has a built-in limit of 30 and in [65] a limit of 30 is also hinted at. With the ability to make a rigorous search for the optimal structure, we decided to see whether this limit has been reasonable.

[8]ZUKER – Unlimited Ken Energy-based RNA-folding, the name reflecting that no limit is imposed on how far to look for the closing base pair of an internal loop.



(a) Maximum loop size 30;Energy: −29.6 kcal/mol

(b) No maximum loop size; Energy: −42.9 kcal/mol

Figure 7.4: Foldings of the sequence GGGGGGGGGGAAAAAAAAAAAAAAAAAA GGGGGGGGGGAAAAACCCCCCCCCCAAAAAAAAAAAAAAAACCCCCCCCCC

7.5.1 A constructed 'mean' sequence

The easiest way to find a loop of size larger than 30 is of course to construct it yourself. We constructed a sequence of length 80 consisting only of C's, G's and A's (but no U's), designed to fold into two stems of 10 base pairing C's and G's separated by an internal loop of 35 unpaired A's, and with a hairpin loop consisting of 5 A's. The result of folding this sequence at 37 °C with and without a size limit of 30, respectively, is shown in figure 7.4

One can observe that the prediction with a cutoff size of 30 does in fact pair most of the C's with G's – but instead of having the A's in one big internal loop they are folded out as two bulges. A further observation is that there can indeed be a major increase in stability by choosing one large internal loop instead of two smaller bulges.

Though this example may be cute, the interesting question of course is whether RNA sequences for which the optimal structure contains a large internal loop occur naturally. The reason that a cutoff size of 30 has been deemed reasonable is of course that no internal loops even close to this size are observed in a standard structure prediction at 37 °C. But when the temperature is increased, base pairs become less stable which may cause short stems of stacking base pairs to break up. We thus decided to look at a couple of sequences for which structure prediction at higher temperatures would be interesting.

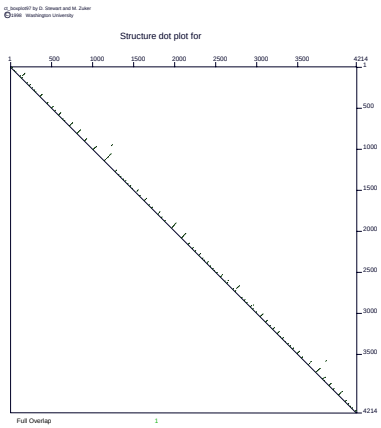


Figure 7.5: Dot-plot of the prediction of the Qβ structure at 65 °C. The absence of long range base pairings (dots far away from the diagonal) is apparent.

7.5.2 Qβ

Jacobson [71] reported on some experiments on determining structural features in Qβ denatured to various extents. It is believed that denaturing effects relates to temperature effects, and we thus chose to fold this sequence at nine different temperatures in the range from 45 °C to 100 °C to see whether we would find any of the structural features reported by Jacobson.

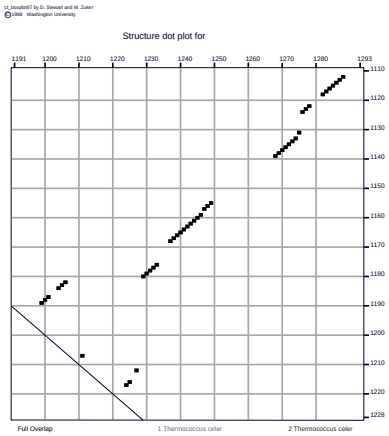
None of these predicted foldings showed any signs of the features Jacobson reported – at higher temperatures the structure simply came apart as small structural fragments, usually covering less than 100 nucleotides. Furthermore we did not observe any internal loops larger than size 25. An example prediction is shown in figure 7.5.

7.5.3 *Thermococcus celer*

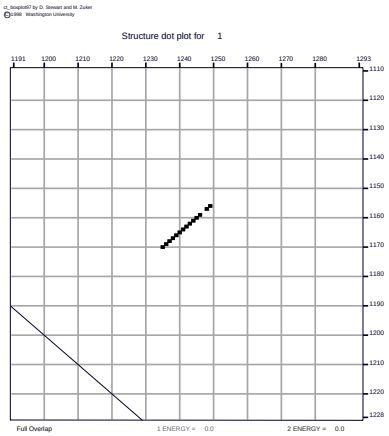
Thermococcus celer is an organism that lives in solfataric marine water holes of Vulcano, Italy, at temperatures around 90 °C; its optimal growth temperature is reported to be around 88 °C [162]. Furthermore, the structure of the 23S subunit exhibits an internal loop of size 33 closed by base pairs $1139 \cdot 1268$ and $1155 \cdot 1249$, cf. [55, 54].

Folding this sequence at 88 °C we did (almost) get the inner stem of this internal loop but the outer stem came apart as two single strands (cf. figure 7.6(b)). When lowering the temperature to 75 °C we did get both stems, but the internal loop was split into two loops of size 2 and 27, respectively, by a short stem consisting of the base pairs $1141 \cdot 1266$ and $1142 \cdot 1265$ (cf. figure 7.6(c)).

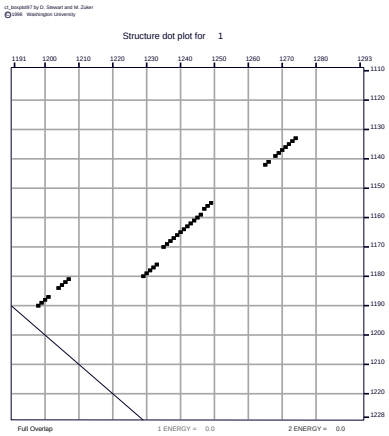
We then tried to search the range of temperatures between 75 °C and 88 °C, and at 82 °C we did in fact correctly predict the internal loop of size 33 (cf. figure 7.6(d)). At this temperature we on the other hand missed the structure



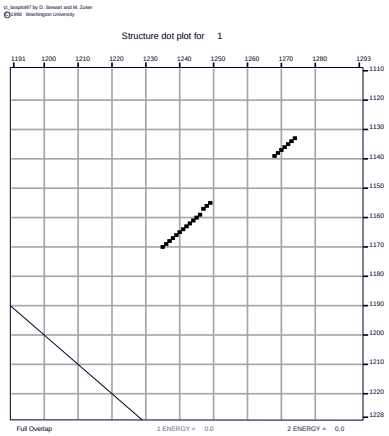
(a) Fragment of the structure between bases 1112 and 1288.



(b) Prediction of the same fragment at 88 °C.



(c) Prediction of the same fragment at 75 °C.



(d) Prediction of the same fragment at 82 °C.

Figure 7.6: Known and Predicted structures for *thermococcus celer*.

inside the inner stem, a structure that is quite well predicted at 75 °C; no temperature thus seemed decisively best for predicting this structural fragment. Generally, as with the Qβ predictions, these predictions missed long-range base pairings and predicted structures consisting of fragments covering less than 300 bases.

It should however be mentioned that a prediction at 82 °C with a cutoff size of 30 completely misses the outer stem and thus makes a prediction of this

fragment identical to the prediction at 88 °C. Thus we get a decisively better prediction at this temperature when examining internal loops of all sizes than when using a cutoff size of 30.

7.6 Discussion

It is well known that heuristics may speed up the evaluation of internal loops in practice. One way to do this, is for all subsequences to keep track of the most stable structure of any of its subsequences. This is then used to cut off the evaluation of large loops closed by a specific base pair, when it is evident that they can not be more stable than the most stable structure closed by that base pair found so far.

As the method described in section 7.3 actually evaluates the internal loops closed by a specific base pair in order of decreasing size, the above heuristic can not be combined with our method. We have instead implemented a heuristic based on determining upper bounds for the free energy of the optimal multibranched loop closed by some base pair. This heuristic unfortunately does not seem to have a positive effect for sequences shorter than 1000 nucleotides, as, for all but very long sequences, the time spent determining when to stop further evaluation exceeds the time that would have been spent evaluating the rest of the loops.

It would of course be more interesting to obtain further improvements on the worst-case behaviour of the algorithm, possibly by applying some advanced search techniques similar to those described in [39]. This is not a straightforward task though, as our method has shifted the focus from the exterior (closing) base pair to the interior base pair of an internal loop. The same interior base pair might be optimal for several choices of exterior base pairs. Furthermore, the exterior base pair that yields the most stable substructure with a specific interior base pair might not even be one of them. Thus it is of no use just to search for the exterior base pair yielding the most stable substructure.

Our studies of structure predictions at high temperatures did not show an abundance of internal loops larger than the hitherto used cutoff size. There is thus no reason to suspect that predictions using this cutoff size are generally erroneous. We were however able to predict one internal loop that exceeds this size limit. Furthermore we predicted a number of internal loops with size larger than 20. This indicates that the cutoff size of 30 is probably a little bit to small for safe predictions at high temperatures. Especially if also suboptimal foldings, cf. [164], are sought for, or if calculating the partition functions as in [101], the cutoff size – if used at all – should be set somewhat higher.

Another observation is that the energy parameters estimated for higher temperatures by extrapolation of parameters experimentally determined at lower temperatures do not seem to allow for a prediction of the long range base pairings. One reason for this might be that structures at higher temperatures tend to have more unpaired bases in multibranched loops. The effect of the number of unpaired bases on the stability of multibranched loops should theoretically be logarithmic but are modelled by a linear function for reasons of computa-

tional efficiency. This might be acceptable for multibranched loops with only a few unpaired bases but becomes prohibitive as the number of unpaired bases grows.

Finally it should be mentioned that current methods for energy based RNA secondary structure prediction only consider structures that do not contain pseudo knots. Probably *the* open question of RNA secondary structure prediction is to put forth a model including pseudo knots that allows fair predictions within reasonable resources. Currently known methods suffer from either being too time- and space-consuming (time $O(n^6)$ and space $O(n^4)$ for the method presented in [125] and time $O(n^5)$ and space $O(n^3)$ for a restricted class of pseudo knots presented in [88]) or shifting the focus from stability of loops back to stability of pairs, cf. [136].

Acknowledgements

The authors would like to thank Darrin Stewart for his help with generating the figures for this article. This work was supported, in part, by NIGMS grant GM54250 to MZ.

Chapter 8

Protein folding in the 2D HP model

> – That's circular reasoning.
> – I prefer to think of it as having no loose ends.
>
> —Scott Adams, *Dilbert*

This paper describes attempts to improve on an approximation algorithm for protein structure prediction in the two dimensional HP model. The paper is based on a mandatory assignment hand-in [89], and has been published as a technical report in the BRICS report series [90].

Protein folding in the 2D HP model

Rune B. Lyngsø* Christian N. S. Pedersen†

Abstract

We study folding algorithms in the two dimensional Hydrophobic-Hydrophilic model (2D HP model) for protein structure formation. We consider three generalizations of the best known approximation algorithm. We show that two of the generalizations do not improve the worst case approximation ratio. The third generalization seems to be better, and the analysis of its approximation ratio leads to an interesting combinatorial problem.

8.1 Introduction

Proteins are polymer chains of amino acids. An interesting feature of nature is that even though there are an infinite amount of amino acids, only twenty different amino acids are used in the formation of proteins. The amino acid sequence of a protein can thus be abstracted as a string over an alphabet of size twenty. In nature proteins are of course not one dimensional strings but fold into three dimensional structures. The three dimensional structure of a protein is not static, but vibrates around an equilibrium known as the *native state*. Famous experiments by Anfinsen et al. [8] showed that a protein in its natural environment folds into, i.e. vibrates around, a unique three dimensional structure, the *native conformation*, independent of the starting conformation. The native conformation of a protein plays an essential role in the functionality of the protein, and it is widely believed that the native conformation of a protein is determined by the amino acid sequence of the protein. As experimental determination of the native conformation is difficult and time consuming, much work has been done to predict the native conformation computationally.

To predict the structure of a protein computationally it is necessary to model protein structure formation in the real system, i.e. in the proteins natural environment. A model is relevant if it reflects some of the properties of protein structure formation in the real system. One obvious property could be *visual equivalence* between the native conformations in the model and the native conformations in the real system. Another more subtle, but useful property, could be *behavioral equivalence* between protein structure formation in the model and protein structure formation in the real system. As the laws of thermodynamics state that the native state of a protein is the state of least free energy, the real

*Department of Computer Science, University of Aarhus, Ny Munkegade, 8000 Århus C, Denmark. E-mail: rlyngsoe@daimi.au.dk.

†Basic Research In Computer Science (BRICS), Center of the Danish National Research Foundation, Department of Computer Science, University of Aarhus, Ny Munkegade, 8000 Århus C, Denmark. E-mail: cstorm@brics.dk.

system is often modeled by a *free energy model* that specifies an energy function that assigns a free energy to every conformation in a set of legal conformations. The native conformation of a protein is then predicted to be a conformation that minimizes the energy function over the set of legal conformations.

The hydrophobic-hydrophilic model proposed by Dill [33] is a free energy model that models the belief that a major contribution to the free energy of the native conformation of a protein is due to interactions between hydrophobic amino acids that tend to form a core in the spatial structure shielded from the surrounding solvent by hydrophilic amino acids. In the model the amino acid sequence of a protein is abstracted as a binary sequence of hydrophobic and hydrophilic amino acids. Even though some amino acids cannot be classified clearly as being either hydrophobic or hydrophilic, the model disregards this fact to achieve simplicity. The model is usually referred to as the HP model where H stands for hydrophobic and P stands for polar.

The HP model is a *lattice model*, so called because the set of legal conformations is embeddings of the abstracted amino acid sequence in a lattice, in this case the two or three dimensional square lattice. In legal conformations amino acids that are adjacent in the sequence occupy adjacent grid points in the lattice, and no grid point in the lattice is occupied by more than one amino acid. Depending on the dimension of the square lattice we refer to the model as the 2D or 3D HP model. The free energy of a conformation depends on the number of non-adjacent hydrophobic amino acids that occupy adjacent grid points in the lattice. Figure 8.1 shows a conformation in the 2D HP model where 9 non-adjacent hydrophobic amino acids occupy adjacent grid points.

Despite the simplicity of the HP model, the folding process in the model have behavioral similarities with the folding process in the real system [34], and the model has been used by chemists to evaluate new hypothesis of protein structure formation [129]. The success of the HP model as a tool for chemists partly stems from the fact that the discrete set of legal conformations makes it possible to enumerate and consider all conformations of small proteins. Many attempts have been made to predict the native conformation, i.e. the conformation of lowest free energy, of a protein in the HP model [146, 160]. Most interestingly, the HP model was the first relevant model for protein folding for which approximation algorithms for the structure prediction problem, i.e. algorithms that find a conformation with free energy guaranteed close to the free energy of the native conformation, were formulated [57]. For a while it was believed that the structure prediction problem in the HP model would be solvable in polynomial time, but recently it was shown NP-complete [17, 28].

In this paper we describe three attempts to improve the best known approximation algorithm for the structure prediction problem in the 2D HP model. We show that two generalizations of this algorithm, the U-fold algorithm and S-fold algorithm, do not improve on the best known 1/4 worst case approximation ratio. The approximation ratio of the third generalization, the C-fold algorithm, seems to be better. We prove that the worst case approximation ratio of the C-fold algorithm is at most 1/3 and observe that it is closely related to an interesting combinatorial problem which we examine experimentally. Most of the work described in this paper was done in the Spring 1996 as part of a

graduate course [89]. Independently of our work Mauri et al. [100] observe experimentally that the approximation ratio of an algorithm similar to our C-fold algorithm seems to be around 3/8.

The rest of this paper is organized as follows. In Section 8.2 we formally describe the 2D HP model and bound the free energy of the native conformation of a protein in the model. In Section 8.3 we describe three attempts to improve the currently best approximation algorithm for the structure prediction problem in the 2D HP model. In Section 8.4 we describe and examine experimentally an interesting problem that is related to the approximation ratio of one of the approximation algorithms described in Section 8.3.

8.2 The 2D HP model

In the 2D HP model a protein, i.e. an amino acid sequence, is abstracted as a string describing the hydrophobicity of each amino acid in the sequence. Throughout this paper we will use S to denote the abstraction of an amino acid sequence of length n, that is, S is a string of length n over the alphabet $\{0, 1\}$ where $S[i]$, for $i = 1, 2, \ldots, n$, is 1 if the ith amino acid in the sequence is hydrophobic and 0 if it is hydrophilic. We will use the term "hydrophobic amino acid" to refer to a 1 at some position in S, and say that the parity of the 1 is even if its position in S is even, and odd if its position in S is odd.

A folding of a protein in the 2D HP model is an embedding of its abstraction S in the 2D square lattice such that adjacent characters in S occupy adjacent grid points in the lattice, and no grid point in the lattice is occupied by more than one character. We say that two 1's in S form a non-local 1-1 bond if they occupy adjacent grid points in the lattice but are not adjacent in S. Figure 8.1 shows a folding of the string 111010100101001001 in the 2D HP model with nine non-local 1-1 bonds. The free energy of a folding of S is the number of non-local 1-1 bonds in the folding multiplied by some constant $\epsilon < 0$. The free energy function models the belief that the driving force of protein structure formation is interactions between hydrophobic amino acids.

We say that the *score* of a folding of S is the number of non-local 1-1 bonds in it, and that the *optimal score* of a folding of S, $\mathrm{OPT}(S)$, is the maximum score of a folding of S. The simple energy function implies that the native conformation of a protein in 2D HP model is a folding of its abstraction with optimal score. The *structure prediction problem* in the 2D HP model is thus to find a folding of S in the 2D square lattice with optimal score. This problem has recently been shown to be NP-complete [17, 28], which makes it interesting to look for approximation algorithms that find a folding of S with score guaranteed to be some fraction of the optimal score of a folding of S. To issue such a guarantee for a folding algorithm, we need an upper bound on $\mathrm{OPT}(S)$. To derive an upper bound on $\mathrm{OPT}(S)$ we make two observations.

The first observation is that a hydrophobic amino acid can form at most two non-local 1-1 bonds in the 2D square lattice except if it is the first or the last amino acid in the sequence, in which case it can form at most three non-local 1-1 bonds. The second observation is that two hydrophobic amino acids, $S[i]$

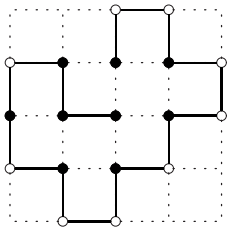


Figure 8.1: A conformation in the 2D HP model with 9 non-local 1-1 bonds.

and $S[j]$, can occupy adjacent grid points in the 2D square lattice, i.e. form a non-local 1-1 bond, if and only if i is even j is odd or vice versa. If we define $\text{EVEN}(S)$ as the set of even positions in S containing a hydrophobic amino acid, i.e. $\{i \mid i \text{ is even and } S[i] = 1\}$, and $\text{ODD}(S)$ as the set of odd positions in S containing a hydrophobic amino acid, i.e. $\{i \mid i \text{ is odd and } S[i] = 1\}$, then the two observations gives

$$\text{OPT}(S) \leq 2 \cdot \min\{|\text{EVEN}(S)|, |\text{ODD}(S)|\} + 2. \tag{8.1}$$

This upper bound was first derived by Hart and Istrail [57], who used it in the performance analysis of a simple folding algorithm that guarantees a folding with score 1/4 of the optimal score. This algorithm and various attempts to improve it is the topic of the next section.

8.3 The folding algorithms

A simple strategy for folding a string in the 2D square lattice is to find a suitable folding point that divides the string into two parts, a prefix and a suffix, that we fold against each other. This creates a "U" structure in which non-local 1-1 bonds can be formed between 1's on opposite stems of the "U". Loops protruding from the two stems of the "U" can be used to increase the number of non-local 1-1 bonds between the stems by contracting parts of the stems. We say that a folding created this way is a U-fold. Figure 8.2 shows a schematic U-fold and the left part of Figure 8.3 shows a U-fold of the string 100100101001010100011 with four non-local 1-1 bonds between the stems and five non-local 1-1 bonds in total.

Hart and Istrail [57] present a folding algorithm that computes a U-fold of S with a guaranteed number of non-local 1-1 bonds between the stems. By a simple argument they show that the folding point can always be chosen such that at least half of the 1's with position in $\text{EVEN}(S)$ are on one stem and at least half of the 1's with position in $\text{ODD}(S)$ are on the other stem. Since there is an odd number of characters between any two characters in S with positions in either $\text{EVEN}(S)$ or $\text{ODD}(S)$, loops can be used to contract each stem such that every second character on the contracted stem is a 1 with even or odd parity depending on the stem. As each contracted stem contains

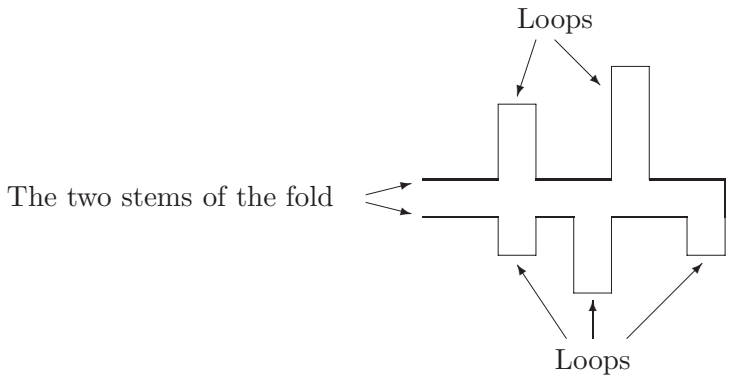


Figure 8.2: A schematic U-fold.

at least $\min\{|\text{EVEN}(S)|, |\text{ODD}(S)|\}/2$ 1's with equal parity placed in every second position along stem, the number of non-local 1-1 bonds between the stems of the created U-fold is at least $\min\{|\text{EVEN}(S)|, |\text{ODD}(S)|\}/2$, so except for a constant term the created U-fold scores at least 1/4 of the upper bound on $\text{OPT}(S)$ given by (8.1). We say that the asymptotic approximation ratio of the algorithm is 1/4. By being a little bit more careful in the choice of folding point Hart and Istrail are able to formulate the folding algorithm such that the create a U-fold, for every string S, scores at least 1/4 of the upper bound on $\text{OPT}(S)$. We say that the absolute approximation ratio of the algorithm is 1/4. The folding algorithm runs in time $O(n)$ where n is the length of S.

Our first attempt to improve the approximation ratio of the folding algorithm by Hart and Istrail, is to count all non-local 1-1 bonds between the two stems of the U-fold, and not only those where the 1's on each stem have equal parity. More precisely, we want to compute a U-fold of S with the maximum number of non-local 1-1 bonds between the stems, i.e. a U-fold of S with optimal score between the stems. Computing such a U-fold is not difficult. As illustrated in Figure 8.3, the trick is to observe that a U-fold of S, with folding point k, that maximizes the number of non-local 1-1 bonds between the stems, corresponds to the an alignment of the prefix $S[1\,..\,k-1]$ with the reversed suffix $S[k+2\,..\,n]^R$ that maximizes the number of matches between 1's, and allows gaps to be folded as loops.

Such an alignment corresponds to an optimal similarity alignment between $S[1\,..\,k-1]$ and $S[k+2\,..\,n]^R$, where a match between two 1's score 1, and all other matches and gaps score 0. To allow gaps to be folded out as loops, all gaps must have even length and between any two gaps in the same string there must be at least two matched characters. These additional rules on gaps can be enforced without increasing the running time of the alignment algorithm, so a U-fold of S with folding point k and optimal score between the stems can be computed in the time required to compute an optimal similarity alignment, i.e. in time $O(n^2)$ where n is the length of S. By considering every folding point this immediately gives an algorithm, the U-fold algorithm, that computes a U-fold with optimal score between the stems in time $O(n^3)$. By observing

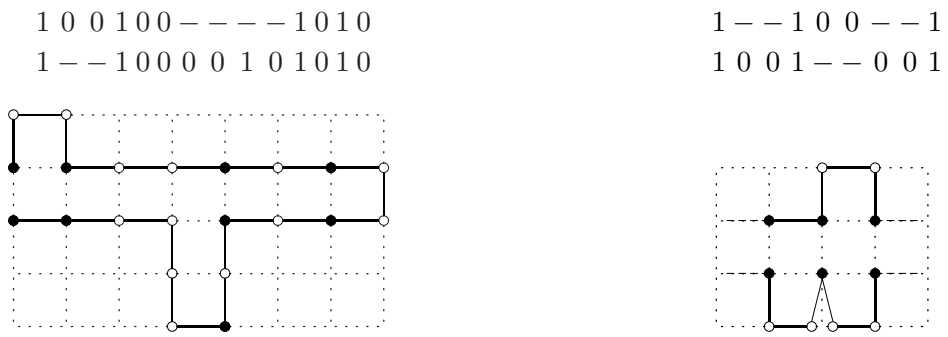
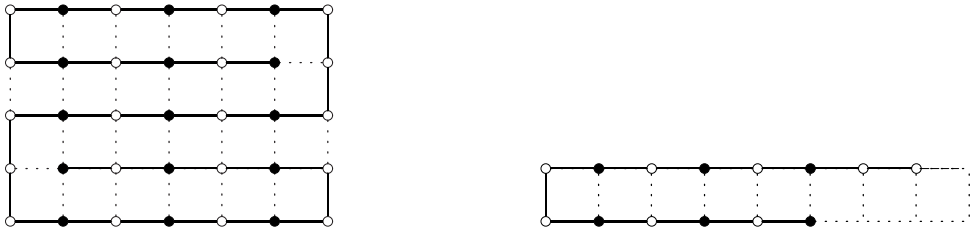


Figure 8.3: Left: Alignment of the prefix 1001001010 of the string 10010010100101010000011 with the rest of the string and the corresponding U-fold. Right: An example of an alignment with illegal gaps. The transformation to a folding implies that two loops protrude from the same element.

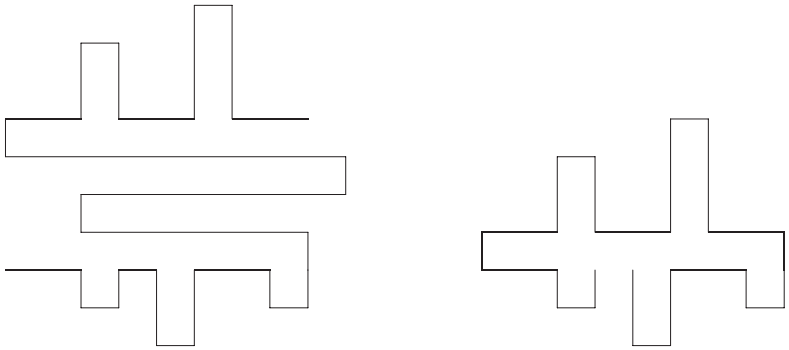


An optimal folding A U-fold

Figure 8.4: A string of the form $(10)^i0(10)^i00(10)^i00(10)^i(01)^i$. For these strings the U-fold with optimal score between the stems is only 1/4 of the score of the optimal folding.

that the best folding point k corresponds to an entry $(k-1, n-k-1)$ with maximum value in the alignment matrix resulting from an alignment of S and S^R with the above parameters, i.e. matches between 1's score 1, everything else score 0, and gaps have to be expressible as loops, we can reduce the running time of the U-fold algorithm to $O(n^2)$.

As the foldings considered by the folding algorithm by Hart and Istrail are a subset of the foldings considered by our U-fold algorithm, the approximation ratio of the U-fold algorithm is at least 1/4. Unfortunately it is no better in the worst case. As illustrated in Figure 8.4, this follows because any string of the form $(10)^i0(10)^i00(10)^i(01)^i$, $i > 0$, when folded as a U-fold with optimal score between the stems only scores 1/4 of the score of an optimal folding. The 1/4 approximation ratio of our U-fold algorithm and the folding algorithm by Hart and Istrail is thus tight. An obvious way to try to improve the approximation ratio of the U-fold algorithm would be to also count and maximize the number of non-local 1-1 bonds occurring between 1's on the loops. Unfortunately, as above, a set of strings can be constructed such that when folded this way they only score 1/4 of the score of an optimal fold.



(a) S-fold (b) C-fold

Figure 8.5: Two ways to generalize the U-fold.

Another way to try to improve the approximation ratio of the U-fold algorithm is to consider a larger set of foldings than U-folds. Figure 8.5 illustrates two ways to do this. The first way is to allow multiple bends of the string and loops on the outer stems. This gives rise to what we call S-folds. The second way is to allow two bends of the string that fold the two ends of the string towards each other and loops on the two stems. This gives rise to what we call C-folds. Both the S-fold and the C-fold with optimal score between the stems can be computed in time $O(n^3)$ using dynamic programming. For the C-fold it is easy to see how. A C-fold of S is a U-fold of a prefix, $S[1\,..\,k]$, and a U-fold of a suffix, $S[k+1\,..\,n]$, glued together to form a C-fold. As there are less than n ways to divide the string, the best C-fold can be found by computing and gluing together $2n$ U-folds. As each of these U-folds can be computed in time $O(n^2)$, the best C-fold can be computed in time $O(n^3)$. The computation of the best S-fold in time $O(n^3)$ is somewhat more technical. We choose to omit the details of the S-fold algorithm as it, as explained below, unfortunately turns out that its approximation ratio is no better than 1/4.

As S- and C-folds are supersets of U-folds, the approximation ratio of both the S- and C-fold algorithm is at least 1/4. Unfortunately this approximation ratio is tight for the S-fold algorithm because any string of the form $(10)^i(0^{2i+1}1)^{4i}(10)^i$, $i > 0$, when folded as a S-fold with optimal score between the stems only scores 1/4 of the score of an optimal folding. Similar to U-folds, we can show that counting and maximizing the number of non-local 1-1 bonds occurring between 1's on the loops of the S-fold does not improve the worst case approximation ratio of the folding algorithm. In contrast to U- and S-folds, we have not been able to find a set of strings that show that the 1/4 approximation ratio of the C-fold algorithm is tight. In fact experiments indicates, as explained in the next section, that the approximation ratio of the C-fold algorithm is somewhat better than 1/4. This is also observed in [100].

In our analysis of the approximation ratio of the C-fold algorithm we came up with a relation to an interesting matching problem. This is the topic of the next section. We end this section by summarizing the presented results.

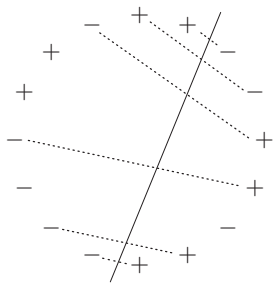


Figure 8.6: An example of a matching in a balanced string

Theorem 5 *The score of the best U- and S-fold of string S is at least, and at most in the worst case, $1/4$ of the score of an optimal fold of S. The score of the best C-fold of string S is at least $1/4$ of the score of the optimal fold of S.*

8.4 The circle problem

Let $P \in \{+,-\}^*$ be a string that contains equally many $+$'s and $-$'s. We say that P is a balanced string of length $n = |P|$. Consider P wrapped around the perimeter of a circle. A matching in P is obtained by dividing the circle by a line and connecting $+$'s with $-$'s using non-crossing lines that all intersect the dividing line. The size of the matching is the number of non-crossing lines connecting $+$'s with $-$'s that intersect the dividing line. Figure 8.6 shows an example of a matching of size 6. A maximum matching in P is a matching in P of maximum size. We use $M(P)$ to denote the size of a maximum matching in P and we use $M(n)$ to denote the minimum of $M(P)$ over all balanced strings P of length n, that is

$$M(n) = \min_{P:|P|=n} M(P).$$

The matching problem in balanced strings, or the circle problem as we call it, is closely related to the approximation ratio of our C-fold algorithm. To see the relation, we introduce the parity labelling of a string.

The *parity labelling* of a string $S \in \{0,1\}^*$ is a string $P_S \in \{+,-\}^*$ in which the ith character indicates the parity of the ith 1 in S, e.g. the parity labelling of 100101110101 is $-++-+++$. A *balanced* parity labelling of S is a maximum length subsequence of P_S that contains equally many $+$'s and $-$'s. From the definition of $\mathrm{EVEN}(S)$ and $\mathrm{ODD}(S)$ follows that P_S contains $|\mathrm{EVEN}(S)|$ $+$'s and $|\mathrm{ODD}(S)|$ $-$'s, so a balanced parity labelling of S is obtained by removing $\big||\mathrm{EVEN}(S)| - |\mathrm{ODD}(S)|\big|$ $+$'s or $-$'s from P_S. The length of a balance parity labelling of S is $2 \cdot \min\{|\mathrm{EVEN}(S)|, |\mathrm{ODD}(S)|\}$, but the labelling is not unique as there can be several ways to choose the $+$'s or $-$'s to remove from P_S, e.g. the parity labelling $-+++-+++$ gives $-++-$, $-+-+$ and $--++$ as possible balanced parity labellings.

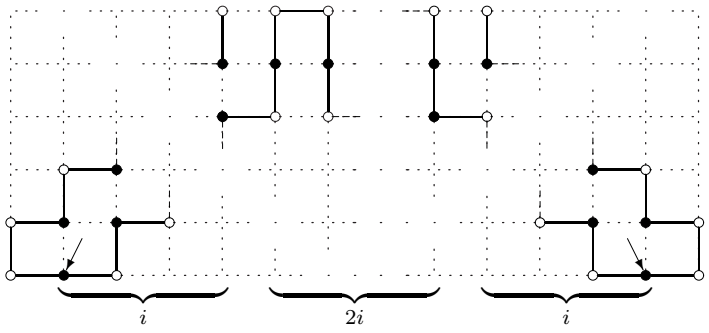


Figure 8.7: A folding of a string of the form $(01)^i000(01)^i(010)^{2i}(10)^i000(10)^i$. Only the two 1's indicated with arrows have less than the optimal two non-local bonds. The total number of non-local bonds in this folding is $2 \cdot \min\{|\text{EVEN}(S)|, |\text{ODD}(S)|\} - 1 = 4i - 1$ and thus, by the balanced parity labelling argument, the optimal score between the stems in a C-fold of this string is approximately 1/3 of the score of the optimal folding.

Upper bounding the C-fold approximation ratio

To get the relation to C-folds, we observe that a C-fold of S with k non-local 1-1 bonds between the stems corresponds to a matching of size k in a balanced parity labelling of S. This implies that an upper bound on $M(n)$ is also an upper bound on the score between the stems of C-folds of strings with balanced parity labellings of length n. In other words, if $M(n) \leq \alpha n$ then the score between the stems of a C-fold of S is upper bounded by α multiplied by the length of a balanced parity labelling of S, i.e. upper bounded by $\alpha \cdot 2 \cdot \min\{|\text{EVEN}(S)|, |\text{ODD}(S)|\}$. Since the length of a balanced parity labelling of S is equal to the upper bound on $\text{OPT}(S)$ given by (8.1), $M(n) \leq \alpha n$ implies that the approximation ratio of the C-fold algorithm, with respect to the upper bound on $\text{OPT}(S)$ given by (8.1), is at most α.

It is easy to prove that $M(+^i -^i (+-)^i -^i +^i) = 2i + 1$ for any $i > 0$. Hence, $M(n) \leq n/3 + 1$, so the asymptotic approximation ratio of our C-fold algorithm is at most 1/3 if analyzed with respect to the upper bound on $\text{OPT}(S)$ given by (8.1). Fortunately, as illustrated in Figure 8.7, for any $i > 0$ there exists a string $(01)^i000(01)^i(010)^{2i}(10)^i000(10)^i$ with balanced parity labelling $+^i-^i(+-)^i-^i+^i$ for which $\text{OPT}(S)$ deviates from the upper bound of (8.1) by at most a constant term. This example shows that the asymptotic approximation ratio of the C-fold algorithm is at most 1/3.

Lower bounding the C-fold approximation ratio

To use a matching in a balanced parity labelling of S to improve on the 1/4 approximation ratio of the C-fold algorithm, two requirements must be met. First, we need to be able to transform a matching in a balanced parity labelling of S into a C-fold of S with a number of non-local bonds proportional by some factor β to the size of the matching. Secondly, we need to lower bound the

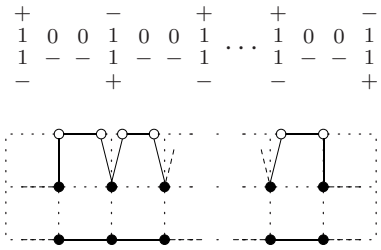


Figure 8.8: An example where the obvious transformation from a matching of a balanced parity labelling of a string to a C-fold, trying to place two 1's with connected labels opposite each other on the stems of a C-fold, fails.

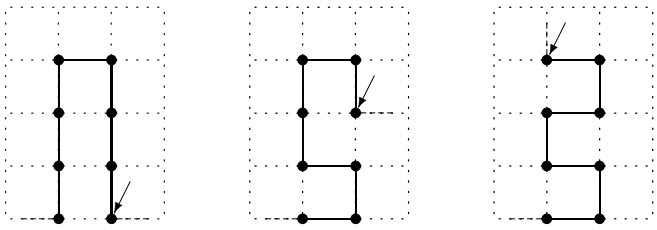


Figure 8.9: Possible hydrophobic loops of eight consecutive 1's. The positions of the embedding of the last 1 in the stretch is indicated with an arrow.

asymptotic ratio of $M(n)/n$ by some constant $\gamma > 1/(4 \cdot \beta)$. This would yield an asymptotic approximation ratio of the C-fold algorithm of $\beta \cdot \gamma > 1/4$. We have not yet solved these problems but will in the following report on some promising approaches and experiments.

The task of transforming a matching in a balanced parity labelling of S to a C-fold of S is not as straightforward as transforming the non-local bonds between the stems of a C-fold of S to a matching in one of the balanced parity labellings of S. Though one can identify the labels with 1's it will not always be the case that there is a legal C-fold of S where the non-local bonds between the stems corresponds to the connections between the corresponding labels in a matching in a balanced parity labelling of S.

To observe this, consider the two strings $S' = 1^{2i}$ and $S'' = (100)^{2i-1}1$, both with balanced parity labellings $P_{S'} = P_{S''} = (-+)^i$. Assume that S contains S' and S'' as substrings and that the labels of these two substrings have been connected with each other in the matching in a balanced parity labelling of S. As illustrated in Figure 8.8, we get the same problem as in the right-hand example in Figure 8.3 with two loops protruding from the same element if we try to make the obvious transformation of this matching to a C-fold of S. We observe that the obvious transformation only fails when we have stretches of consecutive 1's in one of the stems. One approach to solve the problem of transforming a matching in the balanced parity matching of S to a C-fold of S would thus be to 'eliminate' or at least 'shorten' consecutive stretches of 1's by removing 1's while ensuring compensatory non-local bonds.

This can be done in much the same way as when contracting the stems of a C-fold by folding out loops. As illustrated in Figure 8.9 we can fold out a stretch of an even number of consecutive 1's in a hydrophobic loop such that only two 1's remains along the stem. In such a loop where $2i$ 1's have been removed, i of which are at positions in EVEN(S) and i of which are at positions in ODD(S), there will be i non-local bonds. As long as $\beta \cdot \gamma \leq 1/2$ we can thus ensure compensatory non-local bonds. This allows us to remove the 1's that can be folded out in hydrophobic loops from S before finding a matching of a balanced parity labelling of the modified sequence.

Two problems still remain, though. First, the hydrophobic loops make the sequence less flexible since we cannot contract the stems immediately after a hydrophobic loop simply by folding out another loop. As indicated in Figure 8.9 we can however choose the position of the embedding of the last 1 in the stretch of 1's folded out rather freely which allows almost any contracting by an even number of amino acids immediately after a hydrophobic loop. Only when the loop removes $2i$ 1's with i odd there is a problem with contracting the stem by $i + 1$ amino acids as we have to round the corner of the loop from the position furthest away from the stem where we can embed the last 1. Secondly, we have not eliminated stretches of consecutive 1's but merely limited them to being of length at most three. Though we find this approach promising we have not yet been able to carry through with the rigorous case-by-case analysis, an analysis that will require additional tricks besides the hydrophobic loops to handle special cases, of the various situations that can arise when trying to transform a matching of a balanced parity labelling of S to a C-fold of S.

To lower bound the asymptotic ratio of $M(n)/n$ it is easy to observe that $M(n) \geq n/4$. Unless we, unrealistically, hope to transform a matching in the balanced parity labelling of S into a C-fold of S with *more* non-local bonds than connections in the matching this lower bound does not say anything we do not already known, namely that the approximation of the C-fold algorithm is at least 1/4. Narrowing the gap between the trivial lower bound $M(n) \geq n/4$ and the upper bound $M(n) \leq n/3 + 1$ presented above has turned out to be a very difficult problem.

To get an impression of whether or not the trivial lower bound is tight, we did two experiments. First, we computed the value of $M(n)$ for all $n \leq 34$. As illustrated in Figure 8.10, this showed that $M(n) \geq n/3$ for all $n \leq 34$. Secondly, we computed $M(n)$ for a large number of randomly selected larger balanced strings. This random search did not produce a string in which the size of the maximum matching was less than $n/3$. Combined these two experiments lead us to believe that $M(n) \geq n/3$.

To help prove a non-trivial lower bound, one might consider the restricted matching problem where the dividing line must be chosen such that it divides the circle into two halfs. This restriction does not seem to affect the lower bound, as rerunning the experiment presented in Figure 8.10 gives the same results. It might also be helpful to consider other formulations of the problem. We observe that a dividing line in the circular representation of P corresponds to a partition XYZ of P, where the one side of the divided circle is Y and the other side is ZX. The maximum size of a matching in P given a partition XYZ

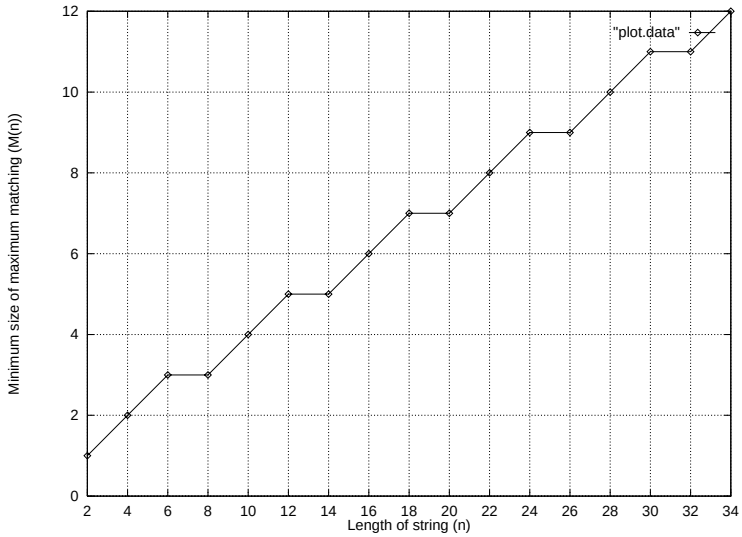


Figure 8.10: The minimum size of the maximum matching in balanced strings with length up to 34.

is the length of the longest common subsequence of Y and $\overline{ZX}^R$, so

$$M(P) = \max_{XYZ\,:\,P=XYZ} |LCS(Y, \overline{ZX}^R)|.$$

In this terminology the above restriction of the problem, i.e. that the circle should be divided into two halfs, corresponds to only maximizing over partitions XYZ of P where $|Y| = |ZX|$. Another formulation of the problem follows from the observation that part of $LCS(Y, \overline{ZX}^R)$ is a subsequence of a prefix Y and $\overline{X}^R$ and the rest is a subsequence of the rest of Y and $\overline{Z}^R$. We can thus split Y according to this and reformulate the calculation of $M(P)$ as

$$M(P) = \max_{X_1,X_2} \{|X_1| + |X_2| \mid X_1\overline{X_1}^R X_2\overline{X_2}^R \text{ is a subsequence of P}\}.$$

This lends an immediate generalization of the problem as we can define

$$M_k(P) = \max_{X_1,\ldots,X_k} \{\sum_{i=1}^{k} |X_i| \mid X_1\overline{X_1}^R \ldots X_k\overline{X_k}^R \text{ is a subsequence of P}\},$$

where $M(P) = M_2(P)$ and $M_1(P)$ is the corresponding problem for the U-fold (equivalent to fixing one end-point of the dividing line in the circle formulation of the problem). One can observe that $M_k(n) < n/2$ for any k because the string $P = + + + - - + --$ gives that $M_k(P) = M_1(P) = 3$, but apart from this we have not been able to come up with any non-trivial bounds for $M_k(n)$.

8.5 Conclusion

We have presented three generalizations of the best known approximation algorithm for structure prediction in the 2D HP model. We have shown that two of

these generalization do not improve the worst case approximation ratio, while the third generalization might be better. The future work is clear. First, prove that a matching in a balanced string can be transformed to a C-fold with score equal to the size of the matching. Secondly, prove or disprove that $M(n) \geq \alpha n$ for some $\alpha > 1/4$. Combined this would give whether or not our C-fold algorithm improves the best known 1/4 approximation ratio for structure prediction in the 2D HP model. We conjecture that the approximation ratio of our C-fold algorithm where non-local bonds in the loops are considered is 1/3.

Bibliography

[1] HMMER. Available at http://hmmer.wustl.edu/ from Sean Eddy's Lab.

[2] HMMpro. Available at http://www.netid.com/html/hmmpro.html from Net-ID.

[3] SAM. Available at http://www.cse.ucsc.edu/research/compbio/sam.html from the Computational Biology group at the University of California, Santa Cruz.

[4] WWWebster dictionary. Available at http://www.m-w.com/.

[5] D. Adams. *The Hitch Hiker's Guide to the Galaxy*. William Heinemann Ltd., London, UK, 1993. A Trilogy in Four Parts.

[6] G. M. Adel'son-Vel'skii and Y. M. Landis. An algorithm for the organization of information. *Doklady Akademii Nauk SSSR*, 146:263–266, 1962. English translation in *Soviet Math. Dokl.*, 3:1259–1262.

[7] R. A. Alberty and R. J. Silbey. *Physical Chemistry*. John Wiley & Sons, Inc., 1st edition, 1992.

[8] C. B. Anfinsen, E. Haber, and F. H. White. The kinetics of the formation of native ribonuclease during oxidation of the reduced polypetide domain. *Proceedings of the National Academy of Sciences of the United States of America*, 47:1309–1314, 1961.

[9] A. Apostolico and F. P. Preparata. Optimal off-line detection of repetitions in a string. *Theoretical Computer Science*, 22:297–315, 1983.

[10] L. Arvestad. Aligning coding DNA in the presence of frame-shift errors. In *Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 1264 of *Lecture Notes in Computer Science*, pages 180–190, 1997.

[11] K. Asai, S. Hayamizu, and K. Onizuka. Prediction of protein secondary structure by the hidden Markov model. *CABIOS*, 9(2):141–146, 1993.

[12] V. Bafna, E. L. Lawler, and P. A. Pevzner. Approximation algorithms for multiple sequence alignment. In *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching (CPM)*, Lecture Notes in Computer Science, pages 43–53, 1994.

[13] R. Bakis. Continuous speech word recognition via centisecond acoustic states. In *Proceedings of the ASA Meeting*, Apr. 1976.

[14] P. Baldi, Y. Chauvin, T. Hunkapiller, and M. A. McClure. Hidden Markov models of biological primary sequence information. *Proceedings of the National Academy of Sciences of the United States of America*, 91:1059–1063, 1994.

[15] C. Barrett, R. Hughey, and K. Karplus. Scoring hidden Markov models. *CABIOS*, 13(2):191–199, 1997.

[16] S. A. Benson, M. N. Hall, and T. J. Silhavy. Genetic analysis of protein export in *Escherichia coli* K12. *Annual Review of Biochemistry*, 54:101–134, 1985.

[17] B. Berger and T. Leighton. Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete. *Journal of Computational Biology*, 5(1):27–40, 1998.

[18] J. U. Bowie and D. Eisenberg. An evolutionary approach to folding small alpha-helical proteins that uses sequence information and an empirical guiding fitness function. *Proceedings of the National Academy of Sciences of the United States of America*, 91(10):4436–4440, 1994.

[19] G. S. Brodal, R. B. Lyngsø, C. N. S. Pedersen, and J. Stoye. Finding maximal pairs with bounded gap. In *Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 1645 of *Lecture Notes in Computer Science*, pages 134–149, 1999.

[20] G. S. Brodal, R. B. Lyngsø, C. N. S. Pedersen, and J. Stoye. Finding maximal pairs with bounded gap. Technical Report RS-99-12, BRICS, Apr. 1999.

[21] M. P. Brown, R. Hughey, A. Krogh, I. S. Mian, K. Sjölander, and D. Haussler. Using Dirichlet mixture priors to derive hidden Markov models for protein families. In L. Hunter, D. Searls, and J. Shavlik, editors, *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 47–55, Menlo Park, California, U.S.A., July 1993. AAAI/MIT Press.

[22] M. R. Brown and R. E. Tarjan. A fast merging algorithm. *Journal of the ACM*, 26(2):211–226, 1979.

[23] K. Bucka-Lassen, O. Caprani, and J. J. Hein. Combining many multiple alignments in one improved alignment. *Bioinformatics*, 15(2):122–130, 1999.

[24] J. M. Butler and D. J. Reeder. Background information on STRs. http://ibm4.carb.nist.gov:8800/dna/intro.htm, May 1999.

[25] H. Carrillo and D. Lipman. The multiple sequence alignment problem in biology. *SIAM Journal on Applied Mathematics*, 48:1073–1082, 1988.

[26] J.-H. Chen, S.-Y. Le, and J. V. Maizel. A procedure for RNA pseudoknot prediction. *CABIOS*, 8(3):243–248, 1992.

[27] G. A. Churchill. Stochastic models for heterogeneous DNA sequences. *Bulletin of Mathematical Biology*, 51:79–94, 1989.

[28] P. Crescenzi, D. Goldman, C. Papadimitriou, A. Piccolboni, and M. Yannakakis. On the complexity of protein folding. *Journal of Computational Biology*, 5(3):423–465, 1998.

[29] M. Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, 12(5):244–250, 1981.

[30] M. Crochemore. Transducers and repetitions. *Theoretical Computer Science*, 45:63–86, 1986.

[31] J. Cupal, I. L. Hofacker, and P. F. Stadler. Dynamic programming algorithm for the density of states of RNA secondary structures. In Hofstädt, T. Lengauer, M. Löffler, and D. Schomburg, editors, *Computer Science and Biology 96 (Prooceedings of the German Conference on Bioinformatics)*, pages 184–186, University of Leipzig, 1996.

[32] P. C. W. Davies and D. S. Betts. *Quantum Mechanics*. Number 8 in Physics and Its Applications. Chapman & Hall, 2–6 Boundary Row, London, United Kingdom, second edition, 1994.

[33] K. A. Dill. Theory for the folding and stability of globular proteins. *Biochemistry*, 24:1501, 1985.

[34] K. A. Dill, S. Bromberg, K. Yue, K. M. Fiebig, D. P. Yee, P. D. Thomas, and H. S. Chan. Principles of protein folding – a perspective from simple exact models. *Protein Science*, 4:561–602, 1995.

[35] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probalistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[36] S. R. Eddy. Hidden Markov models. *Current Opinion in Structurel Biology*, 6:361–365, 1996.

[37] S. R. Eddy. Profile hidden Markov models. *Bioinformatics*, 14:755–763, 1998.

[38] S. R. Eddy and R. Durbin. RNA sequence analysis using covariance models. *Nucleic Acids Research*, 22(11):2079–2088, 1994.

[39] D. Eppstein, Z. Galil, and R. Giancarlo. Speeding up dynamic programming. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 488–496, 1988.

[40] K. Faber. *Biotransformations in Organic Chemistry – A Textbook*. Springer-Verlag, Berlin, Germany, second edition, 1995.

[41] M. Farach. Optimal suffix tree construction with large alphabets. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 137–143, 1997.

[42] R. W. Floyd. Algorithm 245: Treesort3. *Communications of the ACM*, 7(12):701, 1964.

[43] A. S. Fraenkel. Complexity of protein folding. *Bulletin of Mathematical Biology*, 55(6):1199–1210, 1993.

[44] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 338–346, 1984.

[45] J. Gorodkin, L. J. Heyer, and G. D. Stormo. Finding common sequence and strucure motifs in a set of RNA sequences. In T. Gaasterland, P. Karp, K. Karplus, C. Ouzounis, C. Sander, and A. Valencia, editors, *Proceedings of the 5th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 120–123, 445 Burgess Drive, CA94025 Menlo Park, USA, 1997. AAAI Press.

[46] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162:705–708, 1982.

[47] O. Gotoh. Optimal alignment between groups of sequences and its application to multiple sequence alignment. *Computer Applications in the Biociences*, 9(3):361–370, 1993.

[48] M. Gribskov, A. D. McLachlan, and D. Eisenberg. Profile analysis: Detection of distantly related proteins. *Proceedings of the National Academy of Sciences of the United States of America*, 84:4355–4358, 1987.

[49] W. Gruener, R. Giegerich, D. Strothmann, C. Reidys, J. Weber, I. L. Hofacker, P. F. Stadler, and P. Schuster. Analysis of RNA sequence structure maps by exhaustive enumeration. *Monatshefte für Chemie*, 127:355–389, 1996.

[50] L. J. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 8–21, 1978.

[51] S. K. Gupta, J. D. Kececioglu, and A. A. Schäffer. Making the shortest-paths approach to sum-of-pairs multiple sequence alignment more space efficient in practice (extended abstract). In Z. Galil and E. Ukkonen, editors, *Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 937 of *Lecture Notes in Computer Science*, pages 128–143, Espoo, Finland, 5-7 July 1995. Springer.

[52] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.

[53] D. Gusfield and J. Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. Technical Report CSE-98-4, Department of Computer Science, UC Davis, 1998.

[54] R. R. Gutell. RNA secondary structures. Available from http://pundit.icmb.utexas.edu.

[55] R. R. Gutell, M. W. Gray, and M. N. Schnare. A compilation of large subunit (23S and 23S-like) ribosomal RNA structures. *Nucleic Acids Res.*, 21:3055–3074, 1993. Database issue.

[56] W. E. Hart and S. Istrail. Chrystallographical universal approximability: A complexity theory of protein folding algortihms on crystal lattices. Technical Report SAND95-1294, Sandia National Laboratories, aug 1995.

[57] W. E. Hart and S. Istrail. Fast protein folding in the hydrophobic-hydrophilic model within three-eights of optimal. *Journal of Computational Biology*, Spring 1996, 1996.

[58] W. E. Hart and S. Istrail. Invariant patterns in crystal lattices: Implications for protein folding algorithms (extended abstract). In D. Hirschberg and E. W. Myers, editors, *Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 1075 of *Lecture Notes in Computer Science*, pages 288–303, Laguna Beach, CA, USA, 1996. Springer Verlag.

[59] J. J. Hein. Unified approach to alignment and phylogenies. *Methods in Enzymology*, 183:626–645, 1990.

[60] J. J. Hein. An algorithm combining DNA and protein alignment. *Journal of Theoretical Biology*, 167:169–174, 1994.

[61] J. J. Hein and J. Støvlbæk. Genomic alignment. *Journal of Molecular Evolution*, 38:310–316, 1994.

[62] J. J. Hein and J. Støvlbæk. Combined DNA and protein alignment. *Methods in Enzymology*, 266:402–418, 1996.

[63] D. S. Hirschberg. A linear space algortihm for computing maximal common subsequence. *Communications of the ACM*, 18(6):341–343, 1975.

[64] I. L. Hofacker. RNA secondary structures: A tractable model of biopolymer folding. In P. Grassberger, G. Barkema, and W. Nadler, editors, *Monte Carlo Approach to Biopolymers and Protein Folding*, pages 171–182. World Scientific, Singapore, 1998.

[65] I. L. Hofacker, W. Fontana, P. F. Stadler, L. S. Bonhoeffer, M. Tacker, and P. Schuster. Fast folding and comparison of RNA secondary structures. *Monatshefte für Chemie*, 125:167–188, 1994.

[66] K. Hopkin. When RNA ruled – another lost world? *HMS Beagle, The BioMedNet Magazine*, 27, Mar. 1998. Available from http://biomednet.com/hmsbeagle/1998/27/resnews/meeting.htm.

[67] Y. Hua, T. Jiang, and B. Wu. Aligning DNA sequences to minimize the change in protein. In *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 1448 of *Lecture Notes in Computer Science*, pages 221–234, 1998.

[68] S. Huddleston and K. Mehlhorn. A new data structure for representing sorted lists. *Acta Informatica*, 17:157–184, 1982.

[69] R. Hughey and A. Krogh. Hidden Markov models for sequence analysis: Extension and analysis of the basic method. *CABIOS*, 12(2):95–107, 1996.

[70] F. K. Hwang and S. Lin. A simple algorithm for merging two disjoint linearly ordered sets. *SIAM Journal on Computing*, 1(1):31–39, 1972.

[71] A. B. Jacobson. Secondary structure of coliphage Qβ RNA. *Journal of Molecular Biology*, 221:557–570, 1991.

[72] H. Jacobson and W. H. Stockmayer. Intramolecular reaction in polycondensations. I. the theory of linear systems. *JChemPhys*, 18:1600–1606, 1950.

[73] F. Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64:532–536, Apr. 1976.

[74] D. T. Jones, W. R. Taylor, and J. M. Thornton. A new approach to protein fold recognition. *Nature*, 358:86–89, July 1992.

[75] S. Karlin, M. Morris, G. Ghandour, and M.-Y. Leung. Efficient algorithms for molecular sequence analysis. *Proceedings of the National Academy of Sciences of the United States of America*, 85:841–845, 1988.

[76] B. Knudsen and J. J. Hein. RNA secondary structure prediction using stochastic context-free grammers and evolutionary history. *Bioinformatics*, 15(6):446–454, June 1999.

[77] R. Kolpakov and G. Kucherov. Maximal repetitions in words or how to find all squares in linear time. Technical Report 98-R-227, LORIA, 1998.

[78] S. R. Kosaraju. Computation of squares in a string. In *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 807 of *Lecture Notes in Computer Science*, pages 146–150, 1994.

[79] A. Krogh. Two methods for improving performance of an HMM and their application for gene finding. In *Proceedings of the 5th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 179–186, 1997.

[80] A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235:1501–1531, 1994.

[81] J. B. Kruskal. An overview of sequence comparison. In D. Sankoff and J. B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pages 1–44. Addison-Wesley Publishing Co., Reading, Massachusetts, U.S.A., 1983.

[82] G. M. Landau and J. P. Schmidt. An algorithm for approximate tandem repeats. In *Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 684 of *Lecture Notes in Computer Science*, pages 120–133, 1993.

[83] L. L. Larmore and B. Schieber. On-line dynamic programming with applications to the prediction of RNA secondary structure. In D. Johnson, editor, *Proceedings of the 1st Annual Symposium on Discrete Algorithms (SODA)*, pages 503–512, San Francisco, CA, USA, Jan. 1990. SIAM.

[84] R. H. Lathrop. The protein threading problem with sequence amino acid interaction preferences is NP-complete. *Protein Engineering*, 7:1059–1068, 1994.

[85] R. H. Lathrop and T. F. Smith. Global optimum protein threading with gapped alignment and empirical pair score functions. *JMB*, 255:641–665, 1996.

[86] M.-Y. Leung, B. E. Blaisdell, C. Burge, and S. Karlin. An efficient algorithm for identifying matches with errors in multiple long molecular sequences. *Journal of Molecular Biology*, 221:1367–1378, 1991.

[87] R. Luthy, J. U. Bowie, and D. Eisenberg. Assessment of protein models with three-dimensional profiles. *Nature*, 356:83–85, 1992.

[88] R. B. Lyngsø. Computational aspects of biological sequences and structures. Master's thesis, Department of Computer Science, University of Aarhus, June 1997.

[89] R. B. Lyngsø and C. N. S. Pedersen. Prediction of protein structures using simple exact models. Project in a Graduate Course. Available from http://www.daimi.au.dk/∼rlyngsoe/protein.ps, June 1996.

[90] R. B. Lyngsø and C. N. S. Pedersen. Protein folding in the 2D HP model. Technical Report RS-99-16, BRICS, June 1999.

[91] R. B. Lyngsø, C. N. S. Pedersen, and H. Nielsen. Measures on hidden Markov models. Technical Report RS-99-6, BRICS, Apr. 1999.

[92] R. B. Lyngsø, C. N. S. Pedersen, and H. Nielsen. Metrics and similarity measures for hidden Markov models. In T. Lengauer, R. Schneider, P. Bork, D. Brutlag, J. Glasgow, H.-W. Mewes, and R. Zimmer, editors,

Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology (ISMB), pages 178–186, Heidelberg, Germany, 1999. AAAI Press, Menlo Park, CA94025, U.S.A.

[93] R. B. Lyngsø, M. Zuker, and C. N. S. Pedersen. Fast evaluation of internal loops in RNA secondary structure prediction. *Bioinformatics*, 15(6):440–445, June 1999.

[94] R. B. Lyngsø, M. Zuker, and C. N. S. Pedersen. An improved algorithm for RNA secondary structure prediction. Technical Report RS-99-15, BRICS, May 1999.

[95] R. B. Lyngsø, M. Zuker, and C. N. S. Pedersen. Internal loops in RNA secondary structure prediction. In *Proceedings of the 3rd Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 260–267, 1999.

[96] M. G. Main and R. J. Lorentz. An $O(n \log n)$ algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5:422–432, 1984.

[97] M. G. Main and R. J. Lorentz. Linear time recognition of squarefree strings. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume F12 of *NATO ASI Series*, pages 271–278. Springer, Berlin, 1985.

[98] F. Major, D. Gautheret, and R. Cedergren. Reproducing the three-dimensional structure of a tRNA molecule from structural constraints. *Proceedings of the National Academy of Sciences of the United States of America*, 90:9408–9412, Oct. 1993.

[99] D. H. Mathews, J. Sabina, M. Zuker, and D. H. Turner. Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *Journal of Molecular Biology*, 288:911–940, 1999.

[100] G. Mauri, G. Pavesi, and A. Piccolboni. Approximation algorithms for protein folding prediction. In *Proceedings of the 10th Annual Symposium on Discrete Algorithms (SODA)*, pages 945–946, 1999.

[101] J. S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29:1105–1119, 1990.

[102] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, 1976.

[103] J. McMurry. *Fundamentals of Organic Chemistry*. Brooks/Cole Publishing Company, 2nd edition, 1990.

[104] K. Mehlhorn. *Sorting and Searching*, volume 1 of *Data Structures and Algorithms*. Springer-Verlag, 1994.

[105] K. Mehlhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999. To appear. See http://www.mpi-sb.mpg.de/~mehlhorn/LEDAbook.html.

[106] W. Miller and E. W. Myers. Sequence comparison with concave weighting functions. *Bulletin of Mathematical Biology*, 50:97–120, 1988.

[107] E. W. Myers. An overview of sequence comparison algorithms in molecular biology. Technical Report TR 91-29, Department of Computer Science, University of Arizona, Dec. 1991.

[108] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:433–443, 1970.

[109] J. T. Ngo and J. Marks. Computational complexity of a problem in molecular-structure prediction. *Protein Engineering*, 5(4):313–321, 1992.

[110] H. Nielsen, S. Brunak, J. Engelbrecht, and G. von Heijne. Identification of prokaryotic and eukaryotic signal peptides and prediction of their cleavage sites. *Protein Engineering*, 10:1–6, 1997.

[111] H. Nielsen and A. Krogh. Prediction of signal peptides and signal anchors by a hidden Markov model. In J. Glasgow, T. Littlejohn, F. Major, R. Lathrop, D. Sankoff, and C. Sensen, editors, *Proceedings of the 6th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 122–133, Montréal, Québec, Canada, 1998. AAAI Press.

[112] R. Nussinov and A. B. Jacobson. Fast algorithm for predicting the secondary structure of single-stranded RNA. *Proceedings of the National Academy of Sciences of the United States of America*, 77(11):6309–6313, Nov. 1980.

[113] C. M. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, Inc., 1994.

[114] C. Papanicolaou, M. Gouy, and J. Ninio. An energy model that predicts the correct folding of both the tRNA and the 5S RNA molecules. *Nucleic Acids Research*, 12:31–44, 1984.

[115] M. Paterson and T. Przytycka. On the complexity of string folding. In *Proceedings of the 23rd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1099 of *Lecture Notes in Computer Science*, pages 658–669. Springer-Verlag, 1996.

[116] C. N. S. Pedersen, R. Lyngsø, and J. J. Hein. Comparison of coding DNA. In *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 1448 of *Lecture Notes in Computer Science*, pages 153–173, 1998.

[117] C. N. S. Pedersen, R. Lyngsø, and J. J. Hein. Comparison of coding DNA. Technical Report RS-98-3, BRICS, Jan. 1998. Also published as [116].

[118] J. T. Pedersen and J. Moult. Protein folding simulations with genetic algorithms and a detailed molecular description. *Journal of Molecular Biology*, 269(2):240–259, June 1997.

[119] H. Peltola, H. Söderlund, and E. Ukkonen. Algorithms for the search of amino acid patterns in nucleic acid sequences. *Nuclear Acids Research*, 14(1):99–107, 1986.

[120] A. E. Peritz, R. Kierzek, N. Sugimoto, and D. H. Turner. Thermodynamic study of internal loops in oligoribonucleotides: symmetric loops are more stable than asymmetric loops. *Biochemistry*, 30:6428–6436, 1991.

[121] C. W. A. Pleij. RNA pseudoknots. In R. F. Gesteland and J. F. Atkins, editors, *The RNA World*. Cold Spring Harbor Laboratory Press, 1993.

[122] S. Poe. Data set incongruence and the phylogeny of crocodilians. *Systematic Biology*, 45(4):393–414, 1996.

[123] N. Qian and T. J. Sejnowski. Predicting the secondary structure of globular proteins using neural network models. *Journal of Molecular Biology*, 202:865–884, 1988.

[124] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 277–286, 1989.

[125] E. Rivas and S. Eddy. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *Journal of Molecular Biology*, 285(5):2053–2068, 1999.

[126] B. Rost. PHD predicting 1D protein structure by profile based neural networks. *Methods in Enzymology*, 266:525–539, 1996.

[127] M.-F. Sagot and E. W. Myers. Identifying satellites in nucleic acid sequences. In *Proceedings of the 2nd Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 234–242, 1998.

[128] Y. Sakakibara, M. Brown, R. Hughey, I. S. Mian, K. Sjölander, R. C. Underwood, and D. Haussler. Stochastic context-free grammars for tRNA modeling. *Nucleic Acids Research*, 22(23):5112–5120, 1994.

[129] A. Šali, E. Shahknovich, and M. Karplus. How does a protein fold? *Nature*, 369:248–251, 1994.

[130] D. Sankoff. Matching sequences under deletion/insertion constraints. *Proceedings of the National Academy of Sciences of the United States of America*, 69:4–6, 1972.

[131] D. Sankoff. Simultaneous solution of the RNA folding, alignment and protosequence problems. *SIAM Journal on Applied Mathematics*, 45(5):810–825, Oct. 1985.

[132] P. H. Sellers. On the theory and computation of evolutionary distance. *SIAM Journal of Applied Mathematics*, 26:787–793, 1974.

[133] M. J. Sippl. The calculation of conformational ensembles from potentials of mean force. An approach to the knowledge based prediction of local structures in globular proteins. *Journal of Molecular Biology*, 213:859–883, 1990.

[134] E. L. L. Sonnhammer, G. von Heijne, and A. Krogh. A hidden Markov model for predicting transmembrane helices in protein sequences. In J. Glasgow, T. Littlejohn, F. Major, R. Lathrop, D. Sankoff, and C. Sensen, editors, *Proceedings of the 6th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, Montréal, Québec, Canada, 1998. AAAI Press.

[135] J. Stoye and D. Gusfield. Simple and flexible detection of contiguous repeats using a suffix tree. In *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 1448 of *Lecture Notes in Computer Science*, pages 140–152, 1998.

[136] J. E. Tabaska, R. B. Cary, H. N. Gabow, and G. D. Stormo. An RNA folding method capable of identifying pseudoknots and base triples. *Bioinformatics*, 14(8):691–699, 1998.

[137] R. E. Tarjan. Amortized computational complexity. *SIAM Journal on Algebraic and Discrete Methods*, 6:306–318, 1985.

[138] W. R. Taylor. Identification of protein sequence homology by consensus template alignment. *Journal of Molecular Biology*, 188:233–258, 1986.

[139] W. R. Taylor and C. Orengo. Protein structure alignment. *Journal of Molecular Biology*, 208:1–22, 1989.

[140] I. Tinoco. Structures of base pairs involving at least two hydrogen bonds. In R. F. Gesteland and J. F. Atkins, editors, *The RNA World*, pages 603–607. Cold Spring Harbor Laboratory Press, 1993.

[141] I. Tinoco, P. N. Borer, B. Dengler, M. D. Levine, O. C. Uhlenbeck, D. M. Crothers, and J. Gralla. Improved estimation of secondary structure in ribonucleic acids. *Nature New Biology*, 246:40–41, 1973.

[142] I. Tinoco, O. C. Uhlenbeck, and M. D. Levine. Estimation of secondary structure in ribonucleic acids. *Nature*, 230:362–367, 1971.

[143] D. H. Turner, N. Sugimoto, and S. M. Freier. RNA structure prediction. *Annual Review of Biophysics and Biophysical Chemistry*, 17:167–192, 1988.

[144] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14:249–260, 1995.

[145] R. Unger and J. Moult. Finding the lowest free energy conformation of a protein is a NP-hard problem: Proof and implications. *Bulletin of Mathematical Biology*, 55(6):1183–1198, 1993.

[146] R. Unger and J. Moult. Genetic algorithms for protein folding simulations. *Journal of Molecular Biology*, 231:75–81, 1993.

[147] F. H. D. van Batenburg, A. P. Gultyaev, and C. W. A. Pleij. An APL-programmed genetic algorithm for the prediction of RNA secondary structure. *Journal of Theoretical Biology*, 174(3):269–280, 1995.

[148] G. von Heijne. Signal sequences. The limits of variation. *Journal of Molecular Biology*, 184:99–105, 1985.

[149] G. von Heijne and L. Abrahmsén. Species-specific variation in signal peptide design. *FEBS Letter*, 244:439–446, 1989.

[150] J. Vuillemin. A data structure for manipulating priority queues. *Communications of the ACM*, 21(4):309–315, 1978.

[151] R. A. Wagner and M. J. Fisher. The string to string correction problem. *Journal of the ACM*, 21:168–173, 1974.

[152] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1:337–348, 1994.

[153] M. S. Waterman and T. F. Smith. Rapid dynamic programming methods for RNA secondary structure. *Advances in Applied Mathematics*, 7:455–464, 1986.

[154] J. D. Watson and F. H. C. Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 248(451):765, Apr. 1953.

[155] P. Weiner. Linear pattern matching algorithms. In *Proceedings of the 14th Symposium on Switching and Automata Theory*, pages 1–11, 1973.

[156] J. W. J. Williams. Algorithm 232: Heapsort. *Communications of the ACM*, 7(6):347–348, 1964.

[157] S. Wuchty, W. Fontana, I. L. Hofacker, and P. Schuster. Complete suboptimal folding of RNA and the stability of secondary structures. *Biopolymers*, 49:145–165, 1999.

[158] Y. Xu and E. C. Uberbacher. A polynomial-time algorithm for a class of protein threading problems. *CABIOS*, 12(6):511–517, 1996.

[159] J. Yadgari, A. Amir, and R. Unger. Genetic algorithms for protein threading. In J. Glasgow, T. Littlejohn, F. Major, R. Lathrop, D. Sankoff, and C. Sensen, editors, *Proceedings of the 6th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 193–202, Montréal, Québec, Canada, 1998. AAAI Press.

[160] K. Yue and K. A. Dill. Forces of tertiary structural organization in globular proteins. *Proceedings of the National Academy of Sciences of the United States of America*, 92:146–150, 1994.

[161] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, Dec. 1989.

[162] W. Zillig, I. Holz, D. Janekovic, W. Schäfer, and W. D. Reiter. The archaebacterium *Thermococcus celer* represents, a novel genus within the thermophilic branch of archaebacteria. *Systematic and Applied Microbiology*, 4:88–94, 1983.

[163] M. Zuker. RNA web-page at http://www.ibc.wustl.edu/~zuker/rna/-energy/node2.html.

[164] M. Zuker. On finding all suboptimal foldings of an RNA molecule. *Science*, 244:48–52, 1989.

[165] M. Zuker, D. H. Mathews, and D. H. Turner. Algorithms and thermodynamics for RNA secondary structure prediction: A practical guide. In J. Barciszewski and B. Clark, editors, *RNA Biochemistry and Biotechnology*, NATO ASI Series. Kluwer Academic Publishers, 1999. available at http://www.ibc.wustl.edu/~zuker/seqanal/.

[166] M. Zuker and D. Sankoff. RNA secondary structures and their prediction. *Bulletin of Mathematical Biology*, 46:591–621, 1984.

[167] M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Research*, 9:133–148, 1981.

Recent BRICS Dissertation Series Publications

DS-00-5 **Rune B. Lyngsø.** ***Computational Biology*****. March 2000. PhD thesis. xii+173 pp.**

DS-00-4 **Christian N. S. Pedersen.** ***Algorithms in Computational Biology*****. March 2000. PhD thesis. xii+210 pp.**

DS-00-3 **Theis Rauhe.** ***Complexity of Data Structures (Unrevised)*****. March 2000. PhD thesis. xii+115 pp.**

DS-00-2 **Anders B. Sandholm.** ***Programming Languages: Design, Analysis, and Semantics*****. February 2000. PhD thesis. xiv+233 pp.**

DS-00-1 **Thomas Troels Hildebrandt.** ***Categorical Models for Concurrency: Independence, Fairness and Dataflow*****. February 2000. PhD thesis. x+141 pp.**

DS-99-1 **Gian Luca Cattani.** ***Presheaf Models for Concurrency (Unrevised)*****. April 1999. PhD thesis. xiv+255 pp.**

DS-98-3 **Kim Sunesen.** ***Reasoning about Reactive Systems*****. December 1998. PhD thesis. xvi+204 pp.**

DS-98-2 **Søren B. Lassen.** ***Relational Reasoning about Functions and Nondeterminism*****. December 1998. PhD thesis. x+126 pp.**

DS-98-1 **Ole I. Hougaard.** ***The CLP(OIH) Language*****. February 1998. PhD thesis. xii+187 pp.**

DS-97-3 **Thore Husfeldt.** ***Dynamic Computation*****. December 1997. PhD thesis. 90 pp.**

DS-97-2 **Peter Ørbæk.** ***Trust and Dependence Analysis*****. July 1997. PhD thesis. x+175 pp.**

DS-97-1 **Gerth Stølting Brodal.** ***Worst Case Efficient Data Structures*****. January 1997. PhD thesis. x+121 pp.**

DS-96-4 **Torben Braüner.** ***An Axiomatic Approach to Adequacy*****. November 1996. Ph.D. thesis. 168 pp.**

DS-96-3 **Lars Arge.** ***Efficient External-Memory Data Structures and Applications*****. August 1996. Ph.D. thesis. xii+169 pp.**

DS-96-2 **Allan Cheng.** ***Reasoning About Concurrent Computational Systems*****. August 1996. Ph.D. thesis. xiv+229 pp.**