Graph-based Methods for the Design of DNA Computations

Vom Promotionsausschuss der

Technischen Universität Hamburg-Harburg

zur Erlangung des akademischen Grades

Doktorin der Naturwissenschaften

genehmigte Dissertation

von

Svetlana Torgasin

aus

Frunse

2012

1. Gutachter: Prof. Dr. Karl-Heinz Zimmermann
   Institut für Rechnertechnologie, Technische Universität Hamburg-Harburg

2. Gutachter: Prof. Dr. Andrew Torda
   Zentrum für Bioinformatik, Universität Hamburg

Tag der mündlichen Prüfung: 02.11.2011

Vorsitzender des Prüfungsausschusses: Prof. Dr. Ralf Möller
Institut für Softwaresysteme, Technische Universität Hamburg-Harburg

In memory of my grandmother

Abstract

DNA computing is a rapidly evolving field utilizing DNA molecules instead of silicon-based electronic units to perform calculations. The reliability of such computations strongly depends on the DNA sequences that represent units of information. Recently, the thermodynamic constraints, based on the free energy of hybridization between pairs of DNA single strands, are considered as the most reliable criterion to compose such sequences.

The purpose of this thesis is to provide a contribution to the field of finding reliable DNA sequences for the encoding of entities in mathematical problems. The developed methods make use of the nearest neighbor DNA thermodynamic model as a biological fundament. The modelling method uses graph theory. The work addresses the following issues.

The first one is a thermodynamic evaluation of a DNA encoding. The performance of predeveloped published sets *in vitro* differs for particular DNA computations that follow distinct models, since the intended reactions are not the same. The models using an encoding principle proposed by Adleman [2] imply interactions, that are not taken into account by modern strand design applications. Therefore, an evaluation method comprising additional restrictions is proposed, in order to more accurately assess the performance of a candidate DNA encoding for these models. The evaluation is performed with respect to thermodynamic restrictions and allows to find weak spots in the encoding set of DNA words.

The second issue is the prediction of the hybridization energy for a pair of DNA single strands. This comprises finding the secondary structure of the DNA/DNA complex with minimal free energy (MFE), which is usually referred to as MFE problem. The effective solution methods for this problem are the main prerequisites for the thermodynamically based DNA sequence design. Contemporary methods are based on the nearest neighbor model to assess the thermal stability of biomolecular DNA complexes and utilize the paradigm of dynamic programming to find an energetically optimal complex. In this work, a novel graph model for DNA/DNA hybridization complexes is developed. Based on this, two methods for the solution of the MFE problem using the paradigm of dynamic programming are proposed. The performance of the methods is compared with that of currently used methods for this task.

An additional issue concerns one of the basic problems in algorithmic graph theory. Namely, the all-pairs shortest path problem that comprises finding of the paths with minimal weight between each pair of nodes in a graph. There was developed a memory saving technique to perform the calculations for the particular case of bipartite graphs. It is based on transformation of the weight matrix by rules of tropical (min-plus) algebra.

Acknowledgements

With this, I would like to express my deep gratitude to all the people who supported me these years in some or other way: my “Doktorvater”, coleagues, my friends, and my family.

First of all, I thank my supervisor Professor Karl-Heiz Zimmermann, who supported this research intention and gave me enough freedom to explore various topics, which attracted my often scattered attention. I am also greatful for his valuable editorial advises. Special thanks to the second referee of this thesis, Professor Andrew Torda, for his approval of my ideas and radiating the good mood.

I am thankful to the Technical University of Hamburg-Harburg for financial support I got as a staff member. This enabled me to attain a number of advanced courses and conferences for improving my scientific skills and knowledge.

I also thank all my colleagues at the Institute for Computer Technology for friendly working environment. I like to thank Ms. Angelika Koch for her help in administrative matters of TUHH. Special thanks I owe a system administrator of ICT, Mr. Stefan Just, for his helpfulness and technical support. He spared me a lot of care of the background ”ware” (hard- and soft-). Mahwish, thank you for family atmosphere, which had lacked me sometimes.

I appreciate the contribution of my students: Björn Ole Pfannkuche, Silviya Bjankova, and Ji Youn Hong, whose works helped me to prove and shape some of the ideas presented in this thesis.

I am greatful to Margaret Parks, who kindly agreed to proofread the text as a native English speaker.

Further, I would like to thank my distant friends from the second alma mater; this year several of us managed a graduation. Your company is a good distraction from the research work and a nesessary completion to it.

Finally, I would like to express the gratitude to my family: my parents and siblings. Their care and presence has lead me so far in my life, and my achievements are also due to their help and support.

Contents

Nomenclature

A	adenine
C	cytosine
G	guanine
T	thymine
cal	calorie, k (kilo-)
l	litre
mol	mol, m (mili-)
DNA	deoxyribonucleic acid
dsDNA	double stranded DNA
ssDNA	single stranded DNA
RNA	ribonucleic acid
APSP	all-pairs shortest path
DP	dynamic programming
HPP	Hamiltonian path problem
MFE	minimum free energy
PCR	polymerase chain reaction

Chapter 1

Introduction

By means of DNA computations, DNA molecules are exploited as a carrier of information assigned by humans, as against their biological role as preserver of inheritance information. Such computation is a sequence of controlled chemical reactions of DNA manipulations, performed in the molecular biology laboratory. The molecules in use are mostly artificially synthesized DNA with specially composed sequences that encode the entities (variables) of an assignment to be solved.

The first idea of exploiting biomolecules for computational purposes was put forward by Feynman [31] in 1961. The first practical implementation of such a computation was achieved by Adleman [2] in 1994. He solved a seven-vertices instance of the Hamiltonian path problem with DNA. Since then, this task has become a benchmark assignment for testing the performance of new models for DNA computations in the wet laboratory. Thereupon followed many other explorations in this area; development of further DNA computation models was followed by their implementation in the wet laboratory.

Practical implementation of a computation scheme in a wet laboratory is a critical point for a model. First attempts to repeat the Adleman's experiment have failed [55]. Other DNA computing schemes have remained purely theoretical, e.g., the scheme of a DNA computer proposed by Roweis et al. [81].
The main difficulty of implementation is controlling the numerous conditions imposed by DNA chemistry. Modeling is based on the assumption, that only pairs of exactly complementary strands interact (*hybridize*) with each other. However, in a reaction tube inexact hybridizations between pairs of strands, which are only partially complementary, also take place. They hinder the intended flow of reactions and make the result of computations unreliable. Hence, the DNA computations require a preliminary step to find the appropriate *DNA encoding*, that is, the composition of DNA strands, such that under certain conditions of reaction the molecules behave in a manner prescribed by a model.
With the increasing size of the tasks, a manual design of strands sufficient for

early experiments has become complicated. Therefore, the means of traditional silicon-based computers are employed. Contemporary methods of computer-aided DNA strand design are based on combinatorial and/or thermodynamic criteria. The latter are more accurate, as they take into account the free energy of interacting strands by given reaction conditions, such as temperature and composition of the solution. To account for possible undesirable interactions, such methods require an efficient underlying procedure, which computes the thermodynamic stability of the DNA complexes built during the reaction.

In this thesis some novel computational methods are proposed that help to find reliable sets of DNA sequences to use in DNA computations. The task is addressed on two levels. The first one is a thermodynamic evaluation of a DNA encoding. Currently, there are several methods for DNA strand design [45, 30, 98] as well as ready sets of DNA words developed for DNA computing [15, 77, 24]. The performance of such sets *in vitro* differs for particular DNA computations that follow distinct models, since the intended reactions are not the same. The models using an encoding principle proposed by Adleman [2] imply interactions, that are not taken into account by modern strand design applications. Therefore, an evaluation method comprising additional restrictions is proposed, in order to more accurately assess the performance of a candidate DNA encoding for these models.

The second level is an underlying method for such evaluation, which computes thermodynamic stability of a complex built by hybridization of two arbitrary DNA strands. Because a structure of hybridization complex for such strands is initially unknown, this is an optimization problem, known as *minimum free energy (MFE) problem*. Contemporary solutions are based on the nearest neighbour model for assessing the thermal stability of the bimolecular DNA complex and utilize the paradigm of dynamic programming to find the structure of an energetically optimal one. Current methods implement different modifications of dynamic programming, such as greedy algorithm [54] and fractional programming approach [59]. This thesis introduces a novel graph model to comprise the potential structures of a hybridization complex, and computational methods for finding the MFE structure on such graph. These are also useful as a subroutine in applications for thermodynamic strand design.

The last issue addressed in this thesis concerns one of the basic problems of graph theory, *the shortest path problem*. An improved technique is introduced for the solution of this problem for a particular class of graphs, the bipartite graphs. It reduces memory and time requirements of a general solution methods, such as Floyd-Warshall [32, 101] algorithm.

The development of computational methods for biology belongs to the area of algorithmic bioinformatics, which focuses on the creation of novel algorithms and data structures. The first part of this thesis is a work in the field of structural

bioinformatics, which is concerned with the structure of proteins and nucleic acids. The second part is a contribution to algorithmic graph theory.

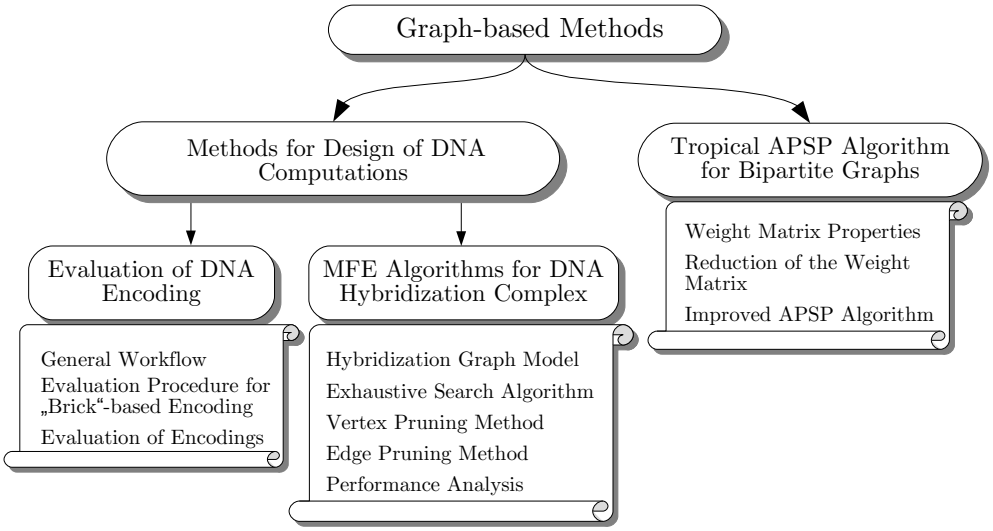


Figure 1.1: Research topics.

The major contributions of this thesis are the following (Figure 1.1):

1. A general workflow is developed for establishing of a procedure, that evaluates quality of a DNA encoding considering particular DNA computation scheme. Such a procedure assesses a temperature-dependent mutual behavior of the ssDNA strands under given reaction conditions. The effectiveness of the approach is demonstrated by establishing of an evaluation for DNA computation models, based on a principle proposed by Adleman [2]. The evaluations of several DNA word sets from the literature is performed.

2. A new approach is developed for solving a minimal free energy (MFE) problem for the hybridization complex built by a pair of partially complementary DNA molecules. It introduces a concept of the *hybridization graph* to represent whole magnitude of potential secondary structures for the bimolecular DNA complex. The MFE structure is represented by a path with minimal weight on this graph. A dynamic algorithm for direct solution, i.e., an exhaustive search, is proposed. This was originally presented in [95, 94].

3. Two advanced computational methods for finding the shortest path in a hybridization graph were developed. They are based on reduction of the graph by deletion of edges, which lead to suboptimal solutions. For both methods, the edge deletion criteria are developed so that they deliver the optimal path.

4. An advanced technique is presented for finding the shortest path for a par-

ticular class of bipartite graphs. It is based on the Floyd-Warshall algorithm and extends a tropical algebra form of the method.

The rest of the thesis is organized as follows:

Chapter 2 describes the basic topics of biology and informatics relevant to this work. It includes the structure of DNA, its hybridization complexes, and a thermodynamic model that allows assessment of their stability. Next, the formalisms of graph theory used in this thesis as main modelling framework are presented. The chapter describes the dynamic programming approach to developing algorithms, concluding with an outline of the area of DNA computing and a survey of work in related fields.

Chapters 3, 4 and 5 present the actual achievements of this thesis. Chapter 3 is concerned with the thermodynamic validation of a set of DNA codewords for a specific type of DNA computations.

Chapter 4 introduces a graph model for secondary structure of DNA/DNA hybridization complexes. Based on this, a straightforward solution method is proposed for the minimal free energy problem. Further on, two graph pruning strategies and implementational aspects of corresponding computational methods are presented. Their advantage in comparison to the straightforward method and the comparable existing ones is given.

Chapter 5 presents the special method for solving the all-pairs shortest path problem on bipartite graphs. Chapter 6 concludes the thesis and outlines directions for further research.

Chapter 2

Theoretical Basis

Structural bioinformatics is a sub-field of computer science and biology investigating the structure of proteins and nucleic acids. This chapter introduces relevant topics from both fields. In the first part, DNA as the central object of current research is presented. Then the related methods of informatics are introduced. Section 2.5 describes the field of DNA computation and the main directions of its evolution. It also presents two specific models of DNA computations addressed in current research. The last part defines the two problems addressed by this thesis: the encoding problem for DNA computations and the minimal free energy problem for two DNA strands. It also gives an overview of contemporary methods for solving these problems.

2.1 DNA

This section describes a biological basis for methods developed in the scope of this study. It concerns the structure of single-stranded DNA molecules and their complexes built by hybridization. Then a state-of-the-art model for quantitative estimation of stability of such complexes is presented.

A DNA single strand (ssDNA) is a linear biopolymer consisting of nucleobases (or simply bases) bound to a sugar-phosphate backbone. The alternation of phosphate and sugar gives an orientation to the backbone and to the whole DNA strand. The bases in the strand are usually taken from a phosphate (denoted as $5'$) to sugar (denoted as $3'$) terminus, if corresponding termini are not explicitely labeled. In DNA four types of nucleobases occur: adenine A, cytosine C, guanine G, and thymine T. The sequence of bases in a strand, e.g., $CGGATCG$, defines its *primary structure*. Adenine can pair with its *complementary base* or *Watson-Crick complement* thymine by hydrogen bonds. Similar bonding occurs between the other pair of complements cytosine and guanine. Binding of complementary bases of the single strand to one another is called

folding. Binding of two ssDNA is called *hybridization*. The resulting molecule is a *hybridization (or bimolecular) complex*. Bonded bases define a *secondary structure* for the folded ssDNA or the hybridization complex. Formally, a DNA single strand is represented as follows.
A DNA *alphabet*, i.e., a finite set of accepted characters, consists of four letters: $\Sigma = \{A, C, G, T\}$.
A DNA single strand is represented by an oriented string $a = a_1 \dots a_n$ defined by this alphabet; its left end corresponds to $5'$-terminus of the DNA strand.
The complementarity is a bidirectional relation on pairs $A \leftrightarrow T$ and $G \leftrightarrow C$. A base complementary to a_i is denoted $\overline{a_i}$.
If for two ssDNA $a(5'-3')$ and $a'(3'-5')$ of length n every base a_i is complementary to a'_i ($a'_i = \overline{a_i}$ for all $1 \leqslant i \leqslant n$), the strands are called complementary $a = \overline{a'}$.
Hybridization of two complementary strands is called *specific*. Two piecewise complementary strands hybridize *non-specifically*. A bimolecular complex built by specific hybridization is a Watson-Crick double helix.

Most DNA in living cells has a double helix structure and preserves genetic information, with the exception of some viruses with ssDNA genomes.

2.1.1 Structure of the Hybridization Complex

In the following section, we observe DNA bimolecular complexes built through hydrogen bonds between bases of different strands or, in other words, interstrand pairings. We assume that a *base pair* comprises two Watson-Crick complementary bases (A and T, or C and G) from different strands bound by hydrogen bonds.
Consider two DNA single strands $a(5'-3')$ and $a'(3'-5')$ of lengths n and m respectively, which are not complementary. In the hybridization complex built by these strands, two types of structural motifs can be observed: internal, that is, flanked by a base pair on two sides, and terminal ones, delimited by a base pair on one side. In Figure 2.1 is shown a complex exhibiting all types of DNA structural motifs.
The internal motifs are:

- *Stem* is a sequence of base pairs, where all constituting bases are consecutive on both strands. A stem is defined by two complementary substrings s and $s' = \overline{s}$ of a and a' respectively, where every base s_i is paired with its complementary one s'_i. Two consecutive base pairs $s_i s_{i+1}/s'_i s'_{i+1}$ are also called *stacking*; a stem is a sequence of stackings.
- *Loop* is a region of unpaired bases enclosed between two base pairs. It is defined by two substrings $l = l_1 \dots l_k$ and $l' = l'_1 \dots l'_{k'}$, where $2 \leqslant k \leqslant n$ and $2 \leqslant k' \leqslant m$, of the strands a and a' respectively. The left flanking bases l_1 and $l'_1 = \overline{l_1}$ build a base pair, the right flanking ones l_k and $l'_{k'} = \overline{l_k}$ too. The

$k-2$ bases between l_1 and l_k on the strand a and $k'-2$ ones between l'_1 and $l'_{k'}$ on the strand a' are unpaired.
Depending on the number of unpaired bases in each strand, three types of loops can be distinguished:

- *symmetric* has an equal number of unpaired bases; that is, $k-2 = k'-2 \geqslant 1$;
- *asymmetric* has unequal number of unpaired bases: $k-2 \neq k'-2$, with $k-2 \geqslant 1$ and $k'-2 \geqslant 1$;
- *bulge* is a loop, where only one of the strands exhibits unpaired bases; that is, either $k-2$ or $k'-2$ is zero.

The terminal motifs are:

– *Blunt end*, where the terminal bases of both strands a_1 and a'_1 (or a_m and a'_n) are paired with each other.

– *Dangling end*, where just one of the strands has an overhanging region of unpaired bases. For instance, a_1 is paired with some a'_j and $j > 1$. Due to strand orientation, the $5'$- and $3'$-dangling ends are distinguished, dependent on which terminus of a strand is overhanging.

– *Terminal mismatch*. Both strands have terminal regions of unpaired bases.

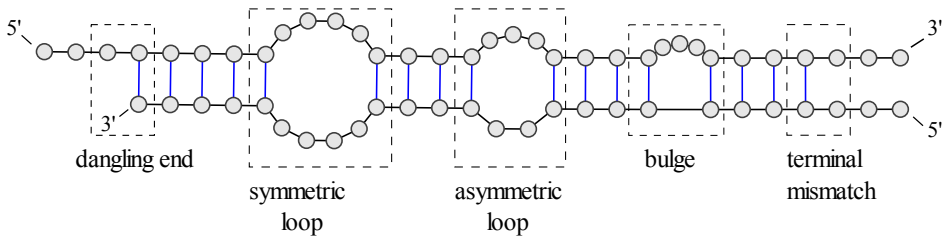


Figure 2.1: Structural motifs of DNA/DNA hybridization complex. Vertical lines denote paired Watson-Crick complementary bases (A-T and C-G).

Bimolecular complexes have two ends and, correspondingly, two terminal motifs. A quantitative measure of the complex stability is Gibbs free energy $\triangle G$ in $kcal/mol$. It is a thermodynamic potential, minimized when a system reaches equilibrium at constant pressure p and temperature T (in Kelvin):

$$\triangle G(p,T) = \triangle H - T \triangle S, \tag{2.1}$$

where $\triangle H$ is change of enthalpy showing heat flow of the reaction, that is whether it is absorbed (positive $\triangle H$) or emitted (negative). It is measured in $kcal/mol$. The change in entropy of the system $\triangle S$ shows a degree of disorder, which is measured in $cal/(molK)$. A DNA hybridization complex with arbitrary strands acquires secondary structure with minimal $\triangle G$.

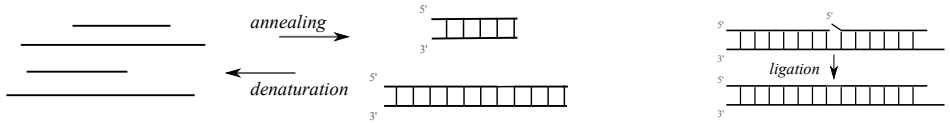
RNA

Ribonucleic acid is another representative of the class of nucleic acids. The chemical composition of RNA is very similar to that of single strands of DNA. The main differences are: the sugar group in backbone is ribose, while in DNA it is deoxyribose; the base uracil in RNA replaces thymine, so that in RNA adenine is complementary to uracil.
Due to these distinctions, RNA tends toward self-folding more than DNA. Correspondingly to this, the research on RNA is focused on predicting of its folding that comprises, beside the structural motifs possible by interstand interactions, hairpins and multiloops.

DNA manipulations

The main methods of DNA manipulation are:

- *Denaturation*, which is a separation of a dsDNA molecule to its constituent single strands without breaking their backbones. By increasing the temperature up to 85°C or 95°C the hydrogen bonds between Watson-Crick bases on different strands break (Figure 2.2a).
- *Annealing reaction*. It enables ssDNA strands to hybridize with one another, building dsDNA by intrastrand base pairings. It is achieved by slow cooling of the reaction tube (Figure 2.2a).
- *Ligation*, performed by a protein called ligase. It establishes a covalent bond between sugar and phosphate groups of two neighbouring nucleotides of the same strand, if their corresponding bases are paired with consecutive ones on the other strand (Figure 2.2b).



(a) Denaturation, Annealing (b) Ligation

Figure 2.2: DNA manipulations. (a) Reaction is reversible. (b) Ligase repairs a break in backbone.

- *Restriction*, performed by proteins called restriction enzymes. They bind to certain sequences of bases inside ssDNA or dsDNA and cleave the strands. That is, they break covalent bonds in the backbone of a strand between the sugar and phosphate groups of neighboring nucleotides.

2.1.2 Thermodynamics: Nearest Neighbour Model

The nearest neighbour model was first introduced for RNA molecules in the 1960s by Crothers and Zimm [21], and DeVoe and Tinoco [25]. This model is based on the observation that the stacking of bases along a helix is reinforced by hydrophobic interactions, van der Waals and other forces. The aggregate energy of these interactions exceeds that of the hydrogen bonding between the paired bases. From this, it was deduced that the stability of the DNA double helixes can be predicted from the primary sequences of the two participating strands, where only the interactions between adjacent base pairs are taken into account [16].

In the 1980s, several laboratories proved that the model is suitable for DNA and obtained corresponding experimental data for thermodynamic stability of stackings [16, 41, 100]. That allowed the prediction of the stability for duplexes built by two complementary DNA single strands.
The model defines nearest neighbours as the stacking of two consecutive base pairs $(a_i a_{i+1}/a'_j a'_{j+1})$ from strands $a(5'-3')$ and $a'(3'-5')$, respectively. That is, Watson-Crick nearest neighbours, which consist of two adjacent pairs of complementary bases such as *(AC/TG)* (in the order $5'-3'/3'-5'$). There are ten types of such nearest neighbours [17], since from 16 possible permutations six are symmetric, e.g., *(CG/GC)* or *(GT/CA)*.
Further research extended the data set with stabilities of loops, containing two unpaired bases of all possible configurations [3, 4, 5, 6]. This allowed the prediction of the stability for hybridization complexes containing such loops, in addition to plain duplexes.
The actual data set for the nearest neighbour model was developed in 2004 in SantaLucia laboratory [83]. It includes the experimental data for Watson-Crick nearest neighbours, internal and terminal mismatches (nearest neighbours with either (a_i/a'_j) or (a_{i+1}/a'_{j+1}) unpaired, Figure 2.3), and dangling ends. The parameters were gathered and extrapolated to predict the stability for DNA structural motifs with regions of unpaired bases longer than two, such as symmetric and asymmetric loops, and also for bulges [84]. This data set allows the calculation of the Gibbs free energy, enthalpy and entropy of DNA/DNA hybridization complexes.

In this section, the Equations (2.3) – (2.8) for the free energy of DNA structural motifs are given according to SantaLucia and Hicks [84].

Let E denote the composite Gibbs free energy of a DNA/DNA complex or motif, and let $\triangle G$ denote the experimental data from the nearest neighbour data set. Since a DNA/DNA complex can be considered as a linear combination of

stems, loops, and terminal motifs, the DNA/DNA pairing energy is given by

$$E_{\text{total}} = E_{\text{add}} + \sum_{i=1}^{k} E_{\text{stem}}(S_i) + \sum_{j=1}^{m} E_{\text{loop}}(L_j) + E_{\text{term}}(left) + E_{\text{term}}(right), \tag{2.2}$$

where the hybridization complex has stems $S_1, \ldots, S_k$ and loops $L_1, \ldots, L_m$. The energy term E_{add} includes effects from symmetry of the sequences and helix initiation energy, which are independent from secondary structure of the complex:

$$E_{\text{add}} = \triangle G_{\text{init}} + \triangle G_{\text{sym}}. \tag{2.3}$$

The energy term $E_{\text{term}}(left/right)$ accounts for terminal motifs such as blunt and dangling ends, terminal mismatches and closing *(A/T)* pairs at corresponding ends of the complex:

$$E_{\text{term}}(left/right) = \triangle G_{\text{AT}}(left/right) + \begin{cases} 0, & \text{blunt end,} \\ \triangle G_{5'-\text{dEnd}}(a_i, a'_j), \text{ (either } i = m \text{ or } j = 1), & 5'\text{-dangling end,} \\ \triangle G_{3'-\text{dEnd}}(a_i, a'_j), \text{ (either } i = 1 \text{ or } j = n), & 3'\text{-dangling end,} \\ \triangle G_{\text{termMM}}(a_i, a'_j), & \text{terminal mismatch.} \end{cases}$$

The terminal penalty $\triangle G_{\text{AT}}$ is applied if a terminal motif is closed by the base pair *(A/T)* or *(T/A)*. The term $\triangle G_{\text{termMM}}(a_i, a'_j)$ is a free energy contribution of a terminal unpaired region. If each strand has more than one unpaired base, this contribution is that of a single mismatch. For the left end it is a left mismatch $(a_{i-1}a_i/a'_{j-1}a'_j)$ (left pair is unbound). For the right end it is a right mismatch $(a_i a_{i+1}/a'_j a'_{j+1})$. In both mismatches only a_i with a'_j build a base pair.

The energy terms E_{stem} and E_{loop} are additive with respect to nearest neighbour pairs. The energy contribution of a stem $S = s_1 \ldots s_n / s'_1 \ldots s'_n$ is given by following equation:

$$E_{\text{stem}}(S) = \sum_{i=1}^{n-1} \triangle G_{\text{NN}}(s_i s_{i+1}/s'_i s'_{i+1}), \tag{2.4}$$

where $\triangle G_{\text{NN}}(s_i s_{i+1}/s'_i s'_{i+1})$ denotes the energy of the stacking $(s_i s_{i+1}/s'_i s'_{i+1})$ (Figure 2.3).

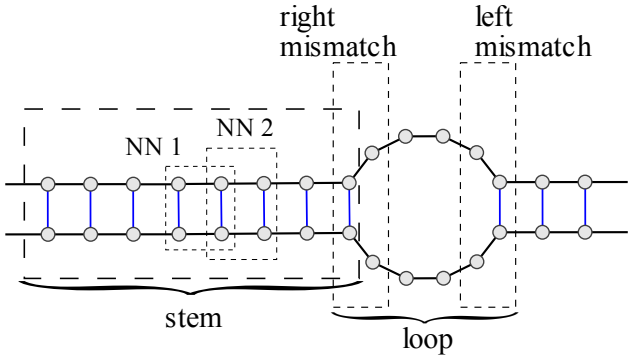


Figure 2.3: Calculation of Gibbs free energy for loops and stems. Here *NN 1* and *NN 2* are Watson-Crick nearest neighbours (stackings). The stem consists of eight base pairs, building seven stackings. The loop is symmetric with eight unpaired bases.

The energy contribution of a loop depends on the loop type:

– A *symmetric loop* L has an equal number of unpaired bases in both sequences and its energy contribution amounts to

$$E_{\text{symLoop}}(L) = \triangle G_{\text{rightMM}} + \triangle G_{\text{loop}}(l) + \triangle G_{\text{leftMM}}, \tag{2.5}$$

where $\triangle G_{\text{rightMM}}$ and $\triangle G_{\text{leftMM}}$ is the Gibbs free energy of neighboring pairs at the beginning and at the end of the loop, respectively (Figure 2.3). Such nearest neighbours contain one noncomplementary pair of bases (mismatch). Moreover, $\triangle G_{\text{loop}}(l)$ is the Gibbs free energy contribution of l unpaired bases in the loop (Figure 2.3). The exact values for loops up to ten bases were found experimentally. For longer loops a Jacobson-Stockmayer extrapolation is used [51]:

$$\triangle G_{\text{loop}}(l) = \triangle G_{\text{loop}}(x) + 2.44 \times R \times 310.15 \times \ln(l/x), \tag{2.6}$$

where $\triangle G_{\text{loop}}(x)$ is a free energy increment of the longest loop of length x with experimental data, and R is the gas constant. The coefficient 2.44 is based on kinetic measurements in DNA [39].

– An *asymmetric loop* L has an unequal number of unpaired bases in the sequences and its energy contribution is

$$E_{\text{asymLoop}}(L) = \triangle G_{\text{rightMM}} + \triangle G_{\text{loop}}(l) + E_{\text{asym}} + \triangle G_{\text{leftMM}}, \tag{2.7}$$

where $E_{\text{asym}} = |l_1 - l_2| \times 0.3\,\text{kcal/mol}$ depends on the difference between the length l_1 and l_2 of unpaired regions in the strands.

– A *bulge loop* L has unpaired bases in only one of the strands and its energy contribution is:

$$E_{\text{bulge}}(L) = \triangle G_{\text{bulge}}(l) + \triangle G(\text{intNN}) + \triangle \text{G}_{\text{ATterm}}, \tag{2.8}$$

where l is the number of unpaired bases in the bulge. The value $\triangle G(\text{intNN})$ comprises the contribution of the intervening nearest neighbours for a bulge of length one base:

$$\triangle G(\text{intNN}) = \begin{cases} 0, & \text{if } l > 1, \\ \triangle G_{\text{NN}}(a_i a_{i+1}/a'_i a'_{i+2}), & \text{if } l = 1 \text{ and } a'_{i+1} \text{ is unpaired}, \\ \triangle G_{\text{NN}}(a_i a_{i+2}/a'_i a'_{i+1}), & \text{if } l = 1 \text{ and } a_{i+1} \text{ is unpaired}. \end{cases}$$

The energy of Watson-Crick nearest neighbours is negative and thus stackings always have a stabilizing effect. On the other hand, the energy of the three types of loops is predominantly positive, except for symmetric two-base loops of certain configurations. Thus, loops generally have a destabilizing effect on the overall structure.

The enthalpy $\triangle H$ for the complex is calculated similarly. The database contains the corresponding parameters for all the structural motifs. The only distinction is that the enthalpy increment for unpaired regions in loops is zero; that is, $\triangle H_{\text{loop}}(l) = \triangle H_{\text{bulge}}(l) = H_{\text{asym}} = 0$.

The calculation of entropy $\triangle S$ of the hybridization complex abides by the same scheme as the Gibbs free energy $\triangle G$, or it can be obtained as follows:

$$\triangle S = \frac{\triangle G - \triangle H}{T},$$

where T is the temperature in degrees Kelvin.

Stability of DNA motifs under different experimental environment

The parameters in the thermodynamic database can be adapted to various experimental conditions. The published thermodynamic database was established under 37°C and NaCl concentration of 1 mol/l. For a different temperature T, the corresponding set of Gibbs free energy contributions is calculated using Equation (2.1): $\triangle G_T = \triangle H - T \triangle S$.
It is assumed that $\triangle H$ and $\triangle S$ are temperature independent [84].
There are also formulae for the change of salt concentration [84]:

$$\triangle G_{37}[\text{Na}^+] = \triangle \text{G}_{37}[1\text{mol/lNaCl}] - 0.114 \times \text{N}/2 \times \ln[\text{Na}^+],$$

$$\triangle S[\text{Na}^+] = \triangle \text{S}[1\text{mol/lNaCl}] + 0.368 \times \text{N}/2 \times \ln[\text{Na}^+],$$

where N is the total number of phosphates in the hybridization complex, and $[\text{Na}^+]$ is the total concentration of monovalent positively charged particles.

An important parameter for controlling the hybridization reaction is the DNA melting temperature. It is the temperature, by which 50% of the molecules are

in single strand state. It is calculated from the changes of enthalpy and entropy as follows:

$$T_M = \frac{\triangle H \times 100}{\triangle S + R \times \ln(C_T/x)} - 273.15, \tag{2.9}$$

where C_T is the total molar strand concentration; R is the gas constant, and x accounts for the self-complementarity of the duplex.

Terminal mismatch parameters

Several laboratories have measured Gibbs free energy, enthalpy and entropy for Watson-Crick nearest neighbours. Comparative analysis provided in SantaLucia [82] shows compatibility of their results. The differences in measured values are issued by distinct reaction conditions such as salt concentration of solution and strand concentration. Further on, the parameters for single mismatches and destabilizing energies for loops of different length were developed in SantaLucia laboratory [83]. Such terminal effects as terminal mismatches are still a matter for research. In particular, there are currently two ways to calculate the energy of terminal mismatches:

- as the sum of dangling ends energies;
- using the energies of internal mismatches.

According to Peyret [78], both these methods give ambiguous results and more basic research is required to obtain better precision. Hence, by implementation of the algorithms described in Chapter 4 the second method is employed to account for the terminal effects in DNA complexes.

2.1.3 Minimum Free Energy Problem

The nearest neighbour thermodynamic model provides the equations and a data base of parameters to calculate the stability of DNA structural motifs. However, for a hybridization complex built by two arbitrary strands, the secondary structure and, correspondingly, the motifs present are initially unknown. According to molecular thermodynamics, the complex tends to acquire the conformation with lowest free energy. So, such conformation should be found among all potential structures. The corresponding optimization problem is usually stated as follows:

Minimum Free Energy Problem: Given two nucleic acid sequences and a thermodynamic model, find the secondary structure with smallest free energy change under this model, to which the sequences can hybridize.

This implies the search over the space of all admissible structures (combinations of base pairs) for two given strands. According to interstrand model, the structure is admissible if in strands a and a' there are no intrastrand pairings and for all base pairs (a_i/a'_j) and (a_{i_1}/a'_{j_1}) holds: if $i < i_1$, then $j < j_1$.
According to the nearest neighbour thermodynamic model, the objective is to find the minimal value of E_{total} given by Equation (2.2). Since the first term E_{add} of this expression is constant for a pair of given strands, the objective value for MFE problem has the following form:

$$E_{\min} = \min \left\{ \sum_{i=1}^{k} E_{\text{stem}}(S_i) + \sum_{j=1}^{m} E_{\text{loop}}(L_j) + E_{\text{term}} \right\}, \tag{2.10}$$

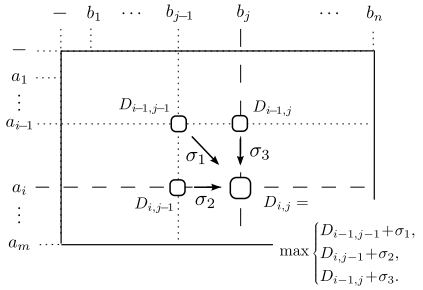
where the minimum ranges over all admissible structures of bimolecular complex for two given DNA strands; S_i and L_j are potential stems and loops, respectively; the term E_{term} accounts for energy contribution of the both left and right termini of the complex.

2.2 Dynamic Programming

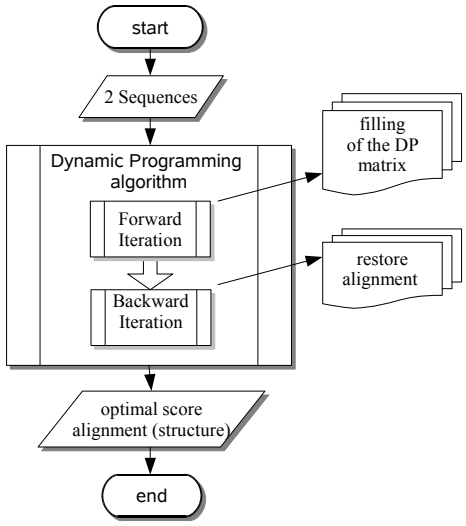
This paradigm was introduced by Bellman [11] in 1957. It is a method for designing algorithms to solve optimization problems. The main principle is to break down the initial problem into smaller subproblems in a recursive manner. Then, starting with the solution to the smallest subproblems, the solutions for enclosing subproblems are found. The approach is widely used in bioinformatics to detect homology between molecules, find genes, and predict the secondary structure for nucleic acids.
These tasks are typically solved by finding an optimal *alignment* between sequences, that represent the primary structure of molecules, i.e., proteins or nucleic acids. An alignment is an arrangement of the sequences one over another by insertion of spaces, so, that they have the same length. To assess the quality of alignment a score is assigned to each column, dependent on whether it contains a space, two different (*mismatch*) or equal (*identity*) letters. This assignment is called a *scoring scheme/function*.

The computation course of the algorithms following this paradigm is based on establishing of a *dynamic programming (DP) matrix* (Figure 2.4a). Its rows and columns are labeled with letters of the corresponding sequences. The cells represent columns of alignment and retain optimal scores found for subproblems, which are alignments of substrings from first positions to the current ones in the strings. So the whole matrix contains the optimal scores for each subproblem.
In most cases, an optimal alignment in addition to the score is also of interest. To obtain it, the transitions leading to the optimal score in every cell are usually

retained while filling the matrix. By tracing back the stored transitions from the cell with the optimal score, the optimal alignment is recovered.
Hence, the dynamic programming algorithms are usually divided into two parts: forward and backward (Figure 2.4b).



(a) dynamic programming matrix
$D_{i,j}$ – iterative function
σ – scoring function



(b) general flow chart for dynamic programming algorithm

Figure 2.4: Implementation of the dynamic programming approach.
(a) The equation is given according to Needleman-Wunsch algorithm. The symbol "–" denotes a space (extension to the accepted alphabet for alignment algorithms).
(b) Two parts of DP algorithm.

The forward part consists of the following three steps:

1. Initialization of the DP matrix, that is, assigning the scores to the smallest subproblems, which are represented by cells of the first row and column. Their scores are usually constant.

2. Filling of the matrix, that is, finding the optimal score for each cell and saving it for further computation. The scores of smaller subproblems are used to find the cost of enclosing ones according to recursive formulae and the given scoring scheme.

3. Finding the optimal score. In simple cases, such as global alignment, the score of last cell in a matrix is the optimal one. In more complex ones, the optimal score is found amongst a number of cells.

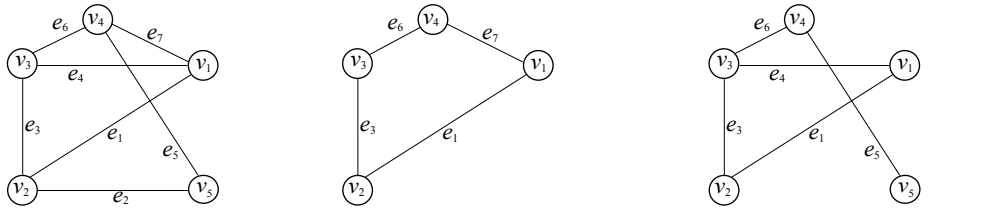
The design of algorithms following the paradigm consists of specification of each step according to a problem. The first and third steps are usually simple computations. The second step is the most elaborate one. A particular implementation of each step depends strongly on the scoring scheme.

Basic dynamic programming algorithms for pairwise sequence alignment that use simple scoring functions are the ones from Needleman and Wunsch [74], and Smith and Waterman [88], the method from Gotoh [40] employs more complex affine scoring.
The dynamic programming approach is also used to find the secondary structure of nucleic acids. The main scheme of computations is similar to that of alignment algorithms.

2.3 Graph Theory

2.3.1 Basic Definitions

A graph $G = (V, E)$ is an ordered pair of non-empty set V and a set E of two-element subsets of V. The elements of V are called *vertices* and that of E are called *edges*. The edges $e = \{v_i, v_j\}$, where $v_i \neq v_j$, of a graph are denoted by a pair of vertices as $v_i v_j$. A graph is commonly described by a diagram, in which the vertices are points and edges are lines joining them pairwise (Figure 2.5a). The cardinality of V is the *order* of the graph, and the cardinality of E is the *size* of the graph.
Two vertices are *adjacent* if they define an edge; it is also said that they are *incident* with this edge. The number of incident edges (or adjacent vertices) for every vertex v_i is its *degree* $d(v_i)$. All adjacent vertices of a vertex v_i build its *neighbourhood* $N_G(v_i)$.



(a) a graph G (b) a subgraph of G (c) a spanning subgraph of G

Figure 2.5: A graph with its subgraph and spanning subgraph.
(a) A graph $G = (V, E)$ with the vertex set $V = \{v_1, v_2, \ldots, v_5\}$ and the edge set $E = \{e_1, e_2, \ldots, e_7\} = \{v_1v_2, v_2v_5, \ldots, v_4v_1\}$. The order of the graph is $|V| = 5$, its size is $|E| = 7$.
(b) A subgraph G' with $V' = \{v_1, v_2, v_3, v_4\}$ and $E' = \{e_1, e_3, e_6, e_7\}$ of the graph G.
(c) Spanning subgraph contains all vertices of the graph G. The edges e_5 and e_7 are deleted from the supergraph.

One of the forms to represent a graph is an *adjacency matrix* $A = (a_{i,j})$. It is a two-dimensional Boolean matrix, in which the rows and columns represent source and destination vertices respectively, and its entries indicate whether an

edge exists between the vertices associated with that row and column. For simple graphs the matrix is symmetric; its main diagonal contains zeros:

$$a_{i,j} = \begin{cases} 1, & \text{if } v_i v_j \in E, \\ 0, & \text{otherwise.} \end{cases}$$

A *subgraph* $G' = (V', E')$ of a graph $G = (V, E)$ consists of sets $V' \subseteq V$ and $E' \subseteq E$ (Figure 2.5b). The graph G is then a supergraph for G'. If $V' = V$, then G' is called *spanning* subgraph of G. So, the spanning subgraph for a given graph is obtained by possibly deleting of some edges.

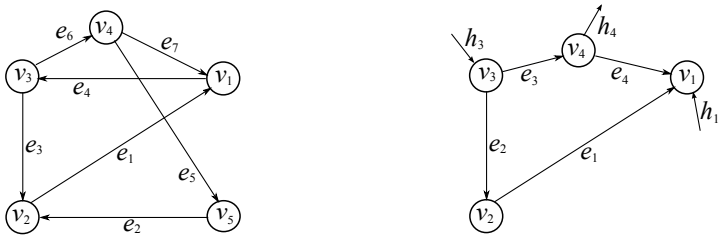
In graph theory, some types (or classes) of graphs, which exhibit special properties are defined. For the thesis the two following graph classes are important. The first class is *complete* graphs, where each pair of vertices is adjacent, e.g., a four-vertex complete graph in Figure 2.6a. The second class is *bipartite* graphs. For them, the set V can be divided in two parts, such that no vertex is adjacent to another one in the same part. Figure 2.6b demonstrates a bipartite graph with five vertices, where one part contains three and another part two vertices.

(a) complete graph (b) bipartite graph

Figure 2.6: Special graphs. (a) Complete graph with four vertices. (b) The three vertices on the left build one part of the graph, the two ones on the right – the second part.

A sequence of adjacent vertices $P(v_1, v_2, \ldots, v_k)$, where $v_i v_{i+1} \in E$ for every i and $1 \leqslant i < k$, is a *path* through the graph. The first vertex in a path v_1 is the *start vertex*, and the last one is the *end vertex*; they both are *terminal vertices*. A path with coinsident starting and ending vertices is called *cycle*.

A graph, where all pairs (v_i, v_j) in E are ordered, is a *directed* graph or *digraph*. Directed edges are denoted in the diagram by arrows (Figure 2.7a). The first vertex in a pair is called the *initial* or *tail* (of an arrow); the second one is the *terminal* or *head*. The edge $v_i v_j$ leaves the vertex v_i and enters v_j. For the vertices of a digraph, in- and out-neighbourhoods are distinguished. The *in-neighbourhood* $N_G^-(v_i)$ of a vertex consists of all vertices incident to it by edges entering v_i. These vertices are also called *direct predecessors* of v_i. The vertices in the *out-neighbourhood* $N_G^+(v_i)$ are joined by edges leaving a vertex v_i. They are *direct successors* to the vertex v_i. For directed graphs, the adjacency matrix is asymmetric.



(a) a digraph (b) a digraph with half-edges

Figure 2.7: Digraphs. (a) Digraph for the graph in Figure 2.5a. (b) Digraph with three half-edges. Vertices v_1 and v_3 are incident to entering half-edges, vertex v_4 has a leaving one.

If edges of a graph are associated with real numerical values, that is, there is a mapping $E \to \mathbb{R}$ for the edges, the graph is called *weighted*. For such a graph, a *path weight* $W(P)$ is the sum of weights of the edges constituting that path.

For some applications, the classical graph theory is extended with additional structures.
Half-edges are edges defined by a single vertex. They reflect some additional properties (e.g., colouring) for single vertices or their groups. Direction and weight can be assigned to half-edges. Therefore, in some directed graphs a vertex can be incident to an entering or leaving half-edge (Figure 2.7b).

2.3.2 Path Problems in Graph Theory

Shortest path problem

The *shortest path* between two nodes in a graph is the one with the lowest weight. For an unweighted graph, the weight of all edges is assumed to be one. So the shortest path in such a graph is that with the lowest number of edges. Depending on particular practical tasks, there are four general forms of the problem:

1. If both starting and ending vertices of a path are given, it is a single-pair shortest path problem. The popular heuristic method for this problem is an *A* search* algorithm.

2. If only the starting vertex is given, it is a single-source shortest path problem. The shortest paths from the given vertex to all other vertices in the graph must be found.

3. If only the ending vertex is given, it is a single-destination shortest path problem. The shortest paths from all vertices in the graph to the given one must be found. This can be reduced to the single-source shortest path problem by reversing the edges in the graph. This problem and the two

previous ones are solved by Dijkstra's [26] algorithm.

4. If the terminal vertices are unknown, then it is an *all-pairs* shortest path (APSP) problem. The shortest paths between every pair of vertices in the graph must be found. This problem can be solved by Floyd-Warshall [32, 101] or Johnson's [52] algorithm. The latter one is faster on sparse graphs.

The last form of the problem is the most general one.

For the purposes of this thesis, another form of the problem is relevant:

$X-Y$ ***shortest path problem:*** If a set $X \subseteq V$ of start vertices and a set $Y \subseteq V$ of end vertices are given, then the objective is to find from all paths with a start in X and an end in Y the single shortest one.

Hamiltonian path problem

A path through a graph is Hamiltonian if it contains all vertices of the graph exactly once. Detecting such a path for a given graph or its absence is a Hamiltonian path problem. It is defined on both undirected graphs and digraphs; for example, in Figure 2.7a the paths $P(v_2, v_1, v_3, v_4, v_5)$ and $P(v_4, v_5, v_2, v_1, v_3)$ are Hamiltonian.
The problem is important for many practical tasks, such as root planning, clustering problems [96], and optimization of manufacturing processes [8].

2.4 APSP Algorithms

2.4.1 Floyd-Warshall Algorithm

The algorithm was introduced by Floyd [32] and extended by Warshall [101] in 1962 to solve the all-pairs shortest path problem for a weighted graph. Another important application of the algorithm is the finding of a transitive closure for a graph (or binary relation), that is, a set of points reachable from every other point.

The weighted graphs are described by the *weight matrix* $M = (m_{ij})$, which is analogous to the adjacency matrix, but contains as entries the weights w_{ij} for the corresponding edges:

$$m_{ij} = \begin{cases} 0, & \text{if } i = j, \\ \infty, & \text{if } i \neq j \text{ and } v_i v_j \notin E, \\ w_{ij}, & \text{otherwise.} \end{cases}$$

The algorithm follows the paradigm of dynamic programming and consists of several steps, each of which updates the whole dynamic programming (DP) matrix. The DP matrix D is initialized by the weight matrix of the graph. At every

step k, the cell $D_{i,j}$ retains the lowest weight of a path from the vertex v_i to v_j, which uses intermediate vertices from v_1 to v_k. After $|V|$ steps, the matrix contains the weights of the shortest paths between all pairs of vertices. Procedure 1 shows the pseudocode of the algorithm.

Procedure 1 Floyd-Warshall Algorithm

```
Given: Graph G = (V, E) with the weight matrix M
D=M
for k = 1 to |V| do
  for i = 1 to |V| do
    for j = 1 to |V| do
      /* Update the weight of the shortest path between vertices v_i and v_j */
      D_ij = min(D_ij, D_ik + D_kj)
    end for
  end for
end for
return D
```

A restriction on the algorithm is that a graph should not contain any cycles with negative weight. If such cycle exists in the graph, one of the cells on the main diagonal of the resulting matrix D has negative value, since these cells represent the paths with the coinsident terminal vertices. The time complexity of the algorithm is $O(|V|^3)$.

2.4.2 APSP Algorithm in Tropical Algebra

Tropical algebra

The field of tropical algebra, also known as min-plus algebra, was pioneered by Simon [87] in the late 1980s. In the last decade, it has gained closer attention with the progress of such areas of applied mathematics as control theory, optimization and statistics. Tropical representation was found to be extremely useful for optimization problems. A number of DP algorithms for optimization problems, e.g., Floyd-Warshall and Dijkstra's, have much simpler forms in this representation.

In tropical algebra, the basic arithmetic operations of addition and multiplication on extended real numbers $\mathbb{R} \cup \{\infty\}$ are redefined as follows:

$$x \oplus y := \min(x, y) \quad \text{and} \quad x \odot y := x + y.$$

Thus, the tropical sum of two numbers is their minimum, and the tropical product is their sum. The neutral element for addition is infinity, for multiplication – zero:

$$x \oplus \infty = x \quad \text{and} \quad x \odot 0 = x.$$

The tropical scalar product of a row vector with a column vector is the scalar

$$\begin{aligned}(r_1, r_2, \ldots, r_n) \odot (c_1, c_2, \ldots, c_n)^T &= r_1 \odot c_1 \oplus r_2 \odot c_2 \oplus \cdots \oplus r_n \odot c_n \\ &= \min(r_1 + c_1, r_2 + c_2, \ldots, r_n + c_n).\end{aligned}$$

Tropical APSP algorithm

Consider the weight matrix M for a graph with n vertices without negative cycles. The tropical multiplication of its row i with column j gives the following value:

$$\begin{aligned}&(m_{i1}, m_{i2}, \ldots, m_{in}) \odot (m_{1j}, m_{2j}, \ldots, m_{nj})^T \\ &= m_{i1} \odot m_{1j} \oplus m_{i2} \odot m_{2j} \oplus \cdots \oplus m_{in} \odot m_{nj} \\ &= \min(m_{i1} + m_{1j}, m_{i2} + m_{2j}, \ldots, m_{in} + m_{nj}).\end{aligned}$$

Each sum in the last expression represents the weight of a path from vertex i to j, which contains maximal two edges. The tropical product of these elements of the weight matrix is the length of the shortest path containing two edges at most. Thus, the second tropical power $M^{\odot 2}$ of a weight matrix contains such weights for every pair of vertices. By induction on the power, every further power $M^{\odot k}$ holds the weights of the shortest paths containing k edges at most. Hence, the matrix $M^{\odot n-1}$ represents a solution to the all-pairs shortest path problem.
This means the solution to APSP problem is found in tropical algebra by multiplication of the weight matrix with itself $n-1$ times:

$$D = \underbrace{M \odot M \odot \cdots \odot M}_{n-1 \text{ times}} = M^{\odot n-1}.$$

Similarly to the matrix of the Floyd-Warshall algorithm, the tropical solution matrix $M^{\odot n-1}$ contains negative entries on the main diagonal, if the graph has negative cycles.

2.5 DNA Computing

This section gives a general description of the field of DNA computations and the main directions of research. Then it demonstrates two coding principles often used in DNA computation models. The section is concluded with detailed description of seminal Adleman's experiment, which became a benchmark for research in this area and constitutes a basis of the one presented in Chapter 3.

The field of DNA computing emerged in 1994 after the first experiment of Adleman [2] and its generalization by Lipton [61]. Adleman [2] solved a seven-vertices instance of the Hamiltonian path problem (HPP) with DNA molecules

in a wet laboratory. Lipton [61] proved the feasibility of the approach for the solution of the satisfyability (SAT) problem and generalized it for contact networks. Since then, the HPP and SAT problem have become benchmark tasks in the field of DNA computing, that are used to prove the consistency of other models of DNA computation. Many other works in the area have followed. Detailed classification is provided, for instance, by Hinze [48].
The main problems approached by DNA computations are:

- computationally hard mathematical problems, such as Hamiltonian path, satisfyability, vertex coloring [64, 62, 106];
- implementation of computational models from computational theory, such as finite state automata [12] and Turing machine;
- implementation of binary logic circuits [43, 44, 102, 80, 109];
- logical control of gene expression [13, 63, 107].

Ignatova et al. [50] describe three historical waves of DNA computing experiments:

- Manual. To obtain the answer, a sequence of operations performed by humans is required. These computations were mainly concerned with the solution of computationally hard mathematical problems.
- Autonomous. The manipulations on DNA strands *in vitro* are fulfilled by enzymes and do not require a human operator.
- Cellular. These computations are also autonomous. The main strain is to develop DNA automatons, which function in living cells and control gene expression products, e.g., substituting mutated protein with a correct one. This direction comprises a wider area than just DNA computation, since it involves further interactions with RNA.

The evolution of the field of DNA computing can be observed from the two characterizations given above. At first, DNA-based computations were considered as solvers of the hard computational problems due to their high parallelism compared to silicon-based computers. Later, that perspective became unclear, since the uncertainty of biochemical reactions makes the implementations hardly scalable [60]. Thus, the focus changed to optimization of the DNA computation process. That has been successfully achieved by employing additional catalytic agents (e.g., restriction enzymes) [12], that digest DNA molecules without intervention from the operator. The restriction depends on short subsequences present in strands. The intended manipulations can be encoded into DNA sequences. Cellular models are now in the development stage: there are some *in vitro* implementations of such automata [13, 47, 46]. An *in vivo* automaton was achieved for *E. coli* bacteria [73]. The ultimate objective is to use them in medicine to repair dangerous mutations that lead to diseases, such as cancer or

diabetes, caused by genetic defects. An example of such implementation is a model of computational genes proposed by Martínez-Pérez et al. [70].

Thus, the evolution of DNA computations has discarded the initial incentive of outperforming the silicon-based ones. Modern research set perspectives on applications in medicine and the life sciences for a better understanding of mechanisms of life and practical exploitations of DNA automata for diagnosis and cure.

The general scheme of a DNA computing experiment is given in Figure 2.8. Each step involves several techniques of molecular biology, based on biochemical reactions involving DNA, such as hybridization, ligation, or restriction.

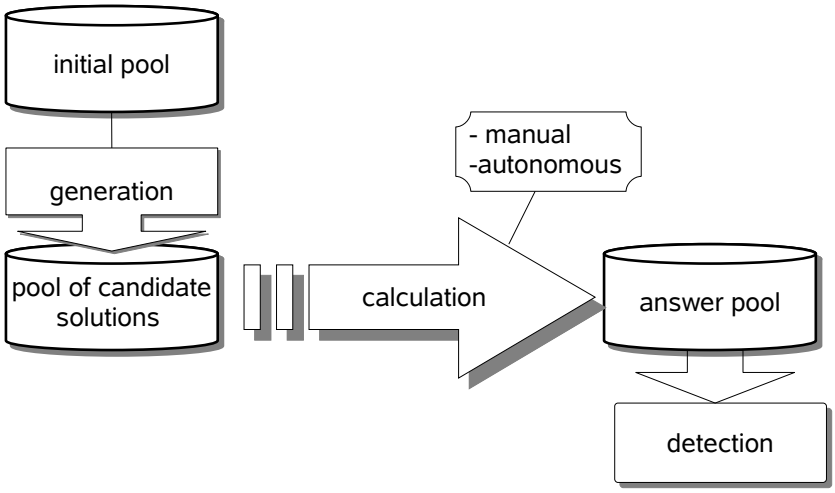


Figure 2.8: General flow chart for DNA computations.

The *initial pool* contains ssDNA or dsDNA molecules that represent the entities of a problem assignment under computation,– for instance, variables for mathematical problems or states and transition rules for automata. This pool is used mainly by *self-assembly models*, that exploit hybridization and following ligation (*hybridization/ligation*) to build a candidate solution molecules from separate units. A candidate solution pool can also be manually designed and generated separately; then the initial pool is not involved as an independent unit.

The calculation step starts with a pool of candidate solutions and results in an *answer pool*. This step is a sequence of DNA manipulations, that yield molecules representing the correct answer. The manipulations are performed manually or in an autonomous manner.

The last detection step is done manually, e.g., by gel electrophoresis.

A *model of DNA computation* defines all steps of the experiment. This requires: a data representation scheme or *coding principle*, an algorithm for the calculation step, and detection method. For manual models the algorithm is a

sequence of operations. For autonomous ones it is an intended mechanism of reaction, i.e., the interactions of DNA blocks with catalytic agents, that lead to DNA molecules representing the answer.
To perform a DNA computation in a wet laboratory, particular substances (DNA sequences and particular catalytic agents) and reaction conditions are selected, including a mapping of problem entities into DNA sequences. The coding principle reflects both the entities of a problem and their intended interactions under a given computation model.

Coding principles in self-assembly models

In the scope of this work the following two coding principles are relevant. They use single stranded DNA molecules as interacting units. These principles are mainly used in linear self-assembly models, where the initial pool is built by hybridization/ligation.

1. A *sticker* principle was initially introduced by Roweis et al. [81] and found its use in further computation experiments [15] and extended models [68]. It defines two types of strands: *memory strands*, which encode a sequence of bits; and *stickers* – the short strands complementary to every bit (Figure 2.9). Intended interactions are binding of a sticker to a memory strand on a region corresponding to its bit. The *memory complexes* are built in this manner.

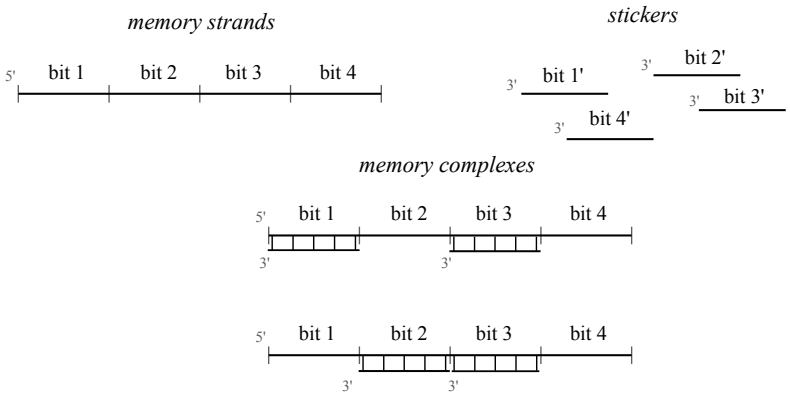


Figure 2.9: Sticker-based scheme.

2. A *brick-based* principle was introduced by Adleman [2] for his manual model and reapplied in the next generation of computations, i.e., autonomous ones [69, 110]. It is employed to solve the computational problems, that involve two types of entities, e.g., vertices and edges. Such entities are encoded interdependently. Every DNA sequence encoding an edge

consists of two parts. The first one is the complement to the second half of the sequence representing its tail. The second one is the complement to the first half of its head (Figure 2.10). Intended interactions are the binding of one strand from a set to a pair of the strands from another set.

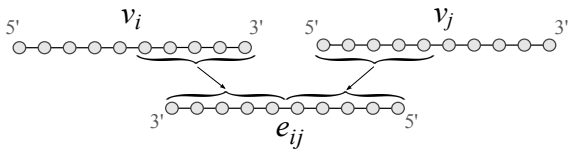


Figure 2.10: Brick-based coding. The words v_i and v_j represent vertices. The word e_{ij} encodes the edge, joining these vertices.

Adleman's solution of the Hamiltonian path problem

In the Figure 2.11, a seven-vertices instance of a Hamiltonian path problem solved by Adleman [2] is shown. Additional conditions in the assignment is the fixed start v_0 and end v_6 vertices for the path. The computational model exploits the brick-based coding principle for data representation.

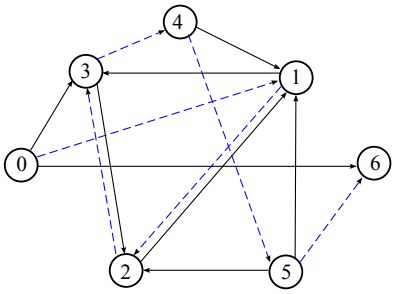


Figure 2.11: Graph considered by Adleman [2]. It comprises seven vertices and 14 edges. Dashed path is Hamiltonian.

The initial pool contains ssDNA of length 20 nt for five vertices $v_1, \ldots, v_5$ and ten edges of the given graph. The strands encoding the edges starting at v_0 or ending at v_6 are 30 nt long; there are four such edges in the given graph. The pool of candidate solutions is build from the initial one through hybridization/ligation. This pool contains dsDNA molecules that represent all paths in the given graph. These strands are at first separated by starting and ending vertex via PCR: only the strands starting with v_0 and ending with v_6 are amplified. From the resulting pool, the strands with the length of 140 nt are extracted that represent all seven-vertex paths of the graph. The computation step comprises denaturation of dsDNA to single strands consisting of vertex- and edge-strands, and five consecutive rounds of manual extractions. At the

first round, the edge-strands containing a sequence complementary to v_1 are extracted from the common pool, then, the extracted strands are objected to a second round with v_2 as detection sequence. After the last round with v_5 as detection sequence, the tube contains all molecules representing paths through all the vertices, that is, Hamiltonian paths. If the resulting tube is empty, the graph does not contain such a path.

2.6 Related Work

The success of DNA computation in wet laboratory strongly depends on correct representation of information units (e.g., variables) through the encoding by DNA molecules. This section outlines the problems of encoding evaluation and free energy calculation for DNA computing addressed in this thesis in the context of modern research.

2.6.1 Evaluation of a DNA Word Set

The importance of proper encoding was recognized in early stages of development of DNA computing [2, 9]. The random encoding, used in early small-size computations such as Adleman's first experiment [2], has been shown to be inappropriate to solve the assignments of larger size [22]. Hence, the finding of reliable sets of DNA strands (*words*) is recognized as separate problem, which is solved by the methods of DNA strand design.

A DNA word set is considered reliable when the strand interactions proceed as defined by the DNA computation model and produce detectable amount of answer molecules under conditions of experiment [38, 37]. The predominant models require specific hybridization of DNA strands with their complementary counterparts in order to build correct solution molecules; the non-specific hybridizations may lead to false positives or decrease the effectiveness of computation by waste of strands in byproducts of the intermediate reactions. Thus, the objective of the majority of the strand design methods is to compose a DNA word set, where specific hybridization (between each word and its complement) is significantly more stable than the non-specific interactions (of a word with other words and their complements) in a set. This properties can be ensured by *in silico* strand design by setting certain criteria and evaluating DNA words according to them.

Evaluation criteria

Currently, two types of criteria are applied by strand design methods. The *combinatorial* criteria comprise the pairwise distances or similarities, analogous to those in information theory, for instance, Hamming or Levenshtein distance

between the words; and GC-content of the strands (so that specific hybridizations have similar stability). In 1997, Garzon et al. [34] introduced the H-measure, which is the lowest Hamming distance for a pair of DNA words considering all possible mutual shifts. This gives more exact estimation for likelihood of hybridization than Hamming distance. The combinatorial constraints allow to achieve the desired difference or similarity in base composition of the strands.

The criteria of another type, the *thermodynamic* ones, give an estimation of hybridization behaviour of DNA strands.

The very basic criterion is a melting temperature (T_M) of the specific hybridizations. It is often used in combination with combinatorial constraints to ensure the uniform melting point by *in vitro* reactions [22, 66].

The most reliable criteria developed so far are based on the Gibbs free energy of specific and non-specific hybridizations. Since they reflect hybridization propensity between all molecules in a DNA pool. It has been shown that such constraints are preferable over Hamming distance [91] and H-measure constraints [92] for separation of the specific hybridizations from the non-specific ones. Tanaka et al. [91] and Deaton et al. [24] implemented the free energy criteria as predefined energy bounds for non-specific hybridizations. Penchovsky and Ackermann [77] introduced *free energy gap* criterion that comprises the difference in free energy between the weakest specific hybridization and the strongest non-specific hybridization for every word within the set. In order to reduce the amount of non-specific hybridizations it should be as large as possible.

The values of the evaluation criteria attained after the design process characterize the quality of a DNA word set.

Evaluation process

By DNA strand design three kinds of evaluations are to consider:

1. Internal evaluation of fitting of a candidate word into a set.
2. Evaluation of quality of a word set.
3. Evaluation of performance of a word set for certain type of computations.

The *internal evaluation* is a built-in procedure for strand design methods that computes the pairwise relations according to selected design criteria for every new candidate word concerning the rest of the set. Considering the exponential space of the words of the length n and the number of possible sets, such computations are highly intense. In this regard, some strand design methods that avoid the complex internal evaluation are worth to describe.

Deaton et al. [23] proposed an *in vitro* approach that allows to obtain a pool of non-interacting strands via PCR selection. This technique delivers high quality sets that can be directly used for a wet laboratory experiment [18]. However, the

base composition of the molecules is unknown, and the sequencing of all the produced molecules is expensive. These factors constitute a significant disadvantage for application of these method for DNA computations.

Other methods are graph-based approaches using rational design rather than traditional constraint-driven selection of sequences. These methods discard the explicit evaluation of a word against the whole set calculating only properties depending on a current candidate, such as GC-content and melting temperature.

Feldkamp et al. [30] describe a method to generate a set of sequences such that each subsequence longer than a given threshhold $t < n$ would be unique throughout the set. For this, a digraph with vertices labeled by DNA strings of length $t+1$ is established. The edges are drawn between vertices, where the last t symbols of the first subsequence coinside with the first t symbols of the second one. The DNA words in a set are then found as superimpositions of the strings at the vertices along the vertex-disjoint paths in the graph. The resulting words are checked for meeting the *GC*-content and melting temperature constraints. The method is implemented in DNASequenceGenerator [30].

Pancoska et al. [75] proposed an approach based on the generation of the permutated sequences for a given one, such that the strands have similar melting temperature. The authors map a given strand to an unweighted four-vertex Eulerian digraph (with the same in- and out-degree for each vertex) with vertices corresponding to the four nucleotides and edges representing the connection of consecutive bases in a given strand. This graph contains multiple edges between two vertices, since two bases can be neighboured in a strand several times. Such a graph with its adjacency matrix defines a family of sequences containing the same nearest neighbour pairs that leads to the similar free energy and melting temperature for specific hybridizations. The single words are found by constructing Eulerian paths (containing all edges exactly ones) in the graph.

Kurniawan et al. [57] represent all possible DNA sequences as complete four-vertex digraph, whose edges are weighted by free energy of the Watson-Crick nearest neighbours. The words are then found as paths of length n; for this, an artificial intelligence approach (intelligent agents) is employed. After this, the paths are filtered out according to their weight (converted to melting temperature) and *GC*-content.

The *evaluation of quality* of a DNA word set is performed *in vitro* or *in silico* to prove the effectiveness of certain design criteria. For instance, the advances of the free energy criteria over Hamming distance and H-measure was shown by Tanaka et al. [92] through *in vitro* investigation of hybridization specificity of the sequences designed under each of these three criteria. Deaton et al. [24] showed by gel electrophoresis the absence of hybridization between the sequences of a word set developed by method proposed in their work.

By *in silico* evaluation the characteristics of a predesigned word set are calculated. Recent methods employ mostly free energy boundaries and free energy

gap as quality measures for comparative evaluation [98, 36, 108]. The size of designed sets ranges from several words (e.g., given in [30, 85]) to several thousand ones (e.g., described in [36, 108]) and is much smaller than the whole space of strands (4^n) considered by internal evaluation. Thus, the evaluation of quality is computationally less intense than the internal one.

Certain models of DNA computation impose additional or even distinct requirements on participating strands, such as interdependence of strands for brick-based or tile assembly models [104], or longer strands for special vertices (e.g., terminal vertices by Adleman's [2] computation). Morover, the results of *in vitro* computations depend on conditions of chemical environment, such as salinity, temperature, presence of enzymes. Such additional conditions are complicated to cover by strand design alone. For such cases, the reliability of encoding can be assessed through *evaluation of performance*. It can be executed *in vitro* by setting appropriate test cases and proving the correctness of result without executing the actual computation. For instance, Penchovsky and Ackermann [77] developed a word set for sticker based models and employed four configurations of 12-bit memory strands to show the absence of unintended interactions.

The other way is *in silico* simulation of a particular computation performed with the help of special software applications, the simulators. They are designed to imitate by conventional computers the chemical reactions in a test tube, thus giving an option of pilot experiments without waste of wet laboratory resources and saving the corresponding time and money. There are simulators handling only certain types of DNA computations, such as Xgrow Simulator [105] and ISU TAS [76] for tile assembly models [104], or a simulator described by Pfannkuche [79] and Kummerfeldt [56] for sticker-based models and automata proposed in Benenson et al. [12].
The more general simulator is EdnaCo [35], which models the tube environment (solution conditions and volume) and random local interactions for single or double stranded DNA typical for biomolecular reactions. It runs on a cluster of 24 PCs. It was reported to successfully simulate such reactions as PCR selection protocol of Deaton et al. [23], Adleman's experiment, tile assembly computation and some others [36].

2.6.2 Computation of Thermodynamic Criteria for DNA Strand Design

By DNA strand design two kinds of thermodynamic criteria are used:

- Melting temperature is a measure of the stability of specific hybridizations;
- Gibbs free energy is more general one comprising stability of both specific and non-specific interactions.

Finding the melting temperature is a straightforward consecutive addition of corresponding parameters (enthalpy and entropy) for each pair of Watson-Crick nearest neighbours present in a given DNA strand and applying the Equation (2.9). The respective software applications, such as DAN and MeltTemp, were developed after publication of the corresponding thermodynamic parameters.

The finding of the Gibbs free energy for hybridization of arbitrary strands requires more intricate algorithms, since the secondary structure of a bimolecular complex is initially unknown. In this case, the minimal free energy (MFE) problem (Section 2.1.3) should be solved for a given pair of strands.

For rigorous MFE computations, the correspondingly adapted RNA-folding methods are applied, such as RNAcofold provided by Vienna RNA package [42], PairFold from RNAsoft [7] suite, or mfold from server DINAMelt [67]. These methods extend the dynamic programming algorithm of Zuker and Stiegler [112], which finds the optimal folding for single RNA molecule, to the case of bimolecular hybridization.
In RNAcofold, the two given strands are concatenated into one, and several abstract bases that can not pair with any other are placed between them. Then, the optimal folding of resulting strand is found by Zuker and Stiegler algorithm.
In PairFold the strands are connected without additional bases and the dynamic algorithm is extended with special equations for structural motifs if they contain a junction point.
PairFold was used in thermodynamically based strand design methods of Tulpan et al. [98], and Zhang et al. [108]; Penchovsky and Ackermann [77], and Ackermann and Gast [1] used RNAcofold.

The time complexity of the Zuker and Stiegler's folding algorithm is $O(n^4)$; restriction of the maximal loop length to 30 nt yields a reduction to $O(n^3)$ [49]. In PairFold the later option is implemented, so for a pair of strands with lengths n and m its complexity is $O((n+m)^3)$.

Due to the cubic time complexity of rigorous MFE algorithms, their employment considerably slows down the large-scale internal evaluation by DNA strand design [91, 98]. To overcome this problem, approximative calculations for free energy are often applied, for instance, by Deaton et al. [24], Kaderali [53], and Garzon et al. [36].
Kaderali and Schliep [54] and Deaton et al. [24] developed approximative methods that extend the basic alignment algorithms. Kaderali and Schliep [54] used the Needleman-Wunsch [74] algorithm applying melting temperature as objective function. This approach misses the optimum in definite cases [53, 59]. Their method was improved by Leber et al. [59] through employment of fractional programming approach. Deaton et al. [24] used the Smith-Waterman [88] algorithm for local alignment with free energy as objective function.

Both methods provided sufficient precision for separation of specific from non-specific hybridizations according to *in vitro* evaluations shown in the respective works. The complexity of the respective algorithms is that of the background alignment algorithms, namely $O(n^2)$.
Tanaka et al. [91] employed a two-stage thermodynamic evaluation: first filtering proves the estimated energy value of candidate words and the second one an exact energy. At the first stage, the method assess the free energy by finding the most stable helix between given strands and then the paired bases in its flanking regions. Only the candidate words, which met the threshhold of the first filter, are subjected to the exact MFE calculation. The complexity of the first-stage energy calculations is $O(n^2)$, of the second-stage ones – $O(n^3)$.

These approximative methods are based on the interstrand hybridization model [91, 53]. Tulpan et al. [98] showed that for short DNA molecules used in DNA computations, the results of rigorous methods and that using this assumption correlate well: for strands of the length 25 nt the Pearson's coefficient is $r = 0.83$; for 50-mers $r = 0.58$.

There are also simplified versions of rigorous MFE algorithms that perform calculations under the interstrand model. In the Vienna RNA package the corresponding method is called RNAduplex, in server DINAMelt [67] – HYBRID.

2.6.3 All Pairs Shortest Path Algorithms

The all-pairs shortest path (APSP) problem is one of the fundamental algorithmic graph problems. The basic methods for solving this problem are Floyd-Warshall [32, 101] and Johnson's [52] algorithms. The computational complexity of the first one is $O(|V|^3)$, of the second $O(|V|^2 \log |V| + |V||E|)$. The main strain of the recent methods in this field is to achieve the sub-cubic complexity. A number of proposed solution methods are based on special data structures and/or properties of computer architecture, such as *table lookup* [33] or *bit-level parallelism* [27]. Other methods follow the heuristic approach, for instance, by solving the all-pairs almost shortest path problem, which is a relaxation of the initial one and allows certain one-sided deviation from the optimal weight [28]. There are a number of approaches that exploit the option of parallel computations, like the methods proposed by Takaoka [89] or Chen [19]. The detailed survey of the methods developed so far can be found in Dragan [29] or Zwick [113].

With the progress of such application fields as networking systems or logistics, the practical importance of specific methods for restricted classes of graphs has increased. Such methods exploit the regularities of the graph geometry exhibited by certain types of graphs, e.g., chordal, permutation, or distance-hereditary graphs. The restriction on the edge weights, such as unit cost (unweighted graph) or bounded positive values, are considered prospective.

The solutions for specific case of bipartite graphs were proposed by Chen [19], and Chin-Wen and Chang [20]. The first work presented sequential and parallel algorithms for unweighted bipartite permutation graphs. The method exploits the symmetry of the adjacency matrix and strong ordering of the vertices characteristical for such graphs. The second one proposed the solution for unweighted chordal bipartite graphs.

Chapter 3

Evaluation of DNA Encoding

DNA word sets designed by corresponding methods satisfy the general requirement, that is, higher stability of the specific hybridizations compared to the non-specific ones (Section 2.6.1). However, different models of DNA computation impose additional conditions that may cause unexpected interactions of the strands. Such effects can be assessed *in silico* through simulation of the DNA computations with appropriate software, such as Xgrow Simulator [105] or EdnaCo [35].

In this chapter, a lightweight software module is proposed that evaluates the performance of a candidate encoding under conditions of the particular DNA computation. Such module computes thermodynamic characteristics of the strands omitting the computationally intense full-scale simulation. By the state-of-the-art, the most reliable measure for the evaluation is Gibbs free energy of hybridization between DNA molecules.
Further, a general workflow for the development of such evaluation modules is formulated. Following this workflow, an evaluation procedure is developed for DNA computions that use Adleman's [2] encoding principle, i.e., the *brick principle*. The method is demonstrated on several DNA word sets from literature for the instance of the Hamiltonian path problem (HPP) first solved by Adleman [2] (Section 2.5).
The presented workflow was used implicitly in other work (Kummerfeldt [56]) to employ thermodynamic evaluation for simulation of other models of DNA computations.

This chapter uses a terminology common for the area of DNA strand design. The DNA words are sequences over DNA alphabet that encode entities of a mathematical problems, e.g., variables. All DNA words in the same word set have the same length. DNA strands involved in computations can comprise several words. The DNA encoding for a particular computation is a set of sequences, corresponding to the variables of a given instance of a problem. The encoding is build from the elements of a DNA word set. Moreover, the encoding

can contain sequences of different length according to the requirements of specific DNA computation model.

3.1 A Workflow for Development of Evaluation Procedure

The evaluation is based on exact calculation of the Gibbs free energy and takes into account the specificity of a particular computation experiment. The main steps by establishing of such procedure are the following:

1. description of a DNA pool;
2. definition of *intended* and *unintended* interactions between DNA molecules under a given computation model;
3. definition of the thermodynamic quality measure for this model to assess the performance quantitatively;
4. implementation of the procedure for calculation of the measure defined in previous step for an encoding from input.

The evaluation procedure requires as input a candidate DNA-encoding and a formalized description for a given problem instance. For example, for Hamiltonian path problem, a problem instance is a particular graph, in which that path should be found. Its formalized description may contain a list of edges given by their starting and ending vertices.

3.2 Evaluation Method for Brick-based Computations

To demonstrate the effectiveness of the proposed evaluation approach, the brick-based computations were selected for the following reasons. Firstly, in the calculation phase, Adleman [2] used polymerase chain reaction to amplify the molecules, representing the paths with intended terminal vertices, and a reaction similar to the sticker-based one was applied to extract the answer strands from the pool of candidate solutions (Section 2.5). Thus, employment of one of the published DNA word sets ensures the correctness of this phase. Secondly, in the hybridization/ligation phase, the reaction differs from the sticker-based scheme; and the words may interact in undesirable manner. The evaluation of a candidate DNA-encoding from predesigned sets helps to answer this question and to avoid the generation of the encoding from scratch.

The next sections follow the presented workflow for the development of the evaluation procedure. In Section 3.3, the results of the evaluation for several

DNA word sets from the literature are demonstrated.

Analysis of DNA pool and reaction scheme

In this section, the first two steps of the workflow are demonstrated.

The brick-based computations comprise two phases, which employ distinct reaction schemes: hybridization/ligation and calculation phases.

Consider the hybridization/ligation phase (Figure 3.1). It requires an initial DNA pool with two interdependent subsets such that each word in one subset relates to a pair of words in another one. The brick-based computations are typically applied to the problems represented by graphs. Thus, the words in the subsets are further denoted as $v_i(5'-3')$ and $e_{ij}(3'-5')$, for the ones encoding vertices and edges respectively. A word e_{ij} that encodes the edge connecting two vertices i and j, is composed from the sequences encoding these vertices as follows: $e_{ij} = \overline{v}_{i,l/2+1} \dots \overline{v}_{i,l}\, \overline{v}_{j,1} \dots \overline{v}_{j,l/2}$, where l is the word length and $v_{i,k}$ is the k-th base in sequence v_i.

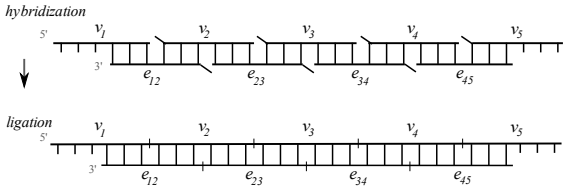
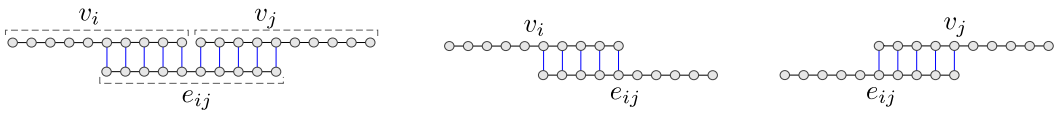


Figure 3.1: Hybridization/ligation reaction.

The brick-based coding principle implies as intended interaction the joining of one word from the one subset (e.g., edge) with two corresponding words from the other subset (e.g., vertices) (Figure 3.2a).



(a) trimolecular complex (b) bimolecular complex (c) bimolecular complex

Figure 3.2: Intended hybridization complexes for brick-based computations in the hybridization/ligation phase.
(a) Edge e_{ij} joining two adjacent vertices.
(b) Bimolecular complex between vertex v_i and edge e_{ij}.
(c) Bimolecular complex between vertex v_j and edge e_{ij}.

The pairwise hybridizations involve the corresponding halves of a vertex- and an edge-word:

1. the second (3′-) part of the word v_i is paired with the first (3′-) one of the word e_{ij} (Figure 3.2b);

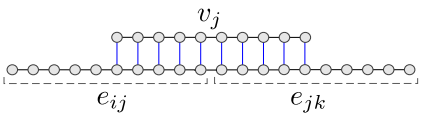
2. the first (5′-) part of the word v_j is paired with the second (5′-) one of the word e_{ij} (Figure 3.2c).

As unintended are considered complexes differing from intended ones. For the brick-based coding scheme, the unintended interactions occur between the pairs of words of the same subset or between words representing non-incident vertex and edge. The secondary structure of such complexes contains mismatches (unpaired bases) and/or features shifted frame relative to the intended complex (Figure 3.3).

(a) mismatch (b) frame shift

Figure 3.3: Unintended hybridizations in hybridization/ligation phase of brick-based computations.
(a) Complex contains mismatch, where $k \neq j$.
(b) Complex with shifted frame, where k may equal j.

Consider the calculation phase. Its course is similar to the sticker-based reaction (Figure 2.9, Section 2.5). It involves the vertex-words as stickers and the resulting strands from the hybridization/ligation phase as memory strands. The latter may contain both vertex-words (in $3' - 5'$ direction) and edge-words in the same strand, since in hybridization/ligation phase the molecules are freely floating and indistinguishable in a tube. The interactions between the vertex-words and the concatenation of the two words representing its incident edges are intended (Figure 3.4a). The examples of unintended structures are given in Figures 3.4b and 3.4c.



(a) intended complex

(b) mismatch (c) frame shift

Figure 3.4: Intended and unintended hybridizations in the calculation phase.
(a) Hybridization of v_j at junction site of corresponding edge-words.
(a) Hybridization complex contains two mismatches (two-base loops).
(c) Hybridization complex demonstrates frame shift.

The intended and unintended hybridizations for brick-based computations are summarized in Table 3.1; the hybridizations typically taken into account by DNA strand design are given for reference.
The DNA pool considered by strand design applications consists of the words $w_i(5'-3')$ and their corresponding complements $c_i(3'-5')$, i.e., $c_i = \overline{w}_i$. The hybridization complex of two strands is denoted as w_i/c_j. The expression $(w_i w_j)$ denotes concatenation of two words into one strand. The expression $rev(w_i)$ denotes the reversed word to ensure the correct direction of DNA molecules by hybridization.

Table 3.1: Intended and unintended hybridizations.

Method	*hybridizations*	
	intended	unintended
common strand design methods extended [98, 86]	w_i/c_i	w_i/c_j, where $i \neq j$, $w_i/rev(w_j)$, $rev(c_j)/c_i$ $(w_i w_j)/c_k$*
brick-based		
hybridization phase	v_i/e_{ij}, v_j/e_{ij}	v_i/e_{jk}, for $i \neq j$ and $i \neq k$ $v_i/rev(v_j)$, for all i, j $rev(e_{cd})/e_{ij}$, for all i, j, c, d
calculation phase	$v_k/(e_{ik}e_{kj})$,	$v_k/(e_{ij}e_{cd})$, where $k \neq j$ or $k \neq c$, $v_k/rev(v_i v_j)$, $v_k/(rev(v_i)e_{cd})$, $v_k/(e_{cd}rev(v_i))$.

* - additional interactions taken into account by some strand design applications.

As can be seen from Table 3.1, the predominant methods of DNA strand design consider pairwise interactions between words. The specific hybridization between a word and its complement is the intended one; the non-specific hybridizations are unintended. The extended methods regard additionally unintended hybridizations for sticker-based reaction: between two concatenated words and a complement.

The brick-based models consider a different DNA pool, which contains the interdependent sets of vertex- and edge-words; the specific hybridizations between any two words are unintended.

These differences between the two reaction schemes lead to the conclusion that the values of the quantitative characteristics attained by design for a DNA word set, will differ under conditions of brick-based computations.

Evaluation criteria and quality measures

The third step in the workflow is the establishment of the set of conditions that allow to assess the performance of DNA encoding for brick-based computation *in silico*.

Consider that in a DNA pool for brick-based computations the subset of vertex-words unambiguously defines the corresponding edge-words and vice versa. Assume that the vertices are encoded by n distinct DNA-words of length l bases and the edge-words are built correspondingly. The following combinatorial conditions prove the absence of specific hybridizations between the words in the pool:

- $\overline{v}_i \neq rev(v_j)$ – excludes specific hybridizations between vertex-words;
- $\overline{e}_{ij} \neq rev(e_{cd})$ – excludes specific hybridizations between edge-words;
- $\overline{v}_i \neq e_{cd}$ – excludes specific hybridizations between vertex- and edge-words;
- $e_{ij} \neq e_{cd}$, where $c \neq i$ and $d \neq j$, – ensures the uniqueness of edge-words;

where i, j, c, and d are indices for vertices, so their range is from 1 to n; the expression $rev()$ means the reversed word to ensure the correct direction of DNA molecules by hybridization. If an encoding set violates any of this conditions, it is obviously inappropriate for brick-based computations.

The free energy gap

By thermodynamically based DNA strand design, the free energy gap between intended and unintended hybridizations is widely employed as a quality measure of a DNA word set. It is calculated as the smallest difference between the MFE of unintended hybridization complexes and the MFE of intended ones for a word:

$$\delta = \min \left\{ E_{\text{unint}}(w_i) - E_{\text{int}}(w_i) \right\}, \tag{3.1}$$

where the minimum is taken over all words w_i in the set, E_{unint} is the lowest MFE of unintended complexes, and E_{int} is an MFE value of the intended complex for a word w_i.
It should be noted that in literature the above formula is given in rather different form and use distinct notation for MFE ($\triangle G$), but the essence is the same.
The exact definition of the terms E_{unint} and E_{int} depends on design method. Usually $E_{\text{int}} = E(w_i/c_i)$ for each word, where $E(w_i/c_i)$ is an MFE of hybridization of the two given strands; the unintended hybridizations can be defined differently (Table 3.1).

Consider the brick-based computations. Regarding Table 3.1 the free energy gap is calculated separately for each of the two phases. Since they are performed consecutively and follow two different reaction schemes.

In the hybridization/ligation phase, the calculation of the energy for intended complexes differs for vertex- and edge-words:

$$E^b_{\text{int}}(e_{ij}) = \max\big(\, E(v_i/e_{ij}),\, E(v_j/e_{ij})\,\big) \text{ for all } v_i,\, v_j \in V,$$

and

$$E^b_{\text{int}}(v_i) = \max\big\{\, E(v_i/e_{ij}),\, E(v_i/e_{ji})\,\big\} \text{ for all } e_{ij},\, e_{ji} \in E,$$

where the superscript b denotes that parameters are calculated for the brick-based models.
The sets of unintended hybridizations are also distinct for the two types of words:

$$E^b_{\text{unint}}(e_{ij}) = \min\big\{\, E(v_k/e_{ij}),\, E(rev(e_{cd})/e_{ij})\,\big\} \quad \text{for all } v_k \in V \text{ with } k \neq i,\, k \neq j, \text{ and } e_{cd} \in E,$$

and

$$E^b_{\text{unint}}(v_i) = \min\big\{\, E(v_i/e_{cd}),\, E(v_i/rev(v_j)\,\big\} \quad \text{for all } e_{cd} \in E \text{ with } c \neq i,\, d \neq i, \text{ and } v_j \in V.$$

Then the energy gap for the hybridization/ligation phase of brick-based computations is calculated as follows:

$$\delta^b = \min\big\{\, E^b_{\text{unint}}(v_i) - E^b_{\text{int}}(v_i),\, E^b_{\text{unint}}(e_{ij}) - E^b_{\text{int}}(e_{ij})\,\big\}, \tag{3.2}$$

where the minimum ranges over all vertex- and edge-words in the encoding.

In the computation phase, the participating strands are the vertex-words as stickers and the strands consisting of vertex and edge-words concatenated in arbitrary order as memory strands. The corresponding free energy gap is calculated between stickers and pairwise concatenated words and is usually denoted as τ for strand design methods. Considering the intended and unintended interactions for this phase from Table 3.1, the respective free energy gap for brick-based computations τ^b is found as follows:

$$\tau^b = \min\big\{\, E^b_{\text{unint}}(v_k) - E^b_{\text{int}}(v_k)\big\}, \tag{3.3}$$

where the minimum ranges over all vertex-words; and

$$E^b_{\text{int}}(v_k) = \max\big\{E(v_k/e_{ik}e_{kj})\big\}, \qquad \text{for all } e_{ij}, e_{jk} \in E,$$

and

$$E^b_{\text{unint}}(v_j) = \min\begin{cases} \min\{E(v_k/rev(v_iv_j)\}, & \text{for all } v_i, v_j \in V, \\ \min\{E(v_k/(e_{ij}e_{cd})\}, & \text{for all } e_{ij}, e_{cd} \in E \text{ with } j \neq k \text{ and } c \neq k, \\ \min\{E(v_k/(rev(v_i)e_{cd})\}, & \text{for all } v_i \in V \text{ and } e_{cd} \in E, \\ \min\{E(v_k/(e_{cd}rev(v_i))\}, & \text{for all } v_i \in V \text{ and } e_{cd} \in E. \end{cases}$$

Evaluation procedure

The evaluation procedure computes the MFE for all intended and unintended hybridization complexes and the corresponding energy gap. To perform the evaluation, the graph representation of an assignment and candidate encoding for the vertices are required. The encoding for the edges is generated correspondingly. The procedure consists of the following steps:

1. After reading the n initial sequences as input, generation of the vertex- and edge-words according to the scheme proposed in Adleman [2].
2. Check of the combinatorial conditions for the given encoding. If this fails, the encoding is classified as inappropriate for brick-based computation and the procedure terminates.
3. Combination of DNA words into pairs, representing intended and unintended complexes.
4. Calculation of the MFE for each pair. It can be performed with the help of external software, such as PairFold [7] or HYBRID from server DINAMelt [67].
5. Calculation of the free energy gaps δ^b and τ^b according to Equations (3.2) and (3.3).

The worst case performance of an encoding can be assessed by computing the energy gap for a complete directed n-vertex graph with $n(n-1)$ edges. To obtain the performance for a particular assignment, given by a formal description, only the existing edges are considered for the calculation of δ^b and τ^b.

The presented evaluation method was implemented in `C++`. For energy calculations, the method described in Chapter 4 was employed. Reaction conditions such as temperature and salinity (NaCl) are adjustable. For a given DNA word set, the procedure delivers the free energy gaps for the worst case (complete graph) and for a particular assignment.

3.3 Evaluation of DNA Encoding

The developed evaluation procedure was applied to the benchmark task in the field of DNA computing, that is, Hamiltonian path assignment solved by Adleman's [2] first experiment. The corresponding graph is given in Figure 3.5. It consists of seven vertices and 14 edges and contains a single Hamiltonian path.

The evaluation settings correspond to the conditions of Adleman's original computation [2], namely: the vertex-words v_0 and v_6 do not participate in the hybridization/ligation phase; words for edges starting at v_0 or ending at v_6 are 30 nt long: $e_{0i} = \overline{v}_{0,1} \ldots \overline{v}_{0,l}\, \overline{v}_{j,1} \ldots \overline{v}_{j,l/2}$ and $e_{i6} = \overline{v}_{i,l/2+1} \ldots \overline{v}_{i,l}\, \overline{v}_{6,1} \ldots \overline{v}_{6,l}$. For the encodings with the word length $l \neq 20$, the length of these edges is $l+l/2$.

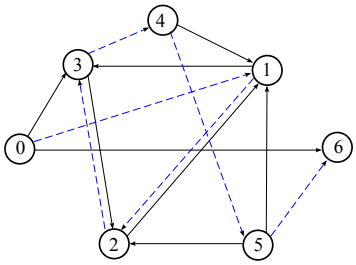


Figure 3.5: Graph considered by Adleman [2]. The path built by dashed edges is Hamiltonian.

The spot check evaluation was performed for a number of sets of DNA words from the literature and the randomly generated ones. The random sets were generated with RandomDNASequenceGenerator [99]. From each set containing more than seven words the ten seven-word subsets were randomly selected for evaluation.

Table 3.2 shows the results of this evaluation. The evaluated word sets are grouped into three blocks according to their word lengths of 12, 16 and 20 nt; the fourth block (sets 15 – 17) shows the performance of the benchmark encoding used in Adleman [2] and specially designed negative control sets. The words in the negative control set 16 were designed to produce overlapped unintended complexes similar to that in Figure 3.4c. The words in the negative control set 17 exhibit one mismatch without frame shifts in unintended complexes in the calculation phase (Figure 3.4b).

The first four columns of the table give general information about the evaluated DNA word sets, such as sources, word length, and set size. The fifth column shows the free energy gap δ, if it is given in a source; "+" indicates that other thermodynamic constraints were used for the design of the word set (e.g., MFE threshold); "–" means that only combinatorial and the melting temperature constraints were applied. The last four columns present the results of the performed evaluation. The columns six and seven show the performance of the encodings in the hybridization/ligation phase; the columns eight and nine – in the calculation phase. In the sixth column the value ranges of δ^b for the complete graph with 42 edges are given. The seventh column shows the largest value of the free energy gap δ^b for the assignment graph with 14 edges; the corresponding lowest value is mainly the same as for the complete graph. The last two columns present the values of τ^b for the complete and assignment graphs, respectively.

A negative value of δ^b indicates the building of unintended hybridization complexes that are more stable than the intended ones for certain words in the encoding. This means that higher amount of molecules would be wasted in building of unintended complexes in the hybridization/ligation phase. Hence, the negative δ^b is a sign of the low efficiency of the DNA computation.

Table 3.2: Evaluation results for DNA encodings.
Reaction conditions: temperature 25°C, strand concentration 0.2 mmol/l, salt concentration 1 mol/l NaCl.

	DNA word set/ source	word length (nt)	set size	δ (kcal/mol)	δ^b(kcal/mol)		τ^b(kcal/mol)	
					complete graph 42 edges	Adleman's graph 14 edges	complete graph 42 edges	Adleman's graph 14 edges
1	random 12	12	21	–	-9.19 – -1.35	1.06	-0.73 – 4.6	5.91
2	Shortreed et al. [86]	12	64	2.84	-4.35 – -2.52	-2.1	1.95 – 2.41	4.01
3	random 16	16	21	–	-4.74 – 0.61	1.94	3.17 – 7.62	9.29
4	Shortreed et al. [86]	16	64	6.39	-4.31 – 1.25	1.37	1.13 – 5.63	6.43
5	Penchovsky and Ackermann [77]	16	24	8.12	-4.33 – -0.84	0.52	2.03 – 6.07	6.75
6	Ackermann and Gast [1]	16	24	14.7	-9.83 – -1.53	2.87	4.31 – 7.64	8.38
7	random 20	20	21	–	-3.3 – 3.82	4.35	3.58 – 10.06	10.69
8	Deaton et al. [22]	20	7	–	-1.05 – 2.57	5.31	6.47 – 8.8	10.26
9	Deaton et al. [24]	20	40	+	-1.62 – 3.48	4.99	5.83 – 8.71	9.89
10	Feldkamp et al. [30] 1	20	7	–	1.43 – 3.21	4.5	7.19 – 9.51	11.09
11	Feldkamp et al. [30] 2	20	7	–	-2.87 – 1.64	5.08	7.67 – 8.79	11.09
12	Feldkamp et al. [30] 3	20	14	–	-5.09 – 2.1	5.15	6.98 – 11.63	11.93
13	Tanaka et al. [90]	20	14	+	-8.28 – 2.05	3.82	4.5 – 8.47	8.83
14	Shin et al. [85]	20	7	–	1.43 – 2.5	3.36	5.04 – 8.05	8.86
15	Adleman [2]	20	7	–	-0.38	-0.38	9.27	10.81
16	negative control 1	20	7	–	-18.6	-12.23	-1.53	-1.53
17	negative control 2	20	7	–	0.4	0.87	1.59	1.59

* negative energy gap indicates overlapping in energy range of intended and unintended complexes, that means some unintended complexes are more stable than intended ones.

A negative or small positive value of τ^b, such as for one of the encodings in set 1 or negative control encodings 16 and 17, indicates the high likelihood of false positives, i.e., strands that do not represent a valid path, but are recognized as the ones in the result of the DNA computation.

In general, the evaluation shows the expected results, namely: significant distinctions between energy gap measured by internal evaluation in the process of strand design δ and energy gap for brick-based scheme δ^b, and the improved performance of the encodings for the assignment graph (columns seven and nine) compaired to that for the complete one (columns six and eight) in both phases. Thus, the energy gaps δ^b and τ^b for the complete graph are good estimations for highly dense graphs (with number of edges close to the maximum). The density of the assignment graph is 0.3, which is the ratio between number of edges present in a graph and the maximum.

The performance of each word set was analyzed with the help of an additional procedure that retrieved the unintended hybridization complexes with MFE smaller than $E^b_{\text{int}}(v_i)$ or $E^b_{\text{int}}(e_{ij})$ for a given encoding. Since all the predesigned sets are PCR-optimizing, it is to expect that the vertex-words do not hybridize with each other, that is, the complexes v_i/v_j have high MFE. The subject of interest is the behaviour of the edge-words, since they are not present in predesigned word sets. The additional source of undesired behaviour (negative energy gaps) are the edges e_{0i}, e_{i6} and e_{06}. Since these edges are $l{+}l/2$ nucleotides long, they have a capacity to attain by interaction with the other molecules an energy value smaller than that of $l/2$ base pairs in intended hybridizations between edge- and vertex-strands of length l.

The common feature of the encodings from the literature is that the hybridization energy between sequences contained in sets (vertex-words) is higher than the upper range of intended hybridizations $E^b_{\text{int}}(v_i)$, as expected. The out-of-range energy values (negative δ^b) are exhibited mostly by hybridizations of the strands representing edges with each other or with vertex-strands. For the encodings from the set 5 such hybridizations occur only between edge- and vertex-strands; for the set 9 – by interaction of e_{0i} and e_{i6} with other edge-strands.

For Adleman's computation the performance of randomly generated encodings is comparable with that of the published word sets for each word length. As can be seen from the performance of the seven-word sets 8, 10, 11, and 14, the order of words is an important factor in Adleman's experiment, due to the special encoding of the terminal edges. From the evaluation of the encoding used by Adleman, it can be seen that a small negative value of δ^b is compensated by the next phase of computation reflected by high τ^b.

The 12 nt encodings are hardly appropriate for Adleman's experiment, due to low negative δ^b and relative low τ^b. However, encoding of this word length specifically generated for the given experiment settings may show better perfor-

mance. The performed evaluation shows that 16- and 20 nt encodings are more appropriate. This result conforms to that of the other works in the area of DNA computations.

3.4 Discussion

In this chapter an evaluation subroutine for a DNA encoding as a separate standalone module has been proposed and the detailed workflow for development of such modules was presented. Such an approach allows to assess the *in vitro* performance of the encoding for DNA computing experiments according to special requirements of the particular protocol. The main features of the proposed approach are:

1. the evaluation is performed for a particular reaction scheme;
2. the implemented module exploits external software for MFE calculations.

The first feature allows to prove applicability of the available DNA word sets – published or generated by certain strand design methods, for the intended DNA computation. For instance, the word sets developed under combinatorial constraints [22, 66] can be evaluated considering thermodynamic properties; or to check whether the sets of fixed-length words can be used as a basis for computations that require strands of different lengths.

The second feature simplifies the implementation of the evaluation procedure. Moreover, it allows to evaluate the performance of the encoding in chemical environment with different salinity and temperature. This option can be used to refine the protocol of intended DNA computations.

The computational model of Adleman was analyzed by Deaton et al. [22] in 1996. However, their approach considers only mismatched hybridizations and is based on Hamming distance. The performed evaluation scheme considers additionally the possibility of the frame shifts, and the assessment is based on more exact free energy criteria.

An alternative approach to evaluation of encoding for brick-based computations presented in this chapter would be considering of trimolecular complexes (Figure 3.2a) in hybridization/ligation phase. Then the composition of the memory strands participating in the calculation phase can be more accurately assisted. However, the available MFE computation methods do not consider such composition of molecules. Even with available corrections for intended complexes, the energy calculations for unintended ones would imply heavy approximations that make the evaluation results ambiguous.

The proposed evaluation modules can be applied as part of more complex software. The module has significantly lower computational complexity than

full-scale simulation with EdnaCo [35] or similar software (Section 2.6.1). Thus, it can be employed for preliminary selection of the candidate encoding in such applications. Furthermore, the evaluation modules can be applied for refinement of the range values of the constraints that are used by internal evaluation in strand design methods.

Chapter 4

Graph-based MFE Algorithms for DNA Hybridization Complex

This chapter describes a new method for solving the minimal free energy (MFE) problem for two piecewise complementary single strands of DNA. Its basic concept – the *hybridization graph* – is introduced in Section 4.1. This graph is built to comprise unambiguously all possible secondary structures of a DNA/DNA complex. Thus, the MFE problem is the same as that of finding a path with minimal weight in this graph. This task is solved using the paradigm of dynamic programming. First, a direct exhaustive search of the graph is described. This leads to dimensional explosion with the increasing length of input DNA sequences. Consequently, there were developed two advanced methods, which are demonstrated later in the chapter. They employ principally distinct techniques for dynamical pruning of the graph to decrease the computational complexity of the direct approach.

4.1 Hybridization Graph

Hybridization model

Prior to the description of computational methods, a biological model is presented.
For pairing of two DNA molecules, an interstrand DNA hybridization model is considered. This model states that two DNA single strands will anneal in parallel without preliminary or a posteriori self-folding (Figure 4.1). So, the model considers exceptionally intermolecular interactions between two single stranded DNA molecules. As both ssDNA are only piecewise complementary, the DNA/DNA hybridization complexes under consideration contain structural motifs of three types: *stems* (i.e., consecutive regions of paired bases); *loops* (i.e., regions of unpaired bases including bulges, symmetric and asymmetric loops); and *terminal*

motifs (i.e., dangling ends and terminally unmatched bases) (Figure 4.2).

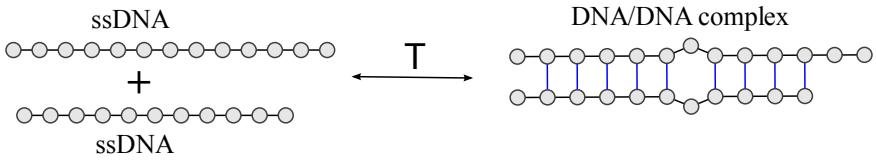


Figure 4.1: Interstrand DNA hybridization. The DNA/DNA complex contains a single mismatch.

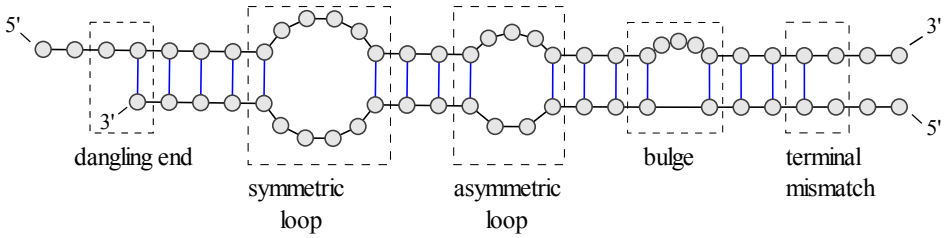


Figure 4.2: Structural motifs of DNA/DNA hybridization complex. Vertical lines denote paired Watson-Crick complementary bases (A-T and C-G).

Hairpins (Figure 4.3a), multibranched loops (Figure 4.3b), and pseudoknots are subject to folding algorithms and not considered under the interstrand hybridization model.

(a) hairpin (b) multibranched loop

Figure 4.3: Example of hairpin and multibranched loop structures.

To assess the stability of potential secondary structures of DNA/DNA complexes, the nearest neighbour thermodynamic model (Section 2.1.2) is employed. So, the objective value of the MFE problem is given as follows:

$$E_{\min} = \min \left\{ \sum_{i=1}^{k} E_{\text{stem}}(S_i) + \sum_{j=1}^{m} E_{\text{loop}}(L_j) + E_{\text{term}} \right\}, \tag{4.1}$$

where the minimum is to find over all admissible structures of the bimolecular complex for two given DNA strands; S_i and L_j are potential stems and loops,

respectively; the term E_{term} accounts for energy contribution of the both left and right termini of the complex. The structure is admissible if in strands a and a' for all base pairs (a_i/a'_j) and (a_{i_1}/a'_{j_1}) holds: if $i < i_1$, then $j < j_1$. That is, the self-folding of the molecules is not allowed.

Hybridization graph

The background of the graph model developed is the representation of all possible structures of the hybridization complex for two ssDNA as a directed weighted graph – the *hybridization graph*. The main idea of constructing the graph is to enrich the base pairing matrix, which incorporates all pairs of complementary nucleotides for two given DNA sequences, with information about all possible contained DNA structural motifs. In this graph, each edge represents such a motif. In this way, a path corresponds to the secondary structure of the DNA/DNA complex. In particular, a path with minimal weight describes secondary structure with lowest free energy.

Given two sequences over the DNA alphabet $a = a_1 \dots a_m$ and $b = b_1 \dots b_n$, let assume without restriction that $m \leqslant n$. The bases in b are denoted in $5' - 3'$ order, while the bases in a are in reverse $3' - 5'$ order to comprise the opposite direction of DNA strands by hybridization. For them, a *hybridization graph* $G = (V, E)$ is a weighted directed graph built as follows.

Vertices are located on a two-dimensional $m \times n$ grid (Figure 4.4). The rows and columns are labelled by the nucleotides of the DNA sequences a and b, respectively. A node (i, j) on the grid defines a *vertex* of the hybridization graph if the bases a_i and b_j are Watson-Crick complementary, $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$. The vertices of the hybridization graph form a subset of the nodes in the grid and directly correspond to the base-pairing matrix used in early approaches for RNA folding [93, 103].

Each ***edge*** $\binom{i',j'}{i,j}$ links vertex (i', j') with vertex (i, j), provided that the following two conditions are met:

- $i' < i$ and $j' < j$; that is, there are no horizontal or vertical edges on the grid, so each base is uniquely paired in a DNA/DNA complex;
- there is no vertex (i'', j'') with $i' < i'' < i$ and $j' < j'' < j$, such that $i'' = i'+1$ and $j'' = j'+1$ or $i'' = i-1$ and $j'' = j-1$; that is, (i', j') has no succeeding vertex on the diagonal $i'-j'$, and (i, j) has no preceding vertex on the corresponding diagonal; this ensures that each edge corresponds to exactly one structural motif.

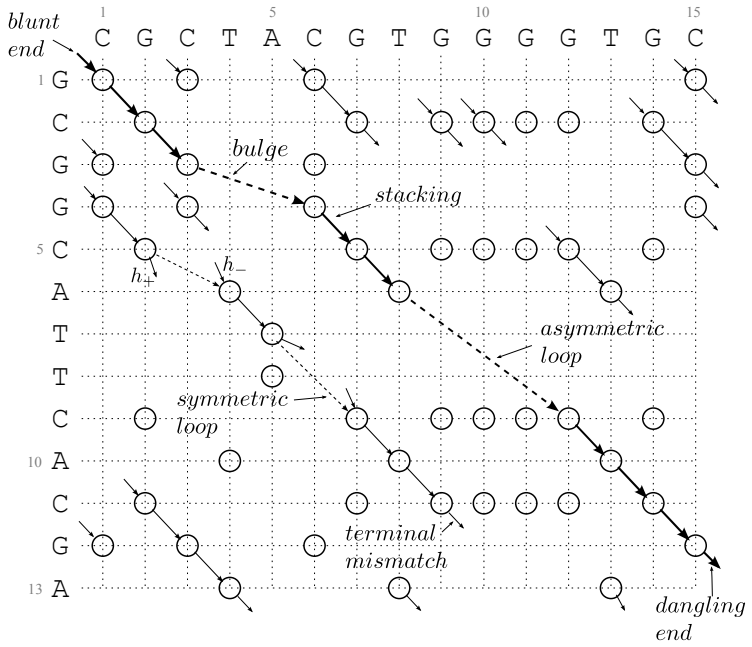
Half-edges, that is, edges with either an initial or a terminal vertex, are set as follows:

- half-edge $h_-(i, j)$ enters a vertex (i, j) if the node $(i-1, j-1)$ is not a vertex;

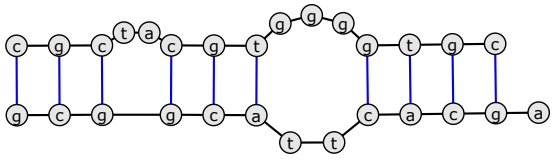
they represent terminal motifs on the left end of a DNA/DNA complex;

– half-edge $h_+(i,j)$ leaves a vertex (i,j) if the node $(i{+}1, j{+}1)$ is not a vertex; they represent terminal motifs on the right end of a DNA/DNA complex.

The graph, built in this way, is acyclic, since each edge $\binom{i',j'}{i,j}$ satisfies $i' < i$ and $j' < j$.



(a) hybridization graph



(b) optimal DNA/DNA complex structure

Figure 4.4: Example of hybridization graph and corresponding bimolecular complex.
(a) The hybridization graph for the sequences $a = 3'$-$GCGGCATTCACGA$-$5'$ and $b = 5'$-$CGCTACGTGGGGTGC$-$3'$ on the two-dimensional grid (not all edges are shown). The total number of edges is 378. The potential stackings are marked by solid arrows. The optimal path goes from the upper left vertex (1,1) to the lower right vertex (12,15) and is shown in bold.
(b) Optimal DNA/DNA hybridization complex corresponding to the graph. It contains three stems, one bulge, one asymmetric loop, blunt end at the left terminus and dangling end at the right one. Minimum free energy of the complex is $E_{\text{total}} = $-4.46 kcal/mol.

Weights of edges and half-edges are assigned according to the free energy of the associated structural motifs (Equations (2.3) – (2.8) in Section 2.1.2). Moreover, the hybridization graph exhibits a one-to-one correspondence of the geometry of its edges with DNA structural motifs (Figure 4.4). Thus, the weights w_e and w_h for edges e and half-edges h, respectively, are also associated with their geometry as follows:

– Diagonal edges with $i-i' = j-j' = 1$ provide Watson-Crick nearest neighbours or *stackings*. Their weights are

$$w_e\binom{i-1,j-1}{i,j} = w_s(i,j) = \triangle G_{\mathrm{NN}}(b_{j-1}b_j/a_{i-1}a_i).$$

The weights w_s are always negative for thermodynamic reasons. Consecutive stackings form a potential stem.

– Diagonal edges with $i-i' = j-j' > 1$ form symmetric loops with the weight

$$w_e\binom{i',j'}{i,j} = w_{\mathrm{sLoop}}\binom{i',j'}{i,j} = E_{\mathrm{symLoop}}(L_{i,j}^{i',j'}).$$

– Nondiagonal edges with $i-i' \neq j-j'$, where $j-j' > 1$ and $i-i' > 1$ form asymmetric loops with weight

$$w_e\binom{i',j'}{i,j} = w_{\mathrm{aLoop}}\binom{i',j'}{i,j} = E_{\mathrm{asymLoop}}(L_{i,j}^{i',j'}).$$

– Nondiagonal edges with $i-i' = 1$ and $j-j' > 1$ build bulged loops with weight

$$w_e\binom{i-1,j'}{i,j} = w_{\mathrm{bulge}}\binom{i-1,j'}{i,j} = \triangle G_{\mathrm{bulge}}(B_{i,j}^{i-1,j'});$$

the ones with $j-j' = 1$ and $i-i' > 1$ build bulged loops with weight

$$w_e\binom{i',j-1}{i,j} = w_{\mathrm{bulge}}\binom{i',j-1}{i,j} = \triangle G_{\mathrm{bulge}}(B_{i,j}^{i',j-1}).$$

– For edges representing loops, there is also a correlation between their length and weight. Since the corresponding energy terms $\triangle G_{\mathrm{loop}}$ increase with the number of unpaired bases between the flanking base pairs, the positive constituents of the weight of corresponding edges increases with their length.

– Half-edges represent the following terminal motifs:

- blunt ends for vertices $(1,1)$ or (m,n);
- $5'$- or $3'$- dangling ends for other vertices on the grid boundaries;
- terminal mismatches for inner vertices on the grid.

Their weights are assigned according to the represented motif as follows:

$$w_h(i,j) = \begin{cases} \triangle G_{\mathrm{bEnd}}(b_j, a_i), & \text{if } (i,j) = (1,1) \text{ or } (i,j) = (m,n), \\ \triangle G_{5'-\mathrm{dEnd}}(b_j, a_i), & \text{if } j = n \text{ and } 1 < i < m \\ & \qquad \text{or } i = 1 \text{ and } 1 < j < n, \\ \triangle G_{3'-\mathrm{dEnd}}(b_j, a_i), & \text{if } i = m \text{ and } 1 < j < n \\ & \qquad \text{or } j = 1 \text{ and } 1 < i < m, \\ \triangle G_{\mathrm{termMM}}(b_j, a_i), & \text{otherwise.} \end{cases}$$

Paths in the hybridization graph

The model of the hybridization graph describes uniquely and completely all possible structural motifs for the DNA/DNA hybridization complex. A path in this graph represents a sequence of consecutive structural motifs, so the shortest path (with the lowest weight) corresponds to the MFE structure of the bimolecular complex. However, the hybridization complex is flanked on both sides with terminal motifs. So, not all the paths in the graph can represent a proper secondary structure. The developed graph model complies with the property of the bimolecular complex without additional restrictions; that is, the terminal vertices of the shortest path always define the terminal DNA motifs.
Namely, if a path $P((i_1, j_1), \dots)$ starts at some vertex (i_1, j_1) not incident to entering half-edge, then the node $(i_1{-}1, j_1{-}1)$ represents a vertex. Then there exists a path $P_1((i_1{-}1, j_1{-}1), (i_1, j_1), \dots)$, that coincides with P from the vertex (i_1, j_1), with the weight $W(P_1) = w_s(i_1, j_1) + W(P)$. This weight is smaller than that of P, since the term w_s is negative, according to the energy of Watson-Crick nearest neighbours. Thus, the path P is *a priori* not the shortest one.
Similarly, if some other path $Q(\dots, (i_k, j_k))$ ends at vertex (i_k, j_k), not incident to a leaving half-edge, then the node $(i_k{+}1, j_k{+}1)$ is a vertex. The weight of a path $Q_1(\dots, (i_k, j_k), (i_k{+}1, j_k{+}1))$, that coincides with Q up to (i_k, j_k), is: $W(Q_1) = W(Q) + w_s(i_k{+}1, j_k{+}1)$, and smaller than $W(Q)$. Hence, such a path Q is also suboptimal.
Thus, both computational and structural reasons allow to restrict the number of separately considered paths as follows.
The proper ***paths*** representing the structure of a DNA bimolecular complex starts at some vertex (i_1, j_1), for which a node $(i_1{-}1, j_1{-}1)$ is not a vertex, and ends at a vertex (i_k, j_k) if the node $(i_k{+}1, j_k{+}1)$ is not a vertex. The weight of such path is

$$W(P) = w_{h_-}(i_1, j_1) + \sum_{t=1}^{t=k-1} w_e\left(\begin{smallmatrix} i_t, j_t \\ i_{t+1}, j_{t+1} \end{smallmatrix}\right) + w_{h_+}(i_k, j_k).$$

As only such paths in the hybridization graph are considered, they are referred

further on simply as paths. The other consecutive sequences of edges are denoted as subpaths or path segments.

Special properties of the elements in the hybridization graph

Considering the special role of the terminal vertices of the paths, the following vertex classification is introduced:

- *Left terminal* vertices, denoted $(i,j)_L$, for which $(i{-}1,j{-}1)\notin V$; for example, vertices $(1,1)$, $(6,4)$, and $(9,7)$ in Figure 4.4. They are incident to entering half-edges and open potential stems.
- *Non-left-terminal* vertices (i,j), for which $(i{-}1,j{-}1)\in V$, are denoted as $(i,j)_{nL}$; for example, vertices $(2,2)$, $(5,7)$, and $(11,14)$ in Figure 4.4.
- *Right terminal* vertices, denoted $(i,j)_R$, for which $(i{+}1,j{+}1)\notin V$; for example, vertices $(2,7)$, $(5,2)$, and $(7,5)$ in Figure 4.4. They are incident to leaving half-edges and close potential stems.
- *Non-right-terminal* vertices (i,j) with $(i{+}1,j{+}1)\in V$, are denoted as $(i,j)_{nR}$; for example, vertices $(5,7)$, $(5,12)$, and $(10,13)$ in Figure 4.4.

The given classification of vertices contains overlaps. The set of non-left-terminal vertices includes the right terminal ones. Some vertices (i,j) are simultaneously left and right terminal if they have no consecutive predecessor and ancestor on diagonal of (i,j), e.g., vertices $(2,9)$, $(2,10)$, and $(4,3)$ in Figure 4.4. Note that the definitions of the first two types vs. the last two ones are made in different contexts, namely, regarding their entering and leaving edges respectively. In the context of edges entering (i,j) the presence of predecessor $(i{-}1,j{-}1)$ only is significant. Similarly, the context of leaving edges gives significance to the presence of a successor vertex $(i{+}1,j{+}1)$. These contexts are treated independently in the course of calculations and the overlaps do not raise any actual contradictions.

The vertex differentiation leads to the following basic properties for edges and paths:

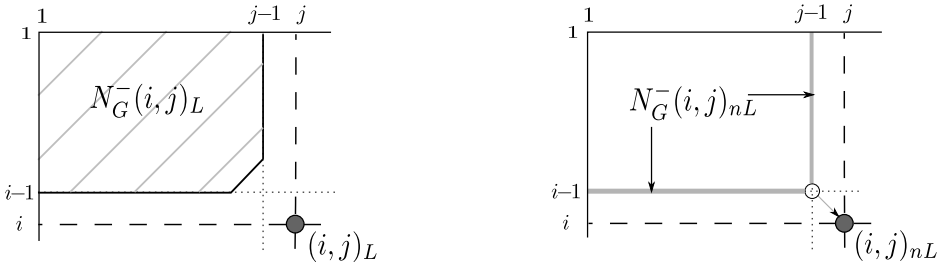
- symmetric and asymmetric loops start at the right terminal vertex and end at the left terminal vertex, according to the edge setting;
- a proper path has the form $P((i_1,j_1)_L,\ldots,(i_k,j_k)_R)$, where $1\leqslant k\leqslant m$.

Regarding the last property, the shortest path problem for the hybridization graph can be restated as reduced shortest path problem with a set L of the start vertices and a set R of the end vertices, which contain the left and right terminal vertices, respectively. So, the MFE problem for DNA/DNA hybridization complex can be solved as $L{-}R$ *shortest path problem* in the hybridization graph.

Considering the definition of the edges, vertices of each type exhibit characteristic neighbourhoods shown in Figures 4.5 and 4.6:

– the in-neighbourhood of $(i,j)_L$ consists of vertices $(i,j)_R$ in the area, shaded in Figure 4.5a, and all vertices on the right and bottom borders of the area;

– the in-neighbourhood of $(i,j)_{nL}$ consists of all vertices on the two segments, shown in Figure 4.5b;

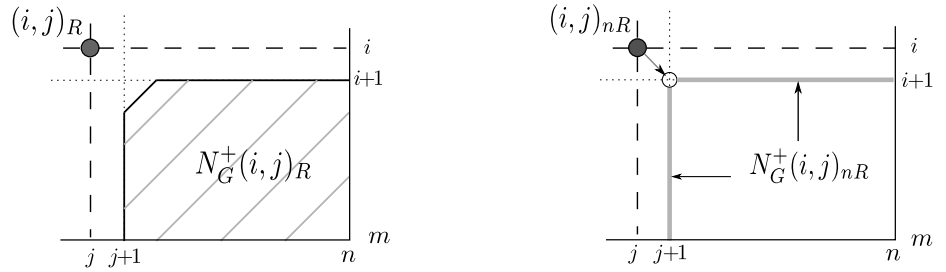


(a) in-neighbourhood area of $(i,j)_L$ (b) in-neighbourhood area of $(i,j)_{nL}$

Figure 4.5: Characteristic in-neighbourhood for (a) left terminal and (b) non-left-terminal vertices.

– the out-neighbourhood of $(i,j)_R$ consists of vertices $(i,j)_L$ of the area, shaded in Figure 4.6a, and all vertices on the left and upper borders of the area;

– the out-neighbourhood of $(i,j)_{nR}$ consists of all vertices on the two segments, shown in Figure 4.6b.



(a) out-neighbourhood area of $(i,j)_R$ (b) out-neighbourhood area of $(i,j)_{nR}$

Figure 4.6: Characteristic out-neighbourhood for (a) right terminal and (b) non-right-terminal vertices.

Notation

To keep the further explanations brief and precise, the following conventions are defined for some expressions used throughout this chapter:

– Index i is used to denote rows, so $1 \leqslant i \leqslant m$. Index j is for columns and ranges in interval $1 \leqslant j \leqslant n$.

– Superscripts, such as i' or j' are used to denote indexes preceding i or j, and subscripts, such as i_1 or j_1 for indexes succeeding i or j.

– A pair (i, j) denotes a vertex of the hybridization graph, if it is not stated explicitly as a grid node.

– *Rectangle* $(i, j){:}(i_1, j_1)$ describes the rectangular area of the two-dimensional grid with corners at nodes $(i, j), (i, j_1), (j, i_1)$, and (i_1, j_1); in notation, only upper left and lower right corners are given.

4.2 Shortest Path Algorithms for Hybridization Graph

Dynamic programming is applied as a basic approach for solving the $L{-}R$ shortest path problem for the hybridization graph. Section 4.2.1 describes the direct application of this method, while Sections 4.2.3 and 4.2.4 demonstrate two advanced algorithms that reduce the complexity of the direct application.

The general steps of an algorithm following the paradigm of dynamic programming were described in Section 2.2. Such an algorithm consists of a forward and a backward part. The forward one evaluates the data (e.g., edge weights) that are used by the back tracing to determine the optimal sequence (e.g., path) [11]. Next, the general steps of the dynamic programming algorithm are restated in the context of the $L{-}R$ shortest path problem for the hybridization graph.

The ***forward part*** associates an optimal score $T_{i,j}$ with each vertex (i, j) of the hybridization graph. This score represents the lowest weight of a subpath (starting at left terminal vertex) to this vertex.
The *initiation* of the DP matrix consists in assigning to the vertices on the first row and column the weights of their entering half-edges h_- (Equation (4.2)):

$$T_{i,j} = w_{h_-}(i, j), \quad \text{if } i = 1 \text{ and } 1 \leqslant j < n, \text{ or } j = 1 \text{ and } 1 < i < m. \tag{4.2}$$

The *matrix filling* procedure consists in finding the minimal score for every vertex, using previously calculated scores of its direct predecessors. For this, from all the entering edges of the vertex the one providing the optimal score for it is found. In other words, we search for the best direct predecessor for every vertex. So, the recursion for dynamic programming is given as follows:

$$T_{i,j} = \min \begin{cases} \min \left\{ T_{i',j'} + w_{\mathrm{e}}\binom{i',j'}{i,j} \right\}, & \text{for all } (i', j') \in N_G^-(i, j), \\ w_{\mathrm{h}_-}(i, j), & \text{if } (i, j) \text{ has an entering half-edge } h_-, \end{cases} \tag{4.3}$$

where minimum ranges over all in-neighbours (i', j') of the vertex (i, j).

The *weight of the L−R shortest path* is found as the lowest weight score among all right terminal vertices after the filling of the DP matrix:

$$W_{\min} = \min_{(i,j)_R} \left\{ T_{(i,j)} + w_{\mathrm{h}_+}(i,j) \right\},$$

where the minimum ranges over all right terminal vertices $(i,j)_R$, and $w_{\mathrm{h}_+}(i,j)$ is the weight of a half-edge leaving the corresponding vertex $(i,j)_R$.

The ***backward part*** restores the sequence of edges in the path with the lowest weight found $W_{\min}$. This operation delivers an ordered list of vertices. For this, the best predecessor for every vertex is stored in addition to its minimal score, while filling the matrix. The backtrace consists of consecutive extraction of such predecessors, starting from the right terminal vertex with the lowest path weight $W_{\min}$, found after the filling step of forward iteration.

The initiation of the DP matrix, finding of optimal path weight, and back tracing procedure are implemented in the same way for all the three developed computational methods. The Sections 4.2.1 to 4.2.4 describe the approach in general and the matrix filling procedure for each of them. All three methods exploit the vertex differentiation according to the types given above.

4.2.1 Exhaustive Search

In general, an exhaustive search is a brute force approach which goes per definition through the complete solution space to find the optimal one. The calculation course presented exploits the dynamic programming paradigm reusing optimal solutions of subproblems, that means the total solution space is already reduced. However, in the given case all edges entering a vertex are considered to find the best score. Hence, the approach is exhaustive in this respect.

The general scheme of the matrix filling step is given in Figure 4.7. The procedure is similar to the common DP approach for sequence alignment. The main distinction is that, by those methods the predecessors are always one node away, whereas now they are one edge away.
Computation of the optimal score $T_{i,j}$ for vertex (i,j) is performed by iterating over all its direct predecessors (i',j') to find the shortest subpath to this vertex. As the method is based on exploring the in-neighbourhood, and considering that the in-degree of left terminal vertices $(i,j)_L$ is usually higher than that of the non-left-terminal ones $(i,j)_{nL}$, it is reasonable to establish separate procedures for each of these vertex types.

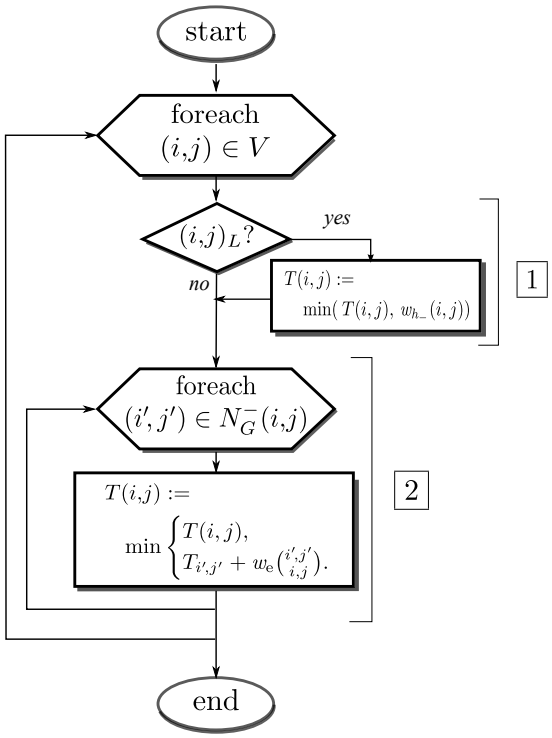


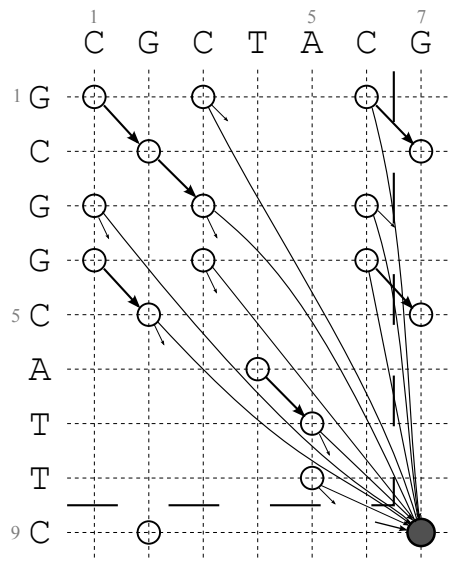
Figure 4.7: Flow chart for filling the DP matrix by forward iteration. It describes calculation flow for an exhaustive search and vertex pruning methods. Conditional block 1 shows the additional calculation of the weight of entering half-edge for the left terminal vertices. To find the optimal score for every vertex (i, j) its in-neighbourhood is explored (block 2).

Filling of the matrix

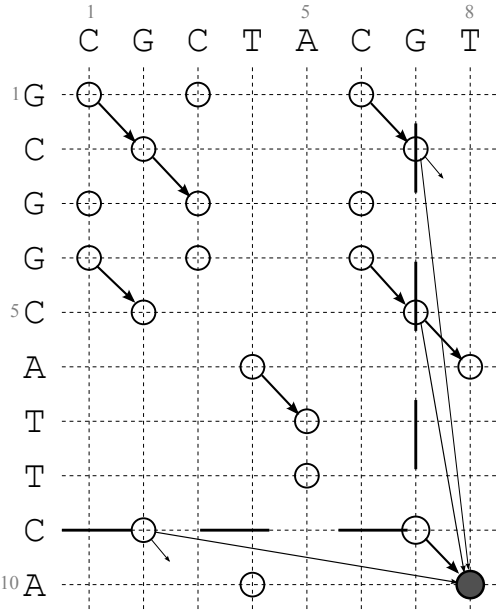
After initialization of the vertices in the first column $(i, 1)$ and in the first row $(1, j)$ as given in Equation (4.2), the remaining vertices of the hybridization graph are differentiated by their in-neighbourhoods. Thus, two different iteration courses are defined for the left terminal vertices $(i, j)_L$ and the non-left-terminal ones $(i, j)_{nL}$ as follows:

1. A left terminal vertex $(i, j)_L$ (Figure 4.8a) has entering edges of the following groups:

- bulges with initial vertex (i', j'), so that either $i-i' = 1$ or $j-j' = 1$;
- symmetric or asymmetric loops, their starting vertices $(i', j')_R$ are found in the rectangle $(1, 1):(i-2, j-2)$;
- a half-edge.



(a) left terminal vertex $(i,j)_L$



(b) non-left-terminal vertex $(i,j)_{nL}$

Figure 4.8: Entering edges for left terminal and not left terminal vertices. Both belong to the same stem and are placed consecutively on the diagonal, so (b) shows an area one row and column larger than (a). Diagonal edges represent stackings and are marked by bold lines.
(a) Entering edges of a left terminal vertex (shaded). This is vertex (9,7) of the graph given in Figure 4.4. The direct predecessors are in the dashed area. There is one entering half-edge; the remaining edges provide loop motifs.
(b) Entering edges for a non-left-terminal vertex (10,8) in the graph of Figure 4.4. Only the vertices on the dashed lines are its direct predecessors. There is one entering stacking edge; the remaining edges provide bulges.

Thus, the optimal weight of subpath to the vertex $(i,j)_L$ is given by the following equation:

$$T_{(i,j)_L} = \min \begin{cases} w_{h\_}(i,j), \\ T_{\text{bestBulge}}(i,j), \\ T_{\text{bestLoop}}(i,j), \end{cases} \tag{4.4}$$

where $w_{h\_}$ is the weight of entering half-edge; $T_{\text{bestBulge}}$ is the lowest weight provided by entering bulge; and T_{bestLoop} is the lowest weight provided by entering loop. They are calculated as follows:

$$T_{\text{bestBulge}}(i,j) = \min \begin{cases} \min\limits_{1 \leqslant i' \leqslant i-2} (T_{i',j-1} + w_{\text{bulge}}(\begin{smallmatrix} i',j-1 \\ i,j \end{smallmatrix})), \\ \min\limits_{1 \leqslant j' \leqslant j-2} (T_{i-1,j'} + w_{\text{bulge}}(\begin{smallmatrix} i-1,j' \\ i,j \end{smallmatrix})). \end{cases}$$

The value for T_{bestLoop} is calculated from each vertex $(i', j')_R$ in rectangle $(1,1)$:$(i-2, j-2)$, according to the definition of $N_G^-(i,j)_L$:

$$T_{\text{bestLoop}}(i,j) = \min_{\substack{1 \leqslant i' \leqslant i-2, \\ 1 \leqslant j' \leqslant j-2}} (T_{i',j'} + w_{\text{loop}}(\tbinom{(i',j')_R}{i,j})),$$

where w_{loop} is the weight of a corresponding loop $e(\tbinom{(i',j')_R}{(i,j)_L})$. The type of the loop and the edge weight calculation depends on the first vertex $(i', j')_R$ in the loop and the last one $(i,j)_L$:

$$w_{\text{loop}}(\tbinom{(i',j')_R}{i,j}) = \begin{cases} w_{\text{sLoop}}(\tbinom{i',j'}{i,j}), & \text{if } i-i' = j-j' \geqslant 2, \\ w_{\text{aLoop}}(\tbinom{i',j'}{i,j}), & \text{otherwise.} \end{cases}$$

2. If a vertex (i,j) is not left terminal (Figure 4.8b), it has entering edges of the following groups:

– stacking edge from the vertex $(i-1, j-1)$;

– bulges with initial vertex (i', j'), so that either $i-i' = 1$ or $j-j' = 1$.

Thus, the weight score for the vertex $(i,j)_{nL}$ is defined as

$$T_{(i,j)_{nL}} = \min \Big\{ T_{i-1,j-1} + w_{\text{s}}(i,j), T_{\text{bestBulge}}(i,j). \tag{4.5}$$

The general weight score of the vertex (i,j) is then given as

$$T_{i,j} = \begin{cases} T_{(i,j)_L}, & \text{if } (i,j) \text{ is a left terminal vertex,} \\ T_{(i,j)_{nL}}, & \text{otherwise.} \end{cases}$$

Complexity of exhaustive search

As can be seen from the course of computation, by finding the shortest path through the hybridization graph all edges are observed once. Thus, the graph size is a main factor for the algorithm complexity, which amounts to $O(|E|)$. The worst case constitutes a graph with all left terminal vertices, due to their higher in-degree compared to that of the other vertices. A vertex layout of such a graph with maximal vertex coverage on the grid exhibits the pattern shown in Figure 4.9. Shifting it one column horizontally or rotating it 90° corresponds with swapping of the sequence a with b and results in a graph of the same size. The DNA sequences producing the pattern comply with the following scheme:

– sequence a is a repetition of the same nucleobase N, that is $a = (N)_m$, with $N \in \Sigma = \{A, C, G, T\}$;

– sequence b contains on every odd position a base $\overline{N}$ complementary to N; its even positions are occupied with bases b_j, where $b_j \in \Sigma = \{A, C, G, T\} \setminus \{\overline{N}\}$; that is, with any base except the complement of N.

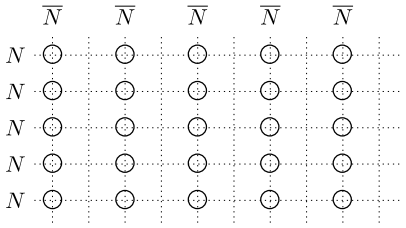


Figure 4.9: Vertex layout for worst case hybridization graph.

Such base composition exhibits maximal coverage of the grid with $n^2/2$ left terminal vertices, where n is the length of DNA sequence. Each of them has $(i-1)\times(j-1)/2$ entering edges. Thus, the graph size amounts approximately to n^4, and worst case complexity of exhaustive search is $O(n^4)$.

4.2.2 Reduction of the Hybridization Graph

The exhaustive search described in the previous section is a direct application of the dynamic programming paradigm to the shortest $L-R$ path problem on the hybridization graph. The computational complexity $O(|E|)$ of this approach amounts in the worst case to $O(n^4)$ for DNA sequences of length n. This is comparable with the complexity of Zuker and Stiegler's algorithm [112]. However, for the purposes of large-scale applications of DNA strand design, such complexity is impractical. This section outlines a general approach, that reduces the amount of calculations for every particular graph. Two distinct computational methods, based on the approach, are presented further in Sections 4.2.3 and 4.2.4.

The exhaustive search method considers all edges of the hybridization graph. However, the main factor defining the shortest path is the weight of the edges, assigned according to their length and the bases of the input sequences. So, to reduce the amount of calculations for particular graphs, the general suggestion is to find the edges with a weight disadvantageous in a given context and delete them from the hybridization graph. The resulting spanning subgraph must retain the shortest path of the initial graph. Such subgraphs are further denoted as the *consistent* ones.

In the rest of this section a formal description for this approach is developed to allow its practical application.

Consider the dynamic programming expression for the $L-R$ shortest path prob-

lem in the hybridization graph:

$$T_{i,j} = \min \begin{cases} \min \left\{ T_{i',j'} + w_{\mathrm{e}}\binom{i',j'}{i,j} \right\}, & \text{for all } (i',j') \in N_G^-(i,j), \\ w_{\mathrm{h}_-}(i,j), & \text{if } (i,j) \text{ has an entering half-edge } h_-, \end{cases} \tag{4.6}$$

where the minimum ranges over all in-neighbours (i', j') of the vertex (i, j). The first sum in the expression represents a set of alternative arguments, which are defined by incoming edges of the vertex (i, j). An argument, corresponding to a particular edge $e = \binom{i',j'}{i,j}$ is further denoted as $T_e(e)$ – a *contribution* of the edge e to the function $T_{i,j}$ at its head:

$$T_e(e) = T_{i',j'} + w_{\mathrm{e}}\binom{i',j'}{i,j}. \tag{4.7}$$

As can be seen from Equation (4.6), only the edges with the lowest contribution $T_e(e)$ at each vertex are necessary to fill the DP matrix. It is obvious, that the number of such edges equals $|V|$ or slightly exceeds it, if there are multiple equal optima at some vertices. Hence, the rest of the edges can be considered as *auxiliary* ones. Deletion of the latter from the hybridization graph results in a consistent subgraph, since all edges leading to optimal score at the vertices and, consequently, the optimal path, are preserved. However, to ascribe a certain edge to one of these categories, a complete calculation of the optimal score for its head is often required. Hence, for establishing of a consistent subgraph, the a priory detectable auxiliary edges are of interest. They can be subsumed into the following class:

Inefficient edges: An edge $e = \binom{i',j'}{i,j}$ is *inefficient*, if the contribution of this edge to the function at its head $T_e(e) = T_{i',j'} + w_{\mathrm{e}}\binom{i',j'}{i,j}$ is not optimal and this feature can be detected in advance, that is, without explicit computation of the term $T_e(e)$, or, more exactly, of the edge weight $w_{\mathrm{e}}(e)$.

Detection of such edges is based on the following criteria:

1. The edge with positive contribution $T_e(e)$ is likely to be inefficient. Since the function $T_{i,j}$ is a minimum, a positive argument hardly constitutes an optimum value for the head.
2. The longer the edge, the higher the positive constituent of its weight. Since the geometric length of the edge on the grid reflects the number of unpaired bases in respective loop, there is direct dependence between the edge length and the corresponding positive energy constituents.
3. The edge with negative weight $w_{\mathrm{e}}(e)$ is not considered as inefficient. Since the weight of such edge depends on respective bases only (according to parameter database), it is hard to discard the possible optimality of its contribution $T_e(e)$ without calculation of the weight.

4. The inefficient edges can represent the following structural motifs: bulges or symmetric/asymmetric loops with more than two unpaired bases. Consider the quantitative features of the thermodynamic parameters for DNA structural motifs gathered in Table 4.1.

Table 4.1: Quantitative features of the thermodynamic parameters (given according to SantaLucia and Hicks [84])

Parameter type	Notation	Energy range (kcal/mol)
Watson-Crick nearest neighbours	$\triangle G_{NN}: w_s$	-2.24 – -0.58
terminal mismatches	$\triangle G_{\text{termMM}}: w_{h_-}, w_{h_+}$	-1.23 – -0.21
lowest energy of a bulge $\binom{GGC}{C-G}$	$\min(w_{\text{bulge}})$	1.76
internal mismatches $\triangle G_{\text{intMM}}$	$\triangle G_{\text{rightMM}}$ or $\triangle G_{\text{leftMM}}$	-1.11 – 1.33
loop length penalty ($l = l_1 + l_2 \geqslant 2$)[a]	$\triangle G_{\text{loop}}(l)$	0, 3.2, ...
loop asymmetry penalty ($s = l_1 - l_2$)	$E_{\text{asym}}(s) = \lvert s \rvert \times 0.3$	0, 0.3, ...

[a] l_1 and l_2 – number of unpaired bases in loop for sequences b and a, respectively; l – loop length, s – loop asymmetry.

As can be seen, the stackings and half-edges always have negative weights; the weight of the two-base loops represented by edges of the type $\binom{i,j}{i+2,j+2}$ may be positive or negative, as shown by the example below. So, according to the previous point, the inefficient edges represent the remaining motifs.

Example of the symmetric loops with the length two bases; they are represented by edges of the type $\binom{i,j}{i+2,j+2}$ (green arrows in Figure 4.10):

$E_{\text{loop}}\binom{GGC}{\dot{C}A\dot{G}} = \triangle G_{\text{rightMM}}\binom{GG}{\dot{G}\dot{C}} + \triangle G_{\text{leftMM}}\binom{GC}{A\dot{G}} = -0.52 - 0.25 = \text{-}0.77\,\text{kcal/mol}.$

$E_{\text{loop}}\binom{TAC}{\dot{A}C\dot{G}} = \triangle G_{\text{rightMM}}\binom{TA}{\dot{A}\dot{C}} + \triangle G_{\text{leftMM}}\binom{AC}{C\dot{G}} = 0.92 + 0.47 = 1.39\,\text{kcal/mol}.$

The base pairs are marked by dots.

Thus, a consistent subgraph is obtained by deletion of inefficient edges, which represent bulges or symmetric/asymmetric loops longer than two bases. The finding of the shortest path in the reduced graph requires less calculations than that in the initial hybridization graph. The consistent subgraph is established dynamically while filling the DP matrix, that is, the calculations of the contributions for inefficient edges are omitted.

For obtaining of the spanning subgraph two approaches were developed:

– Deleting all leaving edges for certain vertices. This strategy is implemented by the *vertex pruning* method shown in Section 4.2.3.

– Deleting of the certain leaving edges for every vertex. This strategy is implemented by the *edge pruning* method described in Section 4.2.4.

4.2.3 Vertex Pruning

This strategy is based on finding certain vertices, whose leaving edges are all inefficient. So, knowing that an edge starts at such vertex, the former can be skipped by the search of the optimal score for its head. As this method isolates certain vertices from their out-neighbours, it is called *vertex pruning*.

To illustrate further explanations, Figure 4.10 reproduces the hybridization graph from Figure 4.4. It shows some edges omitted for clarity in the initial figure.

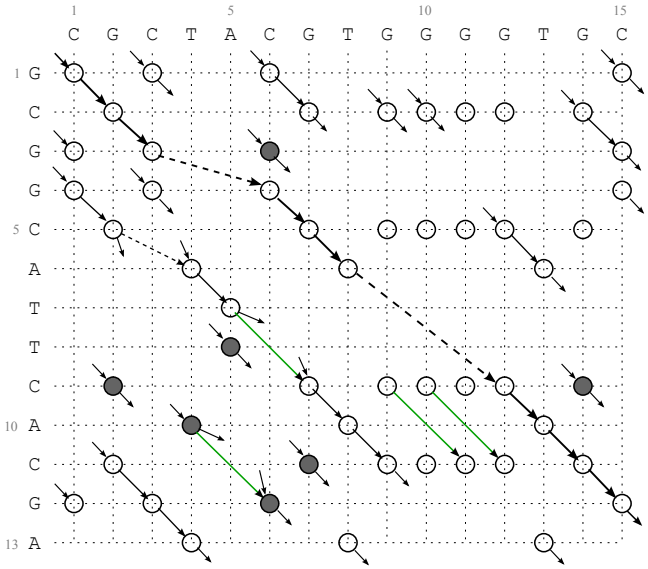


Figure 4.10: Extended representation of the hybridization graph from Figure 4.4. The edges $\binom{i,j}{i+2,j+2}$ representing two-base symmetric loops are shown in green. For the filled vertices all leaving edges are inefficient, except those in green.

The basic reasoning for this method is as follows. Consider the hybridization graph in Figure 4.10. It can be seen that some vertices, such as the ones in the first two rows and columns, or the vertices (3,6), (10,4), (5,9) – (5,11) possesses sparse/scarce or distant in-neighbours. Due to this, they acquire the score close to zero (positive or negative). On the other hand, their out-neighbours are more numerous and also distant, especially for the right terminal vertices. The combination of the small score (in absolute value) and the long leaving edges at such vertices (i, j) results in positive contribution $T_e(e)$ for their leaving edges $e = \binom{i,j}{i_1,j_1}$. Thus, the leaving edges of such vertices (i, j) are likely to be inefficient, according to the first criterion defined in Section 4.2.2. The criterion allowing to recognize such vertices (i, j) is stated by Proposition 4.2.1; its proof shows the inefficiency of the corresponding leaving edges. So, a consistent subgraph is established by deletion of the latter.

Pruning criterion for vertices

The criterion allowing to recognize such vertices (i, j) is stated as follows:

Proposition 4.2.1

> *If for some right terminal vertex $(i,j)_R$ the term $C = T_{i,j} + \triangle G_{\text{rightMM}}(b_j, a_i)$ is positive, then the contribution $T_e(e)$ of each of its leaving edges $e = \binom{i,j}{i_1,j_1}$ is not optimal at their respective heads, with the exception of the edge $\binom{i,j}{i+2,j+2}$ if it exists.*

The assertion can be illustrated by the vertex (10,4) in Figure 4.4. It possesses nine leaving edges, and the one drawn in green $\binom{10,4}{12,6}$ represents the exception to the proposition – a two-base symmetric loop.

Proof: To prove that the contribution $T_e(e)$ of an edge $e = \binom{(i,j)_R}{i_1,j_1}$ is not optimal for its head vertex (i_1, j_1), it should be shown, that the expression T_{i_1,j_1} contains an alternative case with the value lower than this contribution $T_e(e)$.

So, the following reasoning shows for all edges $\binom{(i,j)_R}{i_1,j_1}$ leaving a right terminal vertex, except the edge $\binom{i,j}{i+2,j+2}$, that:

1. if for the tail $(i,j)_R$ the term C is positive, the respective contribution $T_e(\binom{(i,j)_R}{i_1,j_1})$ is positive;
2. the expression T_{i_1,j_1} at the head contains an alternative case with a negative value.

This implies, that the positive contributions of the edges $\binom{(i,j)_R}{i_1,j_1}$ are not optimal for their respective heads and proves the assumption.

1. The edge $e = \binom{(i,j)_R}{i_1,j_1}$ leaving the right terminal vertex represents either a bulge or symmetric/asymmetric loop.

   – Consider the leaving bulges for the vertex $(i,j)_R$. The contribution of the bulge $e = \binom{(i,j)_R}{i_1,j_1}$ is: $T_e(e) = T_{i,j} + w_{\text{bulge}}(\binom{i,j}{i_1,j_1})$. If $C > 0$, this contribution is positive:

$$\begin{aligned} T_e(e) &= T_{i,j} + E_{\text{bulge}}(B^{i,j}_{i_1,j_1}) \\ &= \underbrace{T_{i,j} + \triangle G_{\text{rightMM}}(b_j, a_i)}_{C>0} + \underbrace{E_{\text{bulge}}(B^{i,j}_{i_1,j_1})}_{\geqslant 1.76} - \underbrace{\triangle G_{\text{rightMM}}(b_j, a_i)}_{\leqslant 1.33} > 0, \end{aligned}$$

   where the ranges for E_{bulge} and $\triangle G_{\text{rightMM}}$ are given according to the parameter database (Table 4.1).

   – Consider the leaving edges representing symmetric or asymmetric loops with more than two unpaired bases. If C is positive, their contributions are

positive:

$$\begin{aligned} T_e(e) &= T_{i,j} + E_{\text{loop}}(L^{i,j}_{i_1,j_1}) \\ &= \underbrace{T_{i,j} + \triangle G_{\text{rightMM}}(b_j, a_i)}_{C} + E_{\text{loop}}(l,s) + \triangle G_{\text{leftMM}}(b_{j_1}, a_{i_1}) \\ &= \underbrace{C}_{>0} + \underbrace{\triangle G_{\text{loop}}(l)}_{\geqslant 3.2} + \underbrace{E_{\text{asym}}(s)}_{\geqslant 0} + \underbrace{\triangle G_{\text{leftMM}}(b_{j_1}, a_{i_1})}_{\geqslant -1.11} > 0, \end{aligned}$$

where $E_{\text{loop}}(l,s)$ is a composite term combining penalties of the loop length $\triangle G_{\text{loop}}(l)$ and its asymmetry $E_{\text{asym}}(s)$; their ranges are given for the case $l \geqslant 3$, excluding the symmetric loop with two unpaired bases (Table 4.1).

Thus, the the contributions $T_e\binom{(i,j)_R}{i_1,j_1}$ of the edges leaving vertex $(i,j)_R$, which complies to the condition of the proposition, are positive.

2. Consider the out-neighbours of the vertex $(i,j)_R$. They are either left terminal $(i_1,j_1)_L$ or non-left-terminal $(i_1,j_1)_{nL}$ vertices as shown in Figure 4.11 according to the definition of out-neighbourhood for right-terminal vertices (Figure 4.6).

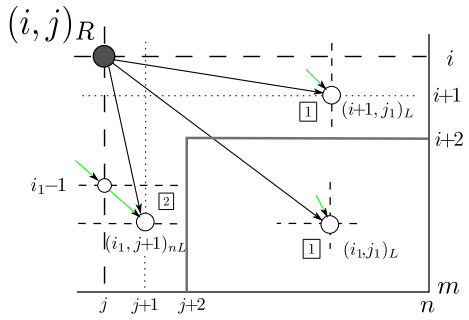


Figure 4.11: Two types of out-neighbours of $(i,j)_R$. For each type the elements with alternative lower contribution are shown in green (thin edges).

– For the first type, the function $T_{(i_1,j_1)_L}$ (Equation (4.4)) contains an alternative case $w_{\text{h\_}}(i_1,j_1)$, its highest value is -0.21 kcal/mol (Table 4.1). The case is shown in Figure 4.11 by the vertices $(i{+}1,j_1)_L$ and $(i_1,j_1)_L$. So, for such heads, the positive contribution $T_e\binom{(i,j)_R}{i_1,j_1}$ is not optimal.

– For the second type, the function $T_{(i_1,j_1)_{nL}}$ (Equation (4.5)) contains the alternative case $T_{i_1-1,j_1-1} + w_{\text{s}}(i_1,j_1)$. The highest value for the term T_{i_1-1,j_1-1} is achieved, when $(i_1{-}1,j_1{-}1)$ is a left terminal vertex, and equals $w_{\text{h\_}}(i_1{-}1,j_1{-}1)$. This case is shown by the vertex $(i_1,j{+}1)_{nL}$ in Figure 4.11. Otherwise, the term T_{i_1-1,j_1-1} contains weights of stackings besides that of the half-edge, and its value is even lower. According to the parameter database, the values of $w_{\text{h\_}}$ and w_{s} are always negative (Table 4.1). Hence, the non-left-terminal heads $(i_1,j_1)_{nL}$ feature a negative alternative case.

Thus, for each head (i_1, j_1) of an edge leaving $(i,j)_R$ the function T_{i_1,j_1} contains an alternative case with a negative value and the respective positive contribution $T_e(e)$ is not optimal for the head.

At last, consider the exception edge $e = \binom{i,j}{i+2,j+2}$ representing symmetric loop with two unpaired bases. The condition of the proposition ($C > 0$) does not ensure the positive contribution $T_e(e)$ for such edges, since

$$T_e\binom{i,j}{i+2,j+2} = \underbrace{C}_{>0} + \underbrace{\triangle G_{\text{loop}}(l)}_{=0} + \underbrace{E_{\text{asym}}(s)}_{=0} + \underbrace{\triangle G_{\text{leftMM}}(b_{j_1}, a_{i_1})}_{\geqslant -1.11}.$$

In this case, to have a positive contribution from the edge $\binom{i,j}{i+2,j+2}$ to $T_{i+2,j+2}$, the relation $C > -G_{\text{leftMM}}(b_{j_1}, a_{i_1})$ should hold. According to the ranges in the above expression, this is not always the case. Hence, the out-neighbour $(i{+}2, j{+}2)$ constitutes an exception. ■

Consequence of Proposition 4.2.1 : The condition of the proposition can be proved for a vertex without computation of the weights of its leaving edges. For a vertex complying with the proposition, its leaving edges exhibit a non-optimal contribution $T_e(e)$ and are detectable in advance. Hence, they are inefficient according to definition.

Filling of the matrix

The forward iteration follows in general the same flow as the exhaustive search (Figure 4.7). That is, it traverses the in-neighbourhood for each vertex (i_1, j_1). However, the number of the traversed in-neighbours is reduced – the vertices complying with the condition of Proposition 4.2.1 are excluded. For this, for each vertex $(i,j)_R$, after finding of its optimal score $T_{i,j}$, the condition term of the proposition is calculated. If it is positive, this vertex is marked and skipped by further calculations. The exceptional edge, defined in the proposition, is handled separately. In this manner, all inefficient edges starting at such vertices are removed from the hybridization graph.

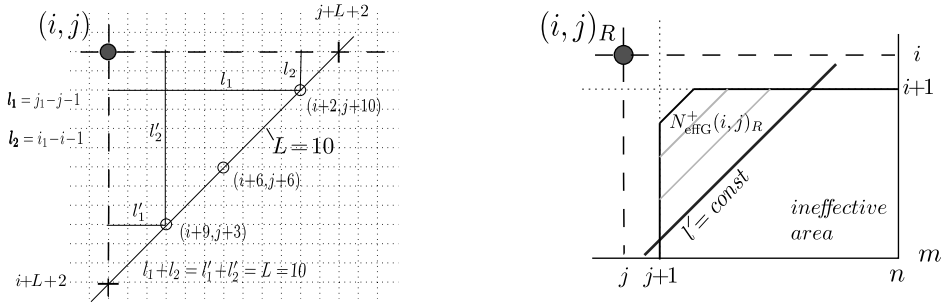
The method allows to prune seven vertices for the graph in Figure 4.4, they are shaded in the Figure 4.10. As can be seen, this method trims mostly vertices that are left and right terminal simultaneously. The resulting spanning subgraph contains 304 edges out of 378 of the initial hybridization graph. The systematic analysis of the performance is given in Section 4.3.

4.2.4 Edge Pruning

This method detects the inefficient leaving edges based on their length for all vertices in a given graph.
For this approach, the basic reasoning is the following. Consider a vertex (i,j) and its leaving edges $e = \binom{i,j}{i_1,j_1}$. For the edges representing loops, a penalty

$\triangle G_{\text{loop}}(l)$ outweights at certain length $l = L$ the corresponding score $T_{i,j}$, and the contribution $T_e(e)$ becomes positive and hardly optimal at the respective head (i_1, j_1). Consequently, the contributions of the edges representing even longer loops with length $l > L$ would be also not optimal, since $\triangle G_{\text{loop}}(l)$ is a non-decreasing function of loop length l (Equation (2.6)). The heads of edges corresponding to the loops of length L are placed in the grid on the line passing through nodes $(i, j{+}L{+}2)$ and $(i{+}L{+}2, j)$ (Figure 4.12a). This allows division of the out-neighbourhood area of the vertex (i, j) into two parts (Figure 4.12b):

- effective area $N^+_{\text{effG}}(i, j)$ contains vertices (i_1, j_1) connected to (i, j) by edges with length at most L bases, for which the contribution $T_e\binom{i,j}{i_1,j_1}$ is still negative, so its optimality at the respective heads is uncertain.
- ineffective area contains the out-neighbours (i_1, j_1) connected to (i, j) by edges with the length greater than L; their contributions are positive.



(a) constant loop length

(b) division of out-neighbourhood area for $(i, j)_R$

Figure 4.12: The effective out-neighbourhood for $(i, j)_R$. For non-right-terminal vertex $(i, j)_{nR}$ the effective area is restricted to the segments of the row $i{+}1$ and of the column $j{+}1$ from the vertex $(i{+}1, j{+}1)$ and up to the respective intersection points with the line L.
(a) All edges representing loops with $L = 10$ unpaired bases terminate at vertices belonging to the shown line.
(b) Effective area is restricted by the line $l = L$. The heads of the longer edges lie in the ineffective area.

The important issue is that these two areas exhibit a distinct border. The criterion allowing to separate these areas in out-neighbourhood of (i, j) is stated by Proposition 4.2.2, that also shows that the edges terminating in the ineffective area are inefficient. Thus, deletion of such edges results in a consistent subgraph. Figure 4.12b shows an approximated representation of the effective area, regarding only the number of unpaired bases in corresponding loops. Actually, the area is not exactly trapezoidal, since the weight of the loop comprises several other factors. The detailed review of this subject is given later by the description of the matrix filling step.

As can be seen, the proposed method implies the traversal of the out-neighbourhood for each vertex unlike the methods introduced previously. Figure 4.13 shows the corresponding general scheme for the matrix filling step.

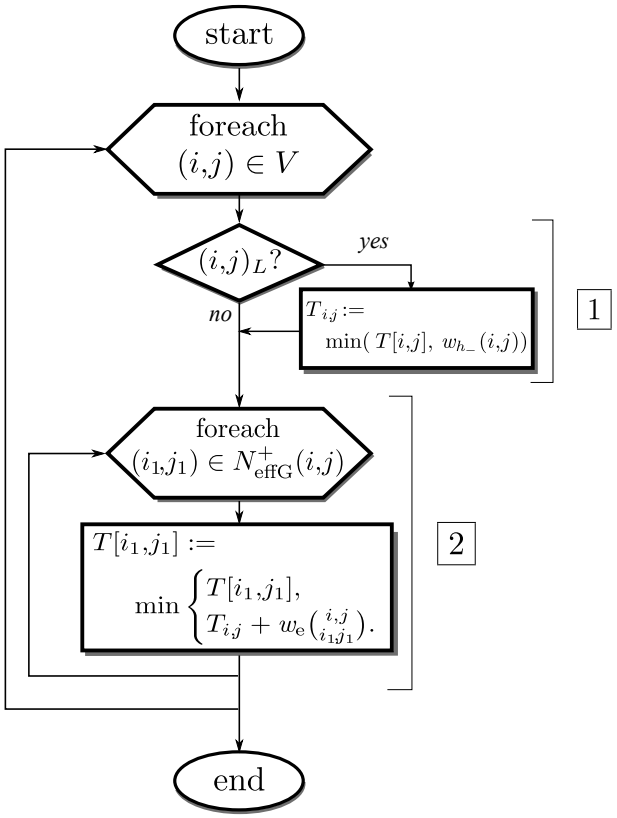


Figure 4.13: Flow chart for filling the DP matrix of the edge pruning method. Conditional block 1 takes into account the weight of entering half-edges for the left terminal vertices. Block 2 shows the traversal of the effective out-neighbourhood for every vertex (i, j); the scores for respective direct successors (i_1, j_1) are updated. The terms $T[i, j]$ with the indices in square brackets designate intermediate values at corresponding vertices, to distinguish them from $T_{i,j}$ denoting the optimal score.

The implementation of this scheme requires certain modification of the traditional calculation flow applied by the dynamic programming algorithms. Moreover, for the traversal of the effective out-neighbourhood a more subtle approach is preferable than straightforward considering of every vertex in the triangle in Figure 4.12b. These aspects and the respectively developed solutions are detailed by the description of the matrix filling step in the last part of this section.

Pruning criterion

For the edge $e = \binom{i,j}{i_1,j_1}$ its contribution can be estimated by the following expression:

$$T_{\text{est}}(e) = T_{i,j} + w_{\text{est}}\binom{i,j}{i_1,j_1}, \tag{4.8}$$

where w_{est} is the estimation of a lowest weight of the edge e. It depends on the represented structural motif:

$$w_{\text{est}}\binom{i,j}{i_1,j_1} = \begin{cases} w_{\text{bulge}}\binom{i,j}{i_1,j_1}, & \text{if the edge } \binom{i,j}{i_1,j_1} \text{ represents a bulge,} \\ w_{\text{estLoop}}\binom{i,j}{i_1,j_1}, & \text{if } \binom{i,j}{i_1,j_1} \text{ represents a sym./asym. loop.} \end{cases} \tag{4.9}$$

The lowest weight of the loop w_{estLoop} is estimated as follows:

$$w_{\text{estLoop}}\binom{i,j}{i_1,j_1} = \triangle G_{\text{rightMM}}(b_j, a_i) + E_{\text{loop}}(l, s) + \min(\triangle G_{\text{intMM}}), \tag{4.10}$$

where $E_{\text{loop}}(l, s)$ is a composite term combining contributions from loop length l and its asymmetry s, and $\min(\triangle G_{\text{intMM}})$ is the minimal value of the free energy of internal mismatch in the thermodynamic database. Based on the Equations (4.8) – (4.10), the following assertion gives a criterion for the separation of the effective area.

Proposition 4.2.2

If for some edge $e = \binom{i,j}{i_1,j_1}$ representing a loop the estimated lowest contribution $T_{\text{est}}(e)$, given by Equation (4.8), is positive, then its actual contribution $T_e(e)$ is not optimal at its head.

Proof: The edge $e = \binom{i,j}{i_1,j_1}$ represents either a bulge or symmetric/asymmetric loop. The following reasoning shows for each of these types that:

1. Equation (4.8) gives the lowest possible value for the corresponding contribution, that is, $T_e(e) \geqslant T_{\text{est}}(e)$ for both types of edges;
2. the function T_{i_1,j_1} at the head of an edge contains an alternative case with the negative value.

The presence of negative alternative case for T_{i_1,j_1} while the value of $T_e(e)$ is positive would prove that the latter is not optimal for the head (i_1, j_1).

1. Consider the edges according to their types.
   – If the edge $e = \binom{i,j}{i_1,j_1}$ represents a bulge, then $T_{\text{est}}(e) = T_{i,j} + w_{\text{bulge}}\binom{i,j}{i_1,j_1}$, according to Equation (4.9). Since $w_e\binom{i,j}{i_1,j_1} = w_{\text{bulge}}\binom{i,j}{i_1,j_1}$, the estimated contribution of the edge equals its real contribution. This means, if $T_{est}(e)$ is positive, so is $T_e(e)$.
   – If the edge $e = \binom{i,j}{i_1,j_1}$ represents a symmetric or asymmetric loop, then its weight equals $w_{\text{loop}}\binom{i,j}{i_1,j_1}$, and the corresponding contribution $T_e(e)$ is:

$$\begin{aligned} T(e) &= T_{i,j} + w_{\text{loop}}(L^{i,j}_{i_1,j_1}) \\ &= T_{i,j} + \triangle G_{\text{rightMM}}(b_j, a_i) + E_{\text{loop}}(l, s) + \triangle G_{\text{leftMM}}(b_{j_1}, a_{i_1}), \end{aligned}$$

where $E_{\text{loop}}(l,s)$ is a composite term combining penalties from the loop length and its asymmetry.
The estimated minimal loop weight given by Equation (4.10) is:

$$w_{\text{estLoop}}\left(\begin{smallmatrix} i,j \\ i_1,j_1 \end{smallmatrix}\right) = \triangle G_{\text{rightMM}}(b_j, a_i) + E_{\text{loop}}(l,s) + \min(\triangle G_{\text{MM}}).$$

Since $\triangle G_{\text{leftMM}}(b_{j_1}, a_{i_1}) \geqslant \min(\triangle G_{\text{MM}})$, the actual contribution $T_e(e)$ is greater or equals the estimation and is positive by hypothesis: $T_e(e) \geqslant T_{\text{est}}(e) > 0$.

2. Consider the head (i_1, j_1) of the edge $\left(\begin{smallmatrix} i,j \\ i_1,j_1 \end{smallmatrix}\right)$. The edge representing bulge can terminate at a left terminal $(i_1,j_1)_L$ or non-left-terminal $(i_1,j_1)_{nL}$ vertex. The edge representing other type of loop terminates only at the left terminal vertex $(i_1,j_1)_L$ (Figure 4.11). The existence of the alternative case with a negative value in function T_{i_1,j_1} has been shown by the proof of Proposition 4.2.1 for both types of vertices:
– for $(i_1,j_1)_L$ the alternative case is $w_{h\_}(i_1,j_1)$,
– for $(i_1,j_1)_{nL}$ it equals $T_{i_1-1,j_1-1} + w_s(i_1,j_1)$.

Thus, all edges $e = \left(\begin{smallmatrix} i,j \\ i_1,j_1 \end{smallmatrix}\right)$ that abide the proposition exhibit positive contribution $T_e(e)$, which are not optimal for their respective heads (i_1,j_1), since the corresponding expression T_{i_1,j_1} of the latter contains an alternative case with the negative value. ■

Consequence of Proposition 4.2.2 : For each vertex (i,j) the border of the effective area can be found by solving the equation: $T_{\text{est}}(e) = T_{i,j} + w_{\text{est}}\left(\begin{smallmatrix} i,j \\ i_1,j_1 \end{smallmatrix}\right) = 0$. Consider the term $w_{\text{est}}\left(\begin{smallmatrix} i,j \\ i_1,j_1 \end{smallmatrix}\right)$. It depends on the loop penalty, and its other factors are constant for the given vertex (i,j). Since the loop penalty is an increasing function of the loop length, the contributions of the longer edges with the heads beyond this border are positive and not optimal according to the proposition. Hence, all such edges are inefficient.

Filling of the matrix

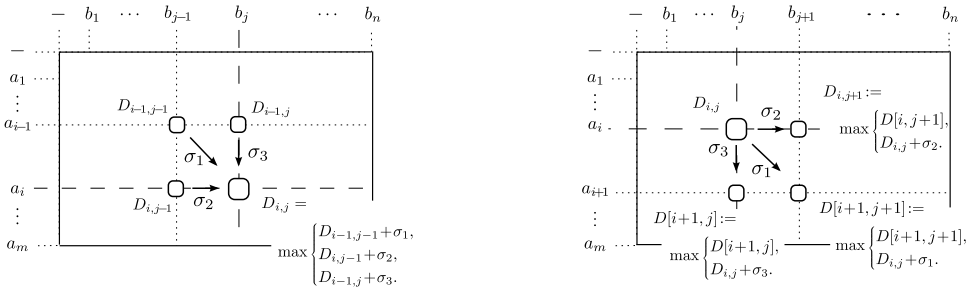
The two practical tasks by implementation of the edge pruning method are:
– traversal of the out-neighbourhood for each vertex instead of its in-neighbourhood,
– separation of the effective area based on the condition described previously.

To handle the first task, it is sufficient to change the typical succession of elementary calculations by filling of the DP matrix. Usually, the matrix filling step of DP algorithms is organized as nested iteration with two levels:

– A global cycle prescribes the order of proceeding from one cell to another in the matrix. It is usually started at the second row and finished at the row m, the cells are processed from the cell $[i,2]$ to $[i,n]$.

– Each local iteration actually finds the optimal score at the cell $[i, j]$ currently selected by the global cycle according to iterative formula $D_{i,j} = \text{opt}\{f(D_{i',j'})\}$, where opt() is either minimum or maximum and ranges over all predecessors $[i', j']$ defined by particular algorithm for the current cell $[i, j]$.

This computation flow is demonstrated in Figure 4.14a by example of the Needleman-Wunsch algorithm. Figure 4.14b shows the required alternative flow that considers successors in the local cycle. Main precondition for this is that the current cell $[i, j]$ contains its optimal score at the time of performing the local cycle for it.



(a) conventional local cycle (b) alternative local cycle

Figure 4.14: Two schemes of computation for the dynamic programming approach. The formulae are shown according to the Needleman-Wunsch algorithm for sequence alignment. The symbol "–" denotes a gap (extension to the accepted alphabet for alignment algorithms). Each figure shows a local cycle of computations for a current cell $[i, j]$ marked by a bigger square. Initial state for the cycle: the cells in the first $i-1$ rows and the first $j-1$ cells of the row i contain optimal scores. For the alternative cycle (b), the cell $[i, j]$ also contains its optimal score at the initial state.
(a): Result: optimal score at the cell $[i, j]$.
(b): Result: updated intermediate values at the cells $[i+1, j]$ and $[i+1, j+1]$, optimal score at $[i, j+1]$. The notation $D[i, j]$ with indices in square brackets designates intermediate values in the respective cells of the DP matrix to distinguish them from the optimal score denoted by subscripted form $D_{i,j}$.

The implementation of the alternative flow implies the following technical changes that do not influence the total result of calculations:

– The initialization of the matrix is the same as for the conventional DP algorithm.

– The global iteration starts at the first row and traces the cells from $[i, 1]$ instead of $[i, 2]$ and up to $[i, n-1]$ instead of $[i, n]$. Consider that after initialization the cells in the first row and column contain their optimal values. Then, by starting the global iteration from these cells the main precondition is ensured for the whole matrix.

– The local cycle updates intermediate value for each successor $[i_1, j_1]$ of the current cell $[i, j]$ using already found optimal value $D_{i,j}$ by formula $D[i_1, j_1] = \mathrm{opt}(D[i_1, j_1],\ f(D_{i,j}))$.

The resulting optimal values in all the cells after the filling of the DP matrix are the same as by conventional DP algorithm. For the alignment algorithms the both computation orders are equivalent. For the edge pruning method, in contrary, the alternative order is essential, since only in this way the reduction of the out-neighbourhood can be implemented.

The second issue is the implementation of the local cycle so that it considers only the out-neighbours in the effective area $N^{+}_{\mathrm{effG}}(i, j)$ for a vertex (i, j).
The trapezoid shown in Figure 4.12b is an overestimated approximation for such area, based on loop length penalty $\triangle G_{\mathrm{loop}}(l)$. The actual area is defined by the criteria of Proposition 4.2.2. Hence, they are used as cut-off conditions to exclude the out-neighbours, which belong to the ineffective area. These criteria depend on the loop type, thus, two separate Procedures 2 and 3 were developed to traverse the out-neighbours in the effective area connected to (i, j) by edges representing bulges and loops of the other types, respectively.

Procedure 2 Calculation of bulges from vertex (i, j) with heads in row $i{+}1$

Ranges: $j{+}1 < j_1 \leqslant n$ and $(i{+}1, j_1) \in V$
while $T_e(e) = T_{i,j} + w_{\mathrm{bulge}}\big(^{i,j}_{i+1,j_1}\big) \ < \ 0$ **do**
  / Update weight score at vertex $(i{+}1, j_1)$ */*
  $T[i{+}1, j_1] = \min\big(T[i{+}1, j_1],\ T_e(e)\big)$
  take next vertex $(i{+}1, j_1)$ *in row* $i{+}1$
end while

where
$T_e(e)$ – contribution of the edge $e\big(^{i,j}_{i+1,j_1}\big)$ to the function at its head $(i{+}1, j_1)$,
$T[i{+}1, j_1]$ – current (intermediate) value at $(i{+}1, j_1)$.

The traversal of the out-neighbours connected by bulges is implemented by considering the vertices in the row $i{+}1$ and column $j{+}1$ in assending order, and employing the criteria $T_e\big(^{i,j}_{i+1,j_1}\big) > 0$ as a cut-off condition. Procedure 2 shows the calculation for all out-neighbours $(i{+}1, j_1)$ of the vertex (i, j) in the effective segment of the row $i{+}1$. The computations for the effective segment of the column $j{+}1$ is similar. All the vertices in the effective segment are traced consequently starting from the left most one, while the respective contributions $T_e\big(^{i,j}_{i+1,j_1}\big)$ remain negative. According to the consequence of Proposition 4.2.2, the rest segment of the row belongs to the ineffective area.

To find all the out-neighbours in the effective area of (i, j) connected by edges representing symmetric/asymmetric loops, a special traversal order is required.

Consider the cut-off condition for these loop-edges: $T_{i,j} + w_{\text{estLoop}} > 0$. After substitution of w_{estLoop} according to Equation (4.10) it has the following form:

$$T_{i,j} + \triangle G_{\text{rightMM}}(b_j, a_i) + E_{\text{loop}}(l, s) + \min(\triangle G_{\text{intMM}}) > 0.$$

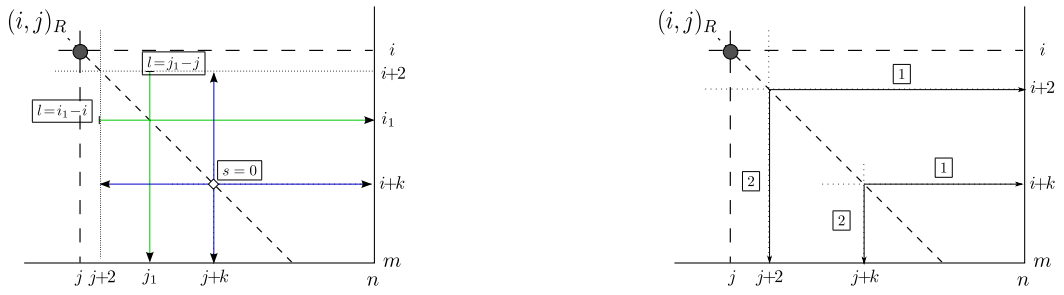
It is defined by loop penalty $E_{\text{loop}}(l, s) = \triangle G_{\text{loop}}(l) + \triangle E_{\text{asym}}(s)$, where l is loop length and s is its asymmetry, since all other terms are constant for a given vertex (i, j). To avoid the loss of information by applying of the cut-off condition, the traversal of the vertices should proceed so that both arguments of E_{loop} increase simultaneously. For two given vertices they are calculated as follows:

$$l = (i_1 - i - 1) + (j_1 - j - 1) = i_1 + j_1 - (i + j) - 2;$$
$$s = |(i_1 - i - 1) - (j_1 - j - 1)| = |(i_1 - j_1) - (i - j)|.$$

The nodes in the grid are usually traced row- or column-wise. The corresponding regions of increase for l and s each are given in Figure 4.15a. The resulting regions of their simultaneous increase are shown in Figure 4.15b:

1. for a fixed row $i_1 = i+k$, the interval is $j_1 \in [j+i_1-i, n] = [j+k, n]$, with $2 \leqslant k < \min(n-j, m-i)$ (segments 1 in Figure 4.15b);
2. for a fixed column $j_1 = j+k$, the interval is $i_1 \in [i+j_1-j, m] = [i+k, m]$ (segments 2 in Figure 4.15b).

Thus, $w_{\text{estLoop}}\binom{i,j}{i_1,j_1}$ is also non-decreasing along these intervals.



(a) regions of increase for l and s

(b) superimposition

Figure 4.15: Regions of increase of the loop length and asymmetry for right terminal vertex $(i, j)_R$. Arrows denote the direction of increase. Short-dashed line represents diagonal of $(i, j)_R$.
(a) The region for l is in green, for s – in blue. Asymmetry has zero point at each node $(i+k, j+k)$, for $2 \leqslant k < \min(n-j, m-i)$, on diagonal of $(i, j)_R$ and increases in four directions along the corresponding row and column.
(b) Segments of the simultaneous increase of the loop length and asymmetry.

Both intervals meet pairwise for the same k at the point $(i+k, j+k)$. Hence, by consecutive pairwise traversal of these intervals for the whole range of k, the

out-neighbourhood $N_G^+(i,j)_R$ in the grid is covered completely.
Application of the cut-off conditions excludes the head vertices in the ineffective area, and so reduces the traversed area to the effective one $N_{\text{effG}}^+(i,j)_R$.

Procedure 3 Traversal of out-neighbours in effective area of the vertex (i,j) connected by loop-edges

$k=2$ increment counter

Ranges: $1 \leqslant i \leqslant m-2,\ 1 \leqslant j \leqslant n-2,\ i+k \leqslant i_1 \leqslant m,\ j+k \leqslant j_1 \leqslant n$

while $T_{\text{est}}\binom{i,j}{i+k,j+k} = T_{i,j} + w_{\text{estLoop}}\big(L^{i,j}_{i+k,j+k}\big) < 0$ **do**

/ Update score for vertices in row $i+k$ right of or at the diagonal of (i,j) */*
take first left terminal vertex $(i+k, j_1)_L$, *s.t.* $j_1 \geqslant j+k$
while $T_{\text{est}}\binom{i,j}{i+k,j_1} = T_{i,j} + w_{\text{estLoop}}\big(L^{i,j}_{i+k,j_1}\big) < 0$ ***do***
$T_e(e) = T_{i,j} + w_{\text{loop}}\big(L^{i,j}_{i+k,j_1}\big)$
$T[i+k, j_1] = \min\big(T[i+k, j_1], T_e(e)\big)$
take next vertex $(i+k, j_1)_L$ *in row* $i+k$
end while

/ Update score for vertices in column $j+k$ below the diagonal of (i,j) */*
take first left terminal vertex $(i_1, j+k)_L$, *s.t.* $i_1 > i+k$
while $T_{\text{est}}\binom{i,j}{i_1,j+k} = T_{i,j} + w_{\text{estLoop}}\big(L^{i,j}_{i_1,j+k}\big) < 0$ ***do***
$T_e(e) = T_{i,j} + w_{\text{loop}}\big(L^{i,j}_{i_1,j+k}\big)$
$T[i_1, j+k] = \min\big(T[i_1, j+k],\ T_e(e)\big)$
take next vertex $(i_1, j+k)_L$ *in column* $j+k$
end while

k++ // proceed to the next pair of column and row
end while

where
$T_{\text{est}}\binom{i,j}{i_1,j_1}$ – estimated contribution of the edge $\binom{i,j}{i_1,j_1}$ to the function at its head (i_1, j_1),
$T_e(e)$ – actual contribution of the edge $e = \binom{i,j}{i_1,j_1}$,
$T[i_1, j_1]$ – current (intermediate) value at vertex (i_1, j_1).

According to the different out-neighbourhood of the vertices, there are again two distinct procedures respective to the vertex types. For the right terminal vertices, both above described procedures are applied; for the non-right-terminal vertices only Procedure 2 applies.

The edge pruning method reduces the example hybridization graph in Figure 4.4 to 86 edges from the initial 378. The systematic comparison of its performance with that of both previously introduced methods is given in the next section.

4.3 Performance and Implementation

In the first part of this section, the performance of the three described methods for finding the shortest path in the hybridization graph is compared. The advantages compared to the modern RNA methods that follow the same hybridization model are also shown. The second part presents some practical aspects of implementing the proposed algorithms. For this, the key points are storage organization and filling order of the dynamic programming matrix.

All three methods presented were implemented in `C++` and compiled under Linux SUSE v.10.0 with the GNU compiler version 4.2. The software uses only the standard `C++` library STL and thus should also run on other operating systems supporting this programming language.

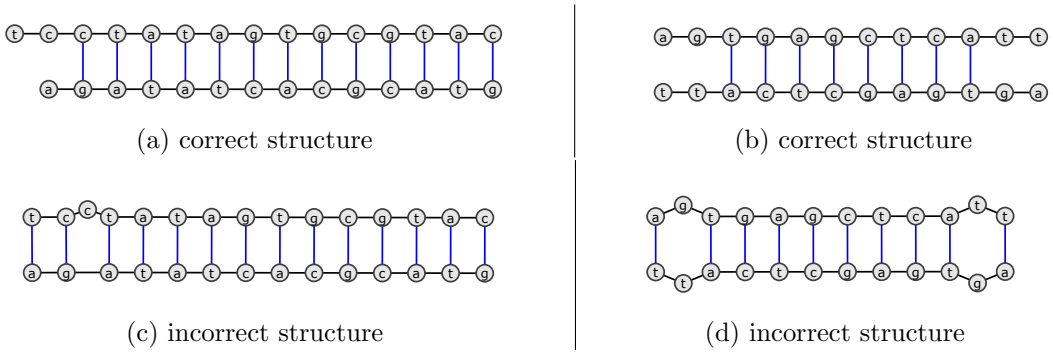
4.3.1 Performance

Benchmark tests

As a benchmark, the proposed methods were applied to predict the MFE structures of 195 DNA/DNA complexes used in the works of Allawi and SantaLucia [3, 4, 5, 6] to establish appropriate thermodynamic parameters. The lengths of the DNA sequences in the set ranged from 9 to 16 bases. These lengths are generally appropriate for DNA strand design in DNA computing [50]. This set of DNA/DNA complexes includes the hybridization complexes with single and double internal loops with a length of two bases and also single symmetric loops with a length of four bases. The predicted energies and secondary structures completely agree with those of the sources.

Particular complexes constituting a challenge for prediction [59, 84] are shown in Figures 4.16a and 4.16b. The hybridization complex in Figure 4.16a is a break point for the greedy method of Kaderali and Schliep [54] that finds a suboptimal structure depicted in Figure 4.16c. The second challenging complex is taken from SantaLucia and Hicks [84]. Figures 4.16b and 4.16d show the correct and false predicted structures, respectively, for this complex. The critical point by both pairings is to find the terminal motifs correctly. The methods of this thesis predict the correct structures for both cases.

Another set of DNA/DNA complexes was developed based on the Random Sequence Generator [99]. The predictions provided by algorithms of this thesis were compared with those made by the PairFold and DINAMelt server [67] that adopted a DNA version of the Zuker and Stiegler's RNA folding algorithm [112]. All three algorithms predicted the same structures, while the total minimum energy scores of the structures differed slightly due to different sets of thermodynamic parameters and counting the terminal effects (Section 2.1.2).



(a) correct structure (b) correct structure

(c) incorrect structure (d) incorrect structure

Figure 4.16: Challenging cases for MFE prediction from Leber et al. [59] (a) and SantaLucia and Hicks [84] (b).
(a) Correct structure of the complex; $E_{\text{total}} =$ -12.59 kcal/mol.
(c) Best structure found by greedy method [54] for sequences at the Figure (a); $E_{\text{total}} =$ -10.7 kcal/mol.
(b) Correct structure with free energy $E_{\text{total}} =$ -7.77 kcal/mol.
(d) Incorrect structure with penultimate mismatches (two-bases loops) for sequences in Figure (b); $E_{\text{total}} =$ -6.26 kcal/mol.

Average performance

Dynamic pruning allows the reduction of the number of edges considered in the hybridization graph during calculations, thus truncating unfeasible edges and path segments. The average improvement gained by both pruning methods is shown in Table 4.2.

Table 4.2: Average size of random hybridization graph and its subgraphs reduced by two pruning methods.

Method	*length of DNA sequences (nt)*			
	30	100	200	300
	smallest and largest graph size (edges)			
	$\times 10^2$	$\times 10^4$	$\times 10^4$	$\times 10^4$
exhaustive search	23 – 103	77 – 101	1306 – 1493	6761 – 7441
vertex pruning	8 – 87	30 – 86	556 – 1309	2890 – 6894
edge pruning	0.4 – 30	0.1 – 16	0.8 – 287	6 – 1430
	smallest and largest subgraph size (%)			
vertex pruning	35 – 85	39 – 85	43 – 88	43 – 92
edge pruning	1.7 – 29	0.13 – 16	0.06 – 19	0.09 – 19

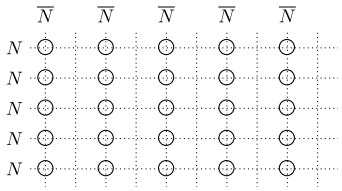
To quantify the effect of pruning techniques, four datasets of 50 sequences each were generated with Random DNA Sequence Generator [99]. The length of

sequences in the sets are 30, 100, 200, and 300 nucleotides. The sequences in each set were hybridized all-against-all. As a measure for the effectiveness of the three methods, the size of the graphs, i.e., the number of edges, is taken. The number of edges considered by exhaustive search corresponds to the size of hybridization graph; for the pruning methods, the sizes of the spanning subgraphs are given.

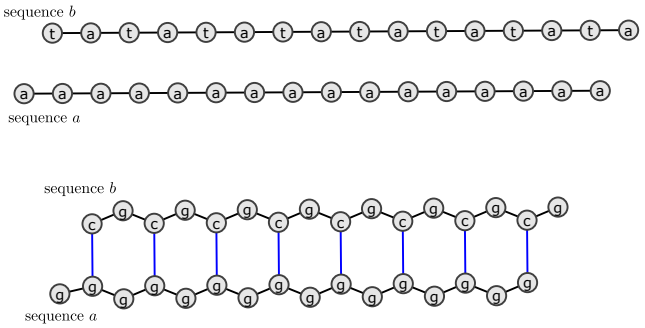
The upper half of Table 4.2 demonstrates minimal and maximal sizes of the hybridization graphs, produced by pairs of random sequences. The lower half shows the corresponding amount of reduction achieved by the pruning techniques.
As can be seen from the first row of the table, graph size grows rapidly with the length of input sequences. This makes the exhaustive search, though computationally manageable, impractical for large-scale applications. The most effective method in the case of a random graph is edge pruning.

Worst case performance

The worst case of the hybridization graph is described in Section 4.2.1. To assess the performance of the pruning methods in this case, a set of pairs of DNA sequences was developed with the length of 30 nt. Each pair in the set obeys the scheme given in Section 4.2.1; namely, $a = (N)_{30}$, even positions $b_{2k} \neq \overline{N}$, and odd ones $b_{2k-1} = \overline{N}$, with $1 \leqslant k \leqslant 15$. Figure 4.17a shows the vertex layout corresponding to such base composition of the input sequences. For each hybridization graph, defined by a pair of sequences in the test set, the size of subgraphs reduced by both pruning methods was found.



(a) worst vertex layout



(b) typical MFE structures

Figure 4.17: Vertex layout for the worst case hybridization graphs and representative MFE structures resulting from corresponding inputs.

Representative pairs of DNA sequences with free energy of their complexes and size of spanning subgraphs relative to the full one are shown in Table 4.3. These pairs are separated into three groups according to the base composition of the strands; each of them exhibits a certain type of MFE structure given in Figure 4.17b.

The first group, given by the first two lines of the Table 4.3, comprises strand pairs, where a is a repetition of base A, C or T, and b is regular, that is, b_{2k} are all the same base. Such strands do not build a bimolecular complex (the two upper separate strands in Figure 4.17b), since their MFE is positive.

Table 4.3: Sequences producing worst case hybridization graph

	Sequences $b(5'-3')$; $a(3'-5')$	stability E_{total} (kcal/mol)	size of subgraph (%) pruning1	pruning2
1	$b=(TA)_{15}$ [a]; $a=(A)_{30}$	positive	6.67	0
2	$b=(AT)_{15}$; $a=(T)_{30}$	positive	19.11	0.03
3	$b=(CT)_{15}$; $a=(G)_{30}$	-2.36	100	2.36
4	$b=(CA)_{15}$; $a=(G)_{30}$	-6.16	100	6.8
5	$b=(CG)_{15}$; $a=(G)_{30}$	-15.57	100	25.23
6	b=cactcgcacgctcgcacgctcacgctcgca; $a=(G)_{30}$	-9.54	100	12.8

[a] – $(TA)_{15}$ is a 30-mer, consisting of alternating T and A bases; the same notation is used for the other sequences.

The second group corresponds to pairs with sequence a, consisting of G-s, and a regular b sequence (rows $3-5$ in the table). These pairs have negative free energy, so they build a DNA/DNA complex similar to the second complex in Figure 4.17b. This complex is built by sequences in the 5th row and represents the most stable case.

The third group contains b-sequences with different nucleotides at the even positions, e.g., the 6th pair in the table. In this group the complexes are also more stable if $a=(G)_{30}$ and their structure is similar to the second complex in Figure 4.17b. For differently composed sequence a the stands do not hybridize.

Both pruning methods show a high level of graph reduction for the first group of sequence pairs. For the second group and stable hybridizations in the third one, the vertex pruning method is completely ineffective, since it computes the whole graph. The edge pruning shows the graph size reduction of at least 75 %; that is comparable to its performance for random sequences of the same length (Table 4.2). The edge pruning method produces the largest spanning subgraph for the fifth pair of sequences in the table. That pair builds the most stable hybridization complex for the worst case hybridization graph.

Figure 4.18 shows the real size of the hybridization graph and corresponding spanning subgraphs for the sequences in Table 4.3. Both pruning methods exhibit an increase in the subgraph size with increasing stability of DNA/DNA complex. The vertex pruning is ineffective for all sequence pairs with negative free energy. For some graphs in the first group, the edge pruning method computes only half-edges; in the diagram it is shown as zero edges in the third category for the

left-most group of bars.

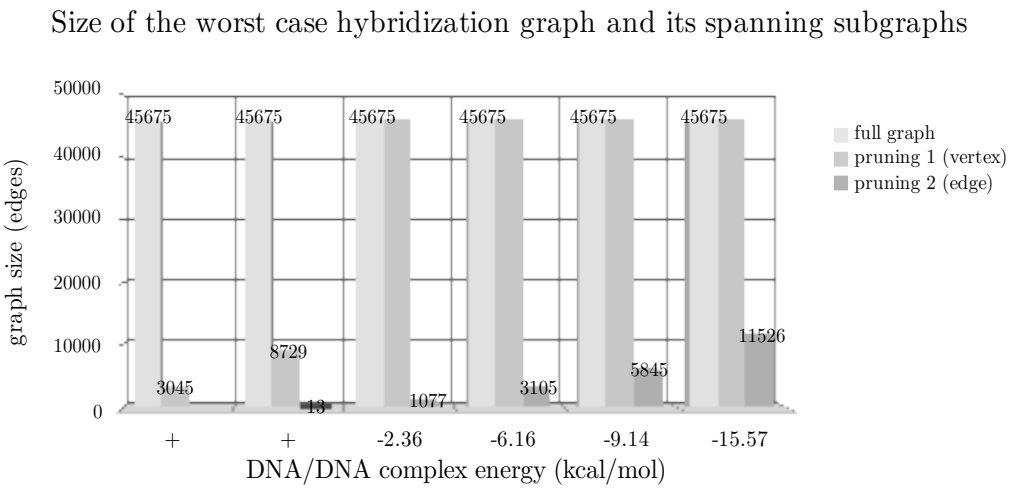


Figure 4.18: Number of observed edges for the worst case hybridization graph (category "full graph") and its subgraphs, reduced by vertex pruning (category "pruning 1") and edge pruning (category "pruning 2") methods for six pairs of input sequences from Table 4.3. The length of input DNA sequences is 30 nt.

The demonstrated results show that both pruning methods can considerably improve the performance of exhaustive search for the worst case hybridization graph. The vertex pruning method reduces the graph for pairs of sequences with positive hybridization energy. The edge pruning technique exhibits for the worst case graph the performance comparable to its results in a random case. This technique is the most effective for both random and worst case hybridization graphs.

Comparison with other tools

To asses the effectiveness of the developed methods relative to that of other modern ones, two applications were selected that exploit the same hybridization model, i.e., they also consider only intermolecular base pairings. Those are HYBRID from server DINAMelt [67] and RNAduplex from Vienna package [42]. They utilize solution strategies different from that presented in this work and were initially developed for RNA. With integration of DNA thermodynamic parameters they became applicable to the DNA hybridization problem. The edge pruning method was taken for the evaluation as the most effective one.
For this, the test sets are the same as for the comparison of graph pruning methods with the exhaustive search in an average case. There are four sets of DNA sequences (30, 100, 200, and 300 nt long) and an all-to-all pairing scheme inside each set. Due to basic differences between approaches, the best criterion for

comparing their performance is runtime. The runs for RNAduplex and the edge pruning method were performed on 4*Dual Core AMD Opteron 2.6 GHz computer. The runtimes for HYBRID are given as reported by server DINAMelt. The average runtimes are shown in Table 4.4.

Table 4.4: Comparison of different methods for intermolecular interaction. The average runtimes are as reported by the Unix *time* command on 4*Dual Core AMD Opteron.

Method	*length of DNA sequences (nt)*			
	30	100	200	300
	average runtime (ms)			
HYBRID (DINAMelt) [a]	11.5	57.2	218.8	480.0
RNAduplex (Vienna)	1.5	35.9	176.5	377.6
edge pruning	0.9	28.0	170.5	1203.2
speed-up [b]	40%	27%	4 %	-218%

[a] – time reported by server DINAMelt.
[b] – the speed-up of edge pruning is given related to RNAduplex performance.

The second pruning method achieves an average speed-up of 40% for sequences of length 30 nt and of 27% for sequences of length 100 nt, when compared with RNAduplex. Moreover, both these methods exhibit comparable performance for sequences of length 200 nt. For longer sequences, the hybridization graphs become more complex and the other approaches become more appropriate.

4.3.2 Implementation Features

For usability of any algorithm an important aspect is a practical implementation of computation flow and storage management. In the case of the dynamic programming approach it concerns memory allocation and filling order of the dynamic programming (DP) matrix. Zuker and Sankoff [111] explored the topics for RNA folding applications. This section investigates some special features of these issues for the hybridization graph. It also describes some methods for reducing the number of intermediate computer operations.

Properties of the grid

As the graph layout is defined by two DNA sequences, the corresponding grid exhibits the following properties:

1. The number of vertices in every row is lower than n.
   Every row i, defined by a nucleotide $a_i = N$, with $N \in \Sigma = \{A,C,G,T\}$, contains vertices at positions j, where $b_j = \overline{N}$ is complementary to N. The number of such complements is lower then n, since in most cases the sequence b contains bases of distinct types.

2. The positions of all vertices on the grid are defined by four arrays containing positions of the same bases in one of the input sequences, since the nucleotide alphabet has four letters and each base has a single complementary one.

Storage organization

The first matter is the organization of storage for the vertices of the hybridization graph. Based on the first property of the grid, the vertices can be stored in an array of m linear arrays of complementary positions, each of them shorter than n (Figure 4.19b).

(a) regular array (b) irregular storage

Figure 4.19: Possible storage organization of DP matrix for hybridization graph.

However, extra storage would be required for each vertex to hold its initial column position j on the grid. This option also takes additional time for allocation of such an irregular array for every pair of input sequences (Table 4.5).

Table 4.5: Comparison of initiation time for two types of data structures. Time is measured for initiation of 1000 data structures of $m = 300$, $n = 360$ units length in `C++`.

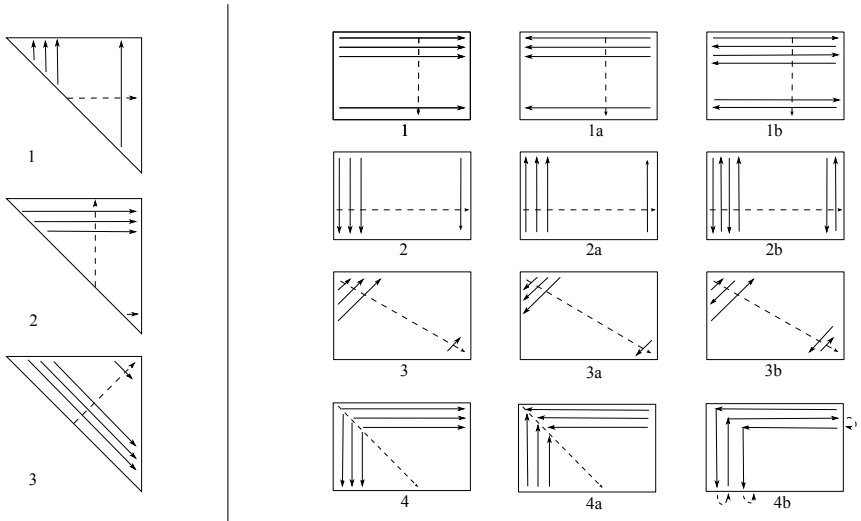
Data structure	initiation time (s)
$m \times n$ rectangular array (Figure 4.19a)	5,34
m arrays of 4 different lengths (Figure 4.19b)	10,60

In case of a single pairing of two sequences, the time is negligible. On the contrary, for large-scale applications, where the allocation is repeated thousands of times, it slows down the total performance significantly. Following this criterion, the grid is organized as regular two dimensional $m \times n$ array (Figure 4.19a).

DP matrix filling order

The specific design of the hybridization graph leads to some novel ways of organizing the global iteration over the DP matrix. These filling orders are distinct from that suitable for RNA folding or for common sequence alignment algorithms, such as ones from Smith and Waterman [88] or Needleman and Wunsch [74]. Zuker and Sankoff [111] described three possible filling orders for RNA folding,

where a triangular matrix is used (Figure 4.20a). They correspond up to the rectangularity of the matrix to the cases 1, 2, and 3 for DNA hybridization graph (Figure 4.20b). For the common sequence alignment, the schemes 1, 2, 3, 3a, 3b, and 4 in Figure 4.20b are appropriate. Thus, there are six new possibilities 1a, 1b, 2a, 2b, 4a, 4b (Figure 4.20b) for filling the DP matrix by computation of the hybridization graph.



(a) RNA folding (b) DNA hybridization graph

Figure 4.20: Possible filling orders of dynamic programming matrix. The solid arrows indicate the direction of consecutive computations. The dashed arrows indicate transition from one solid arrow to the next.
(a) Three filling orders for RNA triangular DP matrix (from Zuker and Sankoff [111]).
(b) Four main filling orders for ssDNA rectangular DP matrix and their variants.

The main restriction on the order for both RNA and DNA cases is that of dynamic programming: to find a score at some cell in the matrix, the scores of its direct predecessors, i.e., the cells it depends on, should already be computed. The main distinction of the methods that leads to the new filling orders is the following. For RNA folding and sequence alignment methods, a cell $[i, j]$ has direct predecessors in row i and column j. Therefore, the rows i can be filled only in ascending order of columns $1 \leqslant j \leqslant n$ (schemes 1 in Figure 4.20a and 4.20b). The ascending row order $1 \leqslant i \leqslant m$ by column j filling (schemes 2 in Figure 4.20a and Figure 4.20b) abides the same restriction. For the hybridization graph, on the contrary, the vertices are independent of those in the same row or column. This property allows the descending filling orders and the switching from one to another after transition to the next row or column. The descending filling flows as follows: for rows – from cell $[i, n]$ to $[i, 1]$; for columns – from cell (m, j) to $(1, j)$. These are depicted in the schemes 1a and 2a (Figure 4.20b). The

switching from ascending to descending flow is shown by the schemes 1b and 2b (Figure 4.20b).

The variants of diagonal filling shown in schemes 3a and 3b (Figure 4.20b) for sequence alignment and DNA hybridization methods arise due to the rectangularity of the matrix. Scheme 4 and its variants (Figure 4.20b) depict an option of segmental filling of rows and columns. A row i is filled in a segment from cell $[i, i]$ to $[i, n]$ and corresponding column $j = i$ – from $[i+1, j]$ to $[m, j]$. The transition is made consecutively to the next pair of segments. The sequence of segments, i.e., first the segment in row i, then the respective one in column $j = i$, or vice versa, remains the same by transition. This order is also appropriate for the sequence alignment methods. The scheme 4a differs only by descending filling of the segments: in row i from cell $[i, m]$ to $[i, i]$, and similarly for the segment in column $j = i$.
The scheme 4b demonstrates the option of switching between ascending and descending flows by filling the segments in row and column. For this, the segments in row i and column $j = i$ are filled in opposite orders. By transition from one row-column pair to another, the sequence of segments also changes. If at first round the sequence is row then column, then at the second one, the segment in column is filled first, then the one in the respective row.

All described filling orders are applicable to both exhaustive search and pruning methods. Although the new filling orders for the hybridization graph are possible in principle, they are impractical regarding the consecutive storage organization of modern computers. Therefore, current software implements simple row-wise filling order given by scheme 1a (Figure 4.20b).

Optimization of access to the vertices

The technique is based on the second property of the grid. To iterate over vertices, the positions of $\overline{a_i}$ in b were stored in eight index arrays: one for each base type A, C, T, G for both sequences. Following the row filling order, to address every vertex in a row its position is taken from the array of complements corresponding to nucleotide a_i. This provides consecutive access exactly to each vertex in a given row or column without considering every node of the grid.

Finding the lowest energy

A final step of the forward iteration consists of finding a right terminal vertex with the lowest weight after the filling of the DP matrix. Since such vertices are spread irregularly over the grid, it requires an additional traversal of the whole matrix. This can be avoided by retaining the actual lowest path weight separately and updating its value after the finding of the optimal score for the next right terminal vertex while filling the matrix. The update step considers

the corresponding half-edge leaving the right terminal vertex as follows:

$$W = \min\Big(W,\ T_{(i,j)_R} + w_{h_+}(i,j)_R\Big).$$

After the matrix filling, W contains the lowest path weight.

Exceptional cases of the graph

For practical reasons, the following two cases are handled separately, that is, the hybridization graph is not established for two input sequences a and b:

1. If the sequences are completely complementary, that is $a = \overline{b}$, the optimal structure of their complex per default is a single uninterrupted double helix; its energy $E_{\min} = \sum \triangle G_{\mathrm{NN}}$ is computed in a straightforward manner.
2. If one of the sequences does not contain any base complementary to the bases of the other sequence, that is, all a_i, b_j satisfy $a_i \neq \overline{b_j}$, the base pairing matrix is empty, and the sequences cannot hybridize with one another; their hybridization free energy is set to zero.

Practical limitation of implemented software

The actual implementation exploits the database of DNA thermodynamic parameters from SantaLucia and Hicks [84]. Currently it is the most complete one, containing the parameters for Watson-Crick nearest neighbours, single internal mismatches, loops and dangling ends. An alternative parameter set used by adapted RNA applications is that from Mathews [71].
The edge pruning method is independent of the particular parameters, so it is compatible with the other database.
The vertex pruning technique is based on the value ranges for mismatches (Section 4.2.3). Thus, its cut-off condition may require appropriate modification for exploitation with other mismatch parameters. However, the main principle of the technique holds.

According to Peyret [78], the exact thermodynamic parameters for terminal mismatches are still a research matter. Moreover, the up-to-date parameters for these motifs from SantaLucia [83] laboratory remain unpublished. Hence, for the current implementation the free energy values of the internal mismatches were also used for weighting the terminal mismatches. This constitutes a limitation for the application of the software in practice, but does not impact the essence of the developed methods.

4.4 Discussion

The introduced hybridization graph model for a DNA/DNA complex allows the application of the shortest path algorithms developed in graph theory. The basic methods for this are Floyd-Warshall [32, 101] and Johnson [52] algorithms.
The first method has complexity of $O(|V|^3)$. For input sequences with equal distribution of nucleotides, the number of vertices in hybridization graph amounts to $n^2/4$, where n is a length of the strand. Thus, the complexity of this algorithm for such graphs is $O(n^5)$. The complexity of the Johnson's method [52] is $O(|V|^2 \log |V| + |V||E|)$. In both cases, the value is higher than the worst case complexity of $O(n^4)$ or $O(|E|)$ (Section 4.2.1) for the exhaustive search method presented in this chapter.
The special features of the hybridization graph, such as mapping to the grid and absence of cycles, have yielded more effective methods than the general ones for solving the shortest path problem for this particular type of graphs.

The direct implementation of the dynamic approach demonstrated by exhaustive search is inefficient for long input sequences due to rapid increase of the graph size. Both pruning methods are designed to reduce the complexity of the direct approach without loss of the optimal solution. The comparison of the results of computations has shown complete agreement in found structures and energy with that found through exhaustive search. The edge pruning method exhibits better reduction of the hybridization graph for both average and worst case graph than the vertex pruning.

The basic programs for DNA/DNA hybridization, such as **Dan** [14], **Melting** [58], and **MeltTemp** [72], take into account either perfect duplexes or symmetric internal loops with a length of two bases. The proposed methods generalize these algorithms by including more complex DNA structural motifs.
An advanced MFE finding method developed by Leber et al. [59] requires two to four iterations over the whole dynamic programming matrix, while the algorithms described above need only one iteration over a reduced set of cells in the matrix. They also exhibit better reliability in finding the terminal motifs than the greedy method of Kaderali and Schliep [54] (Section 4.3).

The basic method for RNA folding applications, which were adapted for DNA/DNA hybridization, is the Zuker and Stiegler's algorithm [112]. Its complexity $O(n^4)$ is comparable to that of the exhaustive search method. Some refinements of the Zuker and Stiegler's algorithm reach a complexity of $O(n^3)$. For this, one of the ways is restriction of the maximal loop length to 30 nt, as by Hofacker et al. [49]. One of the advantages of the developed pruning methods is that loop length remains unrestricted, namely, it is bound only by energy. So, a loop of any length would not be excluded from a complex if it is energetically acceptable.

The main limitation on the application of the proposed algorithms comes from the hybridization model employed. This model is inappropriate for DNA/DNA complexes that form hairpins. For shorter DNA sequences up to 30 nt, DNA/DNA hybridization complexes rarely contain hairpins and multi-branched loops [97]. In this case, the proposed methods yield optimal solutions that correspond to biological structures. For longer sequences, the methods can be used as a subroutine to calculate optimal internal loops. However, they are less effective for RNA sequences, since the corresponding hybridization graphs eventually become much larger due to wobbling *G-U* pairings.

The main targeted application fields for the developed methods are DNA strand design software for DNA computing and PCR or microarray probe design. The edge pruning technique allows the acceleration of a number of large-scale software applications that take into account free energy of hybridization by generating of DNA sequence sets with the restrictions on strand pairings. Such applications depend on calculation of hybridization energy for pairs of ssDNA while searching for the best set of sequences. These calculations are considered as one of the main bottle necks in thermodynamic DNA strand design, as they take the most of time [98, 54]. The state of the art is codes with word length from 16 to 32 nt, which find further use in DNA-based computations. For the sequences of these lengths the restrictions of the hybridization model, such as exclusion of hairpins and multiloops, are biologically viable.
The runtime comparison of the proposed edge pruning method with light-weight RNA tools HYBRID and RNAduplex shows that its integration into large-scale applications for DNA strand design, like those provided by Mann and Noble [65] and Tulpan et al. [98], would significantly speed up their performance.

The edge pruning method proposed in this thesis improves existing DNA/DNA hybridization methods by handling longer loops (more than two bases) and loops of different types. As the density of vertices of the hybridization graph is generally smaller than the density of nodes in the grid, the algorithm attains lower average runtime than comparable RNA algorithms for sequences up to 200 nt. It accelerates the general performance with the same exactness of results.

Chapter 5

Tropical APSP Algorithm for Bipartite Graphs

Bipartite graphs are widely used for modeling of complex networks found in biology, technology, and computer science. Finding of the shortest path is one of the basic subroutines with a high number of executions in these applications. Hence, there is high demand for the optimization of these procedures. In Section 2.4, the general algorithms for finding the shortest path in all types of graphs are described. This chapter presents the special version of the tropical APSP algorithm for the specific case of bipartite graphs. It optimizes the general algorithm by taking into account the properties of this class of graphs. It halves the space complexity and reduces the runtime about tenfold compared to the general tropical algorithm.

5.1 Properties of Bipartite Graphs

A bipartite graph or *bigraph* $G = (V, E)$ with parts V_1 and V_2 can be depicted in the form shown in the Figure 5.1 by grouping together the vertices in each part.

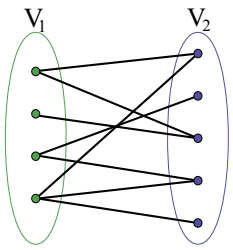


Figure 5.1: Example of a bipartite graph.

In the adjacency matrix of a bigraph, let the first $|V_1|$ rows and columns be labeled with the vertices of set V_1, and the rest with the vertices of set V_2. This

matrix can then be divided into the following four blocks:

$$A = \begin{pmatrix} 0 & A_1 \\ A_2 & 0 \end{pmatrix},$$

where A_1 is an $|V_1|\times|V_2|$ matrix and A_2 is an $|V_2|\times|V_1|$ matrix. The two 0-blocks represent square zero matrices, since the vertices in the same parts are not adjacent to each other. In the case of a complete bipartite graph, the blocks A_1 and A_2 contain only ones. For an undirected bigraph, $A_2 = A_1^T$; for a directed one, this equality does not hold.

The corresponding weight matrix M for a weighted bipartite graph exhibits a similar block structure:

$$M = \begin{pmatrix} U & M_1 \\ M_2 & U \end{pmatrix}.$$

The blocks M_1 and M_2 have the same dimensions as A_1 and A_2, respectively. Their entries are weights of the edges or ∞, if there is no edge between corresponding vertices. For undirected bigraphs, $M_1 = M_2^T$; for directed ones, this equality does not hold. The upper left and lower right square blocks U contain zeros on the main diagonal and all their other entries equal ∞, since the vertices in the same part are not adjacent to each other:

$$U = \begin{pmatrix} 0 & \infty & \cdots & \infty \\ \infty & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \infty \\ \infty & \cdots & \infty & 0 \end{pmatrix}.$$

Now, consider that the tropical APSP algorithm consists in exponentiation of the weight matrix of a graph to the power $|V|-1$ (Section 2.4.2). For each $n \in \mathbb{N}$, the exponentiated weight matrix $M^{\odot n}$ of the bigraph can be divided into four blocks with the same sizes as in the initial weight matrix M:

$$M^{\odot n} = \begin{pmatrix} L_1^{(n)} & M_1^{(n)} \\ M_2^{(n)} & L_2^{(n)} \end{pmatrix},$$

where $L_1^{(n)}$ and $L_2^{(n)}$ are the upper left and lower right blocks of $M^{\odot n}$, and $M_1^{(n)}$ and $M_2^{(n)}$ are the upper right and lower left blocks, respectively.

Due to the special structure of the U-blocks, the higher tropical powers of the matrix M exhibit certain regularities that are shown in the next two sections. These are used further to optimize the tropical APSP algorithm for the specific case of bipartite graphs.

Tropical operations on the matrix blocks

In this section, the properties of the U-blocks of the matrix M in tropical algebra are considered. First, it is shown that the matrix U is similar to the identity matrix of linear algebra. Then, the properties of the matrix U by tropical matrix multiplication and addition are introduced. The important formulae for manipulation of blocks of the weight matrix M are gathered at the end of this section.

First, consider the neutral elements of the tropical operations:

$$x \oplus \infty = \min(x, \infty) = x \quad \text{and} \quad x \odot 0 = x + 0 = x.$$

It should be noted, that the main diagonal of the square matrix U consists of the neutral element of tropical multiplication and all other elements are neutral for tropical addition. In this manner, the matrix U is similar to the identity matrix I of linear algebra. The later one contains ones (neutral elements for regular multiplication) on the main diagonal and zeros (neutral elements for regular addition) in all other positions. Such a composition makes identity matrix to neutral element for matrix multiplication with an $m \times n$ matrix B in linear algebra:

$$I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix}, \qquad I_m B = B I_n = B.$$

The similarity in the structure of the U-matrix and identity matrix leads to the assumption of the similar role of U in tropical matrix multiplication.

The following reasoning shows that the tropical multiplication of matrix U_m with an arbitrary $m \times n$ matrix B results in the same matrix B, that is,

$$U \odot B = B.$$

Consider the row vectors $u_{i,*} = (u_{i1}, \ldots, u_{im})$ of the matrix U_m. Their elements are: $u_{ij} = 0$ for $j = i$ and $u_{ij} = \infty$ for all $j \neq i$. Note that $x \odot \infty = x + \infty = \infty$. The tropical product of $u_{i,*}$ and an arbitrary column vector c gives a scalar:

$$\begin{aligned} (u_{i1}, \ldots, u_{ii}, \ldots, u_{im}) \odot (c_1, c_2, \ldots, c_m)^T & \\ &= (\infty, \ldots, 0, \ldots, \infty) \odot (c_1, c_2, \ldots, c_m)^T \\ &= \infty \odot c_1 \oplus \cdots \oplus 0 \odot c_i \oplus \cdots \oplus \infty \odot c_m \\ &= \min(\infty, \ldots, c_i, \ldots, \infty) \\ &= c_i. \end{aligned}$$

Then the tropical product of a row $u_{i,*}$ and the $m \times n$ matrix B is a row $b_{i,*}$ of this matrix:

$$(u_{i1}, u_{i2}, \ldots, u_{im}) \odot \begin{pmatrix} b_{1,1} & \cdots & b_{1,n} \\ \vdots & \ddots & \vdots \\ b_{m,1} & \cdots & b_{m,n} \end{pmatrix} = (u_i \odot b_{*,1}, u_i \odot b_{*,2}, \ldots, u_i \odot b_{*,n})$$
$$= (b_{i1}, b_{i2}, \ldots, b_{in}) = b_{i,*},$$

where $b_{*,1}$, $b_{*,2}, \ldots$ are the corresponding columns of the matrix B.
Hence, the tropical product of all the rows $u_{i,*}$ with the matrix B is a matrix consisting of all rows of B, i.e., B itself. Therefore,

$$U_m \odot B = B.$$

The neutrality of the U-matrix for the right-side tropical multiplication, that is, $B \odot U_n = B$, can be shown in the same manner. □
Since the matrix U exhibits the same properties in tropical algebra as the identity matrix does in linear algebra, it can be considered as a *tropical identity matrix*.

A further point is the tropical addition of the matrix U_m with an $m \times n$ matrix B. Consider the following identity for the neutral element of tropical multiplication:

$$x \oplus 0 = \begin{cases} 0, & \text{if } x \geqslant 0, \\ x, & \text{if } x < 0. \end{cases}$$

Then the tropical sum of U and B is:

$$U \oplus B = B \oplus U = (b_{ij} \oplus u_{ij})_{m \times m} = \begin{cases} b_{ij}, & \text{if } i \neq j \text{ or } i = j \text{ and } b_{ij} < 0, \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

As can be seen, this operation affects only positive elements on the main diagonal of the matrix B setting them to zero. Such a sum is further denoted as $(B)_0$.

The properties of tropical operations on matrices are similar to those in linear algebra. The Equations (5.2a) – (5.3b) sum up the rules used for mathematical derivations in the next section.
For the matrices A, B, C and the tropical identity matrix U with appropriate dimensions the following equations hold:

for addition:

$$A \oplus A = A, \quad A \oplus B = B \oplus A, \quad (A \oplus B) \oplus C = A \oplus (B \oplus C); \tag{5.2a}$$

for multiplication:

$$A \odot B \neq B \odot A, \qquad A^{\odot 0} = U, \qquad B \odot U = U \odot B = B; \tag{5.2b}$$

distributivity (for explicit proof see Appendix B.1):

$$A \odot (B \oplus C) = A \odot B \oplus A \odot C, \tag{5.2c}$$
$$(A \oplus B) \odot C = A \odot C \oplus B \odot C. \tag{5.2d}$$

The matrix $(B)_0 = U \oplus B$ exhibits the following properties:

$$(B)_0 \oplus B = B \oplus U \oplus B \overset{5.2a}{=} (B)_0; \tag{5.3a}$$

$$(B)_0 \oplus U = B \oplus U \oplus U \overset{5.2a}{=} (B)_0. \tag{5.3b}$$

Tropical powers of the weight matrix for bipartite graphs

In this section, the detailed structure of the four blocks is given for the n-th tropical power, where $n \in \mathbb{N}$, of the weight matrix of a bigraph. At the end of the section, it is shown that one of the blocks of $M^{\odot n}$ is sufficient to obtain the rest of them, avoiding the explicit exponentiation of the whole matrix M.

In this section the following notations are used:

- The matrices $L_1^{(n)}$ and $L_2^{(n)}$ denote upper left $|V_1|\times|V_1|$ and the lower right $|V_2|\times|V_2|$ blocks of the matrix $M^{\odot n}$ respectively.
- The matrices $M_1^{(n)}$ and $M_2^{(n)}$ denote upper right $|V_1|\times|V_2|$ and the lower left $|V_2|\times|V_1|$ blocks of the matrix $M^{\odot n}$ respectively.
- Superscripts in parentheses for the four blocks above, e.g., $L_1^{(n)}$, denote the corresponding tropical power of the matrix M.
- The matrix U denotes the tropical identity matrix.
- The matrix product $M_1 M_2$ denotes tropical matrix multiplication $M_1 \odot M_2$, i.e., juxtaposition is used for tropical matrix multiplication to save space and increase readability.

The detailed composition of the four blocks in the matrix $M^{\odot n}$ is given as follows:

Proposition 5.1.1

For the weight matrix M of a bipartite graph with no negative cycles and $k \geqslant 1$, the four blocks of the matrix $M^{\odot n}$ have the following properties:

$$\begin{aligned}
L_1^{(n=2k)} &= L_1^{(2k+1)} = (M_1 M_2)_0^{\odot k}, \\
L_2^{(n=2k)} &= L_2^{(2k+1)} = (M_2 M_1)_0^{\odot k}, \\
M_1^{(n=2k)} &= M_1^{(2k-1)} = L_1^{(2(k-1))} M_1 = (M_1 M_2)_0^{\odot k-1} M_1, \\
M_2^{(n=2k)} &= M_2^{(2k-1)} = L_2^{(2(k-1))} M_2 = (M_2 M_1)_0^{\odot k-1} M_2,
\end{aligned} \tag{5.4}$$

where $(M_1 M_2)_0 = M_1 M_2 \oplus U_{|V_1|}$ and $(M_2 M_1)_0 = M_2 M_1 \oplus U_{|V_2|}$. The products $M_1 M_2$ and $M_2 M_1$ are square matrices, so the sum $(B)_0$ is defined for both of them.

Proof: The assumptions are shown by induction on n.

For $k=1$, the second power $n=2$ of the matrix is:

$$M^{\odot 2} = \begin{pmatrix} U & M_1 \\ M_2 & U \end{pmatrix} \odot \begin{pmatrix} U & M_1 \\ M_2 & U \end{pmatrix} = \begin{pmatrix} U \oplus M_1M_2 & M_1 \oplus M_1 \\ M_2 \oplus M_2 & M_2M_1 \oplus U \end{pmatrix}.$$

Regarding that $U \oplus M_1M_2 = (M_1M_2)_0$ and $M_1 \oplus M_1 \overset{5.2a}{=} M_1$, the four blocks are:

$$L_1^{(2)} = (M_1M_2)_0, \quad L_2^{(2)} = (M_2M_1)_0, \quad M_1^{(2)} = M_1, \text{ and } M_2^{(2)} = M_2.$$

According to the assumption for $k=1$:

$$L_1^{(2)} = (M_1M_2)_0^{\odot 1}, \quad L_2^{(2)} = (M_2M_1)_0^{\odot 1},$$
$$M_1^{(2)} = (M_1M_2)_0^{\odot 0}M_1 \overset{5.2b}{=} M_1 \text{ and } M_2^{(2)} = (M_2M_1)_0^{\odot 0}M_2 \overset{5.2b}{=} M_2.$$

Thus, the assumption holds for $n=2$.

Assume the proposition holds for all tropical powers of M up to even $n-1=2k$.

The n-th tropical power $M^{\odot n}$ for $n=2k+1$ is calculated as follows:

$$M^{\odot n=2k+1} = \begin{pmatrix} L_1^{(n)} & M_1^{(n)} \\ M_2^{(n)} & L_2^{(n)} \end{pmatrix} = \begin{pmatrix} L_1^{(n-1)} & M_1^{(n-1)} \\ M_2^{(n-1)} & L_2^{(n-1)} \end{pmatrix} \odot \begin{pmatrix} U & M_1 \\ M_2 & U \end{pmatrix}$$
$$= \begin{pmatrix} L_1^{(n-1)} \oplus M_1^{(n-1)}M_2 & L_1^{(n-1)}M_1 \oplus M_1^{(n-1)} \\ M_2^{(n-1)} \oplus L_2^{(n-1)}M_2 & M_2^{(n-1)}M_1 \oplus L_2^{(n-1)} \end{pmatrix}.$$

After substitution of the corresponding values for the four blocks we get by induction:

$$\begin{aligned} L_1^{(n=2k+1)} = L_1^{(2k)} \oplus M_1^{(2k)}M_2 &= (M_1M_2)_0^{\odot k} \oplus (M_1M_2)_0^{\odot k-1}M_1M_2 \\ &= (M_1M_2)_0^{\odot k-1}\big((M_1M_2)_0 \oplus M_1M_2\big) \\ &\overset{5.3a}{=} (M_1M_2)_0^{\odot k-1}(M_1M_2)_0 \\ &= (M_1M_2)_0^{\odot k}, \text{ that equals } L_1^{(2k)}; \end{aligned}$$

$$\begin{aligned} L_2^{(n=2k+1)} = M_2^{(2k)}M_1 \oplus L_2^{(2k)} &= (M_2M_1)_0^{\odot k-1}M_2M_1 \oplus (M_2M_1)_0^{\odot k} \\ &= (M_2M_1)_0^{\odot k-1}\big(M_2M_1 \oplus (M_2M_1)_0\big) \\ &= (M_2M_1)_0^{\odot k}, \text{ that equals } L_2^{(2k)}; \end{aligned}$$

$$\begin{aligned} M_1^{(n=2k+1)} = L_1^{(2k)}M_1 \oplus M_1^{(2k)} &= (M_1M_2)_0^{\odot k}M_1 \oplus (M_1M_2)_0^{\odot k-1}M_1 \\ &= \big((M_1M_2)_0 \oplus U\big)(M_1M_2)_0^{\odot k-1}M_1 \\ &\overset{5.3b}{=} (M_1M_2)_0^{\odot k}M_1, \text{ that equals } L_1^{(2k)}M_1; \end{aligned}$$

$$\begin{aligned} M_2^{(n=2k+1)} = M_2^{(2k)} \oplus L_2^{(2k)} M_2 &= (M_2M_1)_0^{\odot k-1} M_2 \oplus (M_1M_2)_0^{\odot k} M_2 \\ &= \big((M_2M_1)_0 \oplus U\big)(M_2M_1)_0^{\odot k-1} M_2 \\ &= (M_2M_1)_0^{\odot k} M_2, \text{ that equals } L_2^{(2k)} M_2. \end{aligned}$$

■

From Equation group (5.4), it can be seen that all blocks of the matrix $M^{\odot n}$ depend on two products: $(M_1M_2)_0$ and $(M_2M_1)_0$. The following equations show how to obtain the values of the four blocks after exponentiation of one of these products.

At first, consider the equation

$$(M_2M_1)_0 M_2 = M_2(M_1M_2)_0, \tag{5.5}$$

obtained by following transformation:

$$\begin{aligned} (M_2M_1)_0 M_2 = (M_2M_1 \oplus U)M_2 &\overset{5.2d}{=} M_2M_1M_2 \oplus UM_2 \\ &\overset{5.2b}{=} M_2M_1M_2 \oplus M_2U \overset{5.2c}{=} M_2(M_1M_2 \oplus U) = M_2(M_1M_2)_0. \end{aligned}$$

Consequent application of Equation (5.5) yields:

$$\begin{aligned} (M_2M_1)_0^{\odot k} M_2 &= (M_2M_1)_0^{\odot k-1} \underbrace{(M_2M_1)_0 M_2}_{M_2(M_1M_2)_0} \\ &\overset{5.5}{=} (M_2M_1)_0^{\odot k-2} \underbrace{(M_2M_1)_0 M_2}_{M_2(M_1M_2)_0} (M_1M_2)_0 = \dots \\ &\overset{5.5}{=} \underbrace{(M_2M_1)_0^{\odot 1} M_2}_{M_2(M_1M_2)_0} (M_1M_2)_0^{\odot k-1} \\ &= M_2(M_1M_2)_0^{\odot k}. \end{aligned} \tag{5.6}$$

This allows the computation of the blocks $M_1^{(n)}$ and $M_2^{(n)}$ from $(M_1M_2)_0^{\odot k}$.
The next equation allows $L_2^{(n)}$ to be obtained from $(M_1M_2)_0^{\odot k-1}$ (for explicit proof see Appendix B.2):

$$(M_2M_1)_0^{\odot k} = M_2(M_1M_2)_0^{\odot k-1} M_1 \oplus U \tag{5.7}$$

From the Equations (5.6) and (5.7), it can be seen that for obtaining the whole matrix $M^{\odot n}$ it is sufficient to find $(M_1M_2)_0^{\odot k-1}$.

5.2 Tropical APSP Algorithm for Bigraphs

In this section, the established properties of the bipartite graphs are used to formulate a special version of the tropical APSP algorithm for these graphs.

First, it is shown that the maximal number of edges in an acyclic path for a bipartite graph is less than that for a general graph.
For a directed bigraph $G=(V,E)$ with the parts V_1 and V_2, let assume without restriction that $|V_1| \leqslant |V_2|$.
The maximal number of edges in an acyclic path in G is bound by the cardinality of the smaller subset of vertices, that is, $|V_1|$. This number differs depending on the sets to which the terminal vertices belong.

– For acyclic paths with both terminal vertices in the same subset:

  - if the subset is V_2, then a path contains at most $2|V_1|$ edges;
  - if the subset is V_1, then a path contains at most $2(|V_1|-1)$, as it requires two edges for transition between two vertices of the same part.

– For acyclic paths with terminal vertices in different subsets, the maximal length is $2(|V_1|-1)+1=2|V_1|-1$.

Hence, the maximal number of edges in the acyclic path in a bipartite graph is $2|V_1|$, that is, less than $|V|-1$ edges in general one. □

This means that the $2|V_1|$-th tropical power of the weight matrix M represents the solution of the all-pairs shortest path problem for a bipartite graph.

All four blocks of the matrix $M^{\odot 2|V_1|}$ can be obtained from the matrix $(M_1M_2)_0^{\odot |V_1|-1}$ by the Equations (5.6) and (5.7) established in the previous section. So, the smaller $|V_1|\times|V_1|$ matrix $(M_1M_2)_0$ can be used in calculations instead of the $(|V_1|+|V_2|)\times(|V_1|+|V_2|)$ weight matrix M without loss of information. This leads to the specific APSP algorithm for the bipartite graphs represented by Procedure 4.
The dynamic programming matrix is initialized with the matrix $(M_1M_2)_0$. Then the $|V_1|-1$-th power of this matrix is found. Finally, the matrix $M^{\odot 2|V_1|}$, which contains the shortest paths between all the pairs of vertices in a bipartite graph is restored.
For better readability, Procedure 4 shows a straightforward variant of the matrix exponentiation, which requires $|V_1|-2$ multiplications. Actually, the n-th power of a number can be found in $O(\log_2 n)$ time using exponentiation by squaring (for implementation see Appendix B.3). This leads to $O(n^3 \log_2 n)$ complexity for matrix exponentiation, since every matrix multiplication requires n^3 operations. This algorithm exhibits the following features:

– The dimensions of the dynamic programming matrix are $|V_1|\times|V_1|$.

– The six additional matrix multiplications are used to initialize the DP matrix and restore the resulting blocks of $M^{\odot 2|V_1|}$.

– The storage required is $2|V_1|^2$, for the matrix $(M_1M_2)_0$ and the DP matrix.

Procedure 4 BipartiteAPSP

Input: two blocks of the weight matrix: $M_1,\ M_2$
Output: four blocks of $M^{\odot 2|V_1|}$: $L_1^{(2|V_1|)},\ L_2^{(2|V_1|)},\ M_1^{(2|V_1|)},\ M_2^{(2|V_1|)}$

$D = D_1 = (M_1 \odot M_2 \oplus U)$ *// DP matrix initialization*

/ Find* $(M_1M_2)_0^{\odot|V_1|-1}$ **/*
for $k = 2$ to $|V_1|-1$ **do**
  $D = D \odot D_1$
end for

/ Restore the four blocks for the resulting matrix */*
$L_1^{(2|V_1|)} = D \odot D_1$
$L_2^{(2|V_1|)} = M_2 \odot D \odot M_1 \oplus U$
$M_1^{(2|V_1|)} = D \odot M_1$
$M_2^{(2|V_1|)} = M_2 \odot L_1^{(2|V_1|)}$
return $L_1^{(2|V_1|)},\ L_2^{(2|V_1|)},\ M_1^{(2|V_1|)},\ M_2^{(2|V_1|)}$

where U is the $|V_1|\times|V_1|$ tropical identity matrix

Consider that the original APSP algorithm delivers meaningful results for the all-pairs shortest path for the graphs without negative cycles. If a graph contains such cycles, their existence is detected by finding the negative elements on the main diagonal of the final DP matrix (Section 2.4.2).
The proposed version in this case exhibits the same behaviour for the bipartite graphs. Since the blocks $L_1^{(2|V_1|)}$ and $L_2^{(2|V_1|)}$, containing the main diagonal of the resulting matrix, depend on the block $(M_1M_2)_0 = (M_1 \odot M_2 \oplus U)$, and the sum of the tropical identity matrix U and an arbitrary one retains the negative entries on the main diagonal according to Equation (5.1).

5.2.1 Time and Space Requirements

The space requirements for the general tropical APSP algorithm and the proposed one are given in Table 5.1. Due to the special structure of the bipartite graphs, the two U-blocks do not require separate memory, as their elements can be acquired dynamically from the corresponding vertices. This reduces the storage space for the weight matrix of a bipartite graph compared to the general one. The proposed version optimizes the general algorithm in two ways:

– the dimensions of the dynamic programming matrix are reduced from $(|V_1|+|V_2|)^2$ to $|V_1|^2$. This leads to acceleration of computations, as the smaller matrices require less time to multiply.

– by direct exponentiation, the number of multiplications for the dynamic programming matrix is reduced from $|V|-1=|V_1|+|V_2|-1$ in general case to $|V_1|-2$ in the inner for-cycle and six additional ones to restore the whole matrix.

In the case of an undirected bigraph, the computations take even less memory. Since the block M_2 of the weight matrix is a transposition of M_1, the latter does not require separate storage.

Table 5.1: Space requirements of the tropical APSP algorithms.

Method	*matrix dimensions*	
	weight matrix	DP matrix
tropical APSP algorithm		
directed graph	$\|V\|\times\|V\|$	$\|V\|\times\|V\|$
undirected graph	$1/2(\|V\|\times\|V\|)$	$\|V\|\times\|V\|$
tropical APSP algorithm for bigraphs		
directed bigraph	$2(\|V_1\|\times\|V_2\|)$	$2(\|V_1\|\times\|V_1\|)^*$
undirected bigraph	$\|V_1\|\times\|V_2\|$	$2(\|V_1\|\times\|V_1\|))^*$

where $|V|=|V_1|+|V_2|$;
* – two matrices are required: $(M_1M_2)_0$ and D, since $(M_1M_2)_0$ differs from the weight matrix.

The worst case for the proposed version comprises bipartite graphs with the vertex subsets of the same cardinality, i.e., $|V_1|=|V_2|$. So, the DP matrix takes $|V|^2/4$ cells and the required tropical power of the weight matrix is $|V|/2$. In this case, by direct exponentiation the computations are performed 16 times faster than by the general tropical APSP algorithm.

The developed version reduces the problem dimensions to $|V|/2$ in the worst case; the $O(|V|^3 \log_2 |V|)$ order of time complexity remains the same as in original tropical APSP algorithm. In practical terms, it means shorter computation time of the proposed version for the same size input, and accelerates the performance of the large-scale applications correspondingly.

5.3 Discussion

Graphs are widely used in computer science to represent the systems of interconnected objects. Many such systems naturally implicate a division into two groups of mutually unconnected entities. For instance, numerous hierarchical

systems, such as electric circuits or workflows. Such systems are often modeled with the help of the bipartite graphs. Morover, such popular models as Petri nets, neuronal nets, and trees represent actually bipartite graphs.
For the applications based on these models, the finding of the shortest path in a graph is an underlying procedure. Due to the high number of calls it exhibits significant influence on the performance of the large-scale applications, which operate with high number of graphs.

It may be possible to optimize the Floyd-Warshall algorithm using analysis similar to the presented one for the respective DP matrix. However, the difference in data contained in the DP matrix of the tropical and regular versions should be taken into account.

The proposed technique reduces the dimensions of the associated DP matrix and consequently, the execution time. For the worst case, when a bipartite graph contains the vertex subsets of the same cardinality $|V_1| = |V_2|$, the algorithm requires the DP matrix one-quarter of the size of that used in the tropical APSP algorithm for general graphs. This makes it highly usefull subroutine for large-scale applications based on bipartite graphs. The algorithm comprises the general case of directed weighted bipartite graphs, unlike the other existing methods for this class of graphs that handle more specific cases: restricted by weight and/or topology. Full advantages of this approach can be unfolded by using the repeated squaring technique for matrix multiplication and software libraries optimized for matrix computations, such as CUDA.

Chapter 6

Conclusion

This thesis has provided reliable and efficient computational methods in the field of DNA computing and has addressed one of the fundamental graph problems, the APSP problem. The methods introduced in Chapters 3 and 4 describe underlying techniques for the design of DNA computations. Chapter 5 concerns the shortest path problem in the graph theory.

In Chapter 3 a specific approach for *in silico* evaluation of the DNA encoding has been proposed that considers requirements of the protocol for particular DNA computations. A comprehensive workflow for development of the corresponding software modules has been further provided. The performance of the encoding is assessed by finding its thermodynamic characteristics, e.g., free energy gap, for the hybridization complexes built during the DNA computation. The outstanding characteristic of the proposed approach in comparison with the available similar methods is that it provides sufficient precision to assess the results of the particular DNA computation, while remaining lightweight regarding the full-scale simulation. Such modules can be used separately or as a part of large-scale software. The approach enhances the precision of the preliminary *in silico* simulations and allows to refine the protocol conditions of *in vitro* experiments. The protocol-specific evaluation is especially valuable for DNA computations that exploit DNA strands of unequal lengths, since it allows to select the encoding based on fixed-length DNA word sets developed by conventional DNA strand design methods.

Chapter 4 has introduced a new graph-based approach to the solution of the minimal free energy problem for the hybridization of two arbitrary single stranded DNA. The novel model of hybridization graph has been developed to represent all potential structural motifs of the DNA hybridization complex. On this model the MFE of the hybridization complex can be found as the weight of the shortest path in the graph. For solving the latter problem, three computational methods based on the paradigm of dynamic programming have been

introduced; all of them provide the optimal solution. Their comparison has shown that edge pruning technique is the most efficient one. The comparison of its runtime with that of analogous MFE methods has shown a significant speed up for the sequences with the length of up to 100 nt. Since the length of the strands currently used for the DNA computations is 16 to 32 nt, the integration of the proposed MFE method into large-scale applications for DNA strand design that use thermodynamic criteria allows to significantly improve the performance of the design software.

In Chapter 5 a new algorithm for solving the all-pairs shortest path problem for particular class of bipartite graphs has been developed. The algorithm is based on tropical multiplication of the weight matrix. The proposed technique reduces the dimensions of the associated DP matrix and consequently, the execution time. For the worst case, when the bipartite graph contains the subsets of vertices with the same cardinality, the method requires the DP matrix one-quarter of the size of that used in the tropical APSP algorithm for general graphs. The algorithm comprises the general case of directed weighted bipartite graphs, unlike the other existing methods for this class of graphs that handle more specific cases: restricted by weight and/or topology. Full advantages of this approach can be unfolded by using the repeated squaring technique for matrix multiplication and software libraries optimized for matrix computations, such as CUDA.

In this thesis a number of novel graph-based algorithms for the design of DNA computations, and for solving the APSP problem have been presented. A direction for further work concerning the first case is rather practical. For instance, extending of the MFE algorithm to a more general hybridization model including DNA self-folding.
Computational implementation of the proposed APSP algorithm has also practical importance. Moreover, the analysis similar to that presented in Chapter 5 can be applied to reduce the DP matrix of the Floyd-Warshall algorithm for the bipartite graphs. Since the latter does not employ matrix multiplication, a sub-cubic complexity for APSP algorithm on bipartite graphs can be attained.

The long term purpose of bioinformatics has two following aspects. The practical one is the enhancement of the efficiency of the biologically relevant computations; this eventually saves wet laboratory resources, time, and associated finances. The scientific one is gaining of new knowledge of the mechanisms of life. We hope the presented work would contribute to both these directions.

Bibliography

[1] J. Ackermann and F.U. Gast. Word design for biomolecular information processing. *Zeitschrift für Naturforschung*, 58(2/3):157–161, 2003.

[2] L.M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994.

[3] H.T. Allawi and J. SantaLucia, Jr. Thermodynamics and NMR of internal G-T mismatches in DNA. *Biochemistry*, 36:10581–94, 1997.

[4] H.T. Allawi and J. SantaLucia, Jr. Nearest-neighbour themodynamics of internal A-C mismatches in DNA: sequence dependance and pH effects. *Biochemistry*, 7:9435–9444, 1998.

[5] H.T. Allawi and J. SantaLucia, Jr. Nearest-neighbour themodynamics parameters for internal G-A mismatches in DNA. *Biochemistry*, 37:2170–2179, 1998.

[6] H.T. Allawi and J. SantaLucia, Jr. Themodynamics of internal CT mismatches in DNA. *Nucleic Asids Res.*, 37:2694–2701, 1998.

[7] M. Andronescu, R. Aguirre-Hernández, A. Condon, and H.H. Hoos. Rnasoft: A suite of RNA secondary structure prediction and design software tools. *Nucleic Acids Res*, 31:3416–3422, 2003.

[8] N. Ascheuer. *Hamiltonian path problems in the on-line optimization of flexible manufacturing systems.* PhD thesis, Technical University Berlin, 1995.

[9] E. Bach, A. Condon, E. Glaser, and C. Tanguay. DNA models and algorithms for NP-complete problems. *J Comput Syst Sci*, 57(2):172–186, 1998.

[10] C.S. Bassoy, S. Torgasin, M. Yang, and K.H. Zimmermann. Accelerating Scalar-Product Based Sequence Alignment using Graphics Processor Units. *Journal of Signal Processing Systems*, 61:117–125, 2010.

[11] R.E Bellman. *Dynamic Programming.* Princeton Univ. Press, Princeton, 1957.

[12] Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, and E. Shapiro. Programmable and autonomous computing machine made of biomolecules. *Nature*, 414(6862):430–434, 2001.

[13] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, and E. Shapiro. An autonomous molecular computer for logical control of gene expression. *Nature*, 429 (6990):423–429, 2004.

[14] A. Bleasby. emboss dan, bioweb.pasteur.fr/seqanal/interfaces/dan.html. 1999.

[15] R.S. Braich, N. Chelyapov, C. Johnson, P.W.K. Rothemund, and L. Adleman. Solution of a 20-variable 3-SAT problem on a DNA computer. *Science*, 296(5567):499–502, 2002.

[16] K.J. Breslauer, R. Frank, H. Blcker, and L.A. Marky. Predicting DNA duplex stability from the base sequence. *Biochemistry*, 83:3746–3750, 1986.

[17] C.R. Cantor and P.R. Schimmel. *Biophysical Chemistry: The conformation of biological macromolecules*. WH Freeman & Co, 1980.

[18] J. Chen, R. Deaton, M. Garzon, J.W. Kim, D.H. Wood, H. Bi, D. Carpenter, and Y.Z. Wang. Characterization of non-crosshybridizing DNA oligonucleotides manufactured in vitro. *Natural Computing*, 5(2):165–181, 2006.

[19] L. Chen. Solving the shortest-paths problem on bipartite permutation graphs efficiently. *Information processing letters*, 55(5):259–264, 1995.

[20] H. Chin-Wen and J.M. Chang. Solving the all-pairs-shortest-length problem on chordal bipartite graphs. *Information processing letters*, 69(2):87–93, 1999.

[21] D.M. Crothers and B.H. Zimm. Theory of the melting transition of synthetic polynucleotides: Evaluation of the stacking free energy*. *Journal of Molecular Biology*, 9(1):1–9, 1964.

[22] R. Deaton, R.C. Murphy, M. Garzon, D.R. Franceschetti, and S.E. Stevens, Jr. Good encodings for DNA-based solutions to combinatorial problems. In *Proceedings of the Second Annual Meeting on DNA Based Computers*, volume 44, pages 247–259, 1996.

[23] R. Deaton, J. Chen, H. Bi, M. Garzon, H. Rubin, and D.H. Wood. A PCR-based protocol for in vitro selection of non-crosshybridizing oligonucleotides. *DNA Computing*, pages 196–204, 2003.

[24] R. Deaton, K. Jin-Woo, and J. Chen. Design and test of noncrosshybridizing oligonucleotide building blocks for DNA computers and nanostructures.

Applied Physics Letters, 82(8):1305–1307, feb 2003.

[25] H. DeVoe and I. Tinoco, Jr. The stability of helical polynucleotides: base contributions. *Journal of Molecular Biology*, 4(6):500–517, 1962.

[26] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[27] W. Dobosiewicz. A more efficient algorithm for the min-plus multiplication. *International journal of computer mathematics*, 32(1):49–60, 1990.

[28] D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29:1740, 2000.

[29] F.F. Dragan. Estimating all pairs shortest paths in restricted graph families: a unified approach. *Journal of Algorithms*, 57(1):1–21, 2005.

[30] U. Feldkamp, H. Rauhe, and W. Banzhaf. Software tools for DNA sequence design. *Genetic Programming and Evolvable Machines*, 4(2):153–171, 2003.

[31] R.P. Feynman. Miniaturization. *Reinhold, New York*, pages 282–296, 1961.

[32] R.W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, 1962.

[33] M.L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing*, 5:83, 1976.

[34] M. Garzon, P. Neathery, R. Deaton, R.C. Murphy, D.R. Franceschetti, and S.E. Stevens, Jr. A new metric for DNA computing. *Genetic Programming*, pages 479–490, 1997.

[35] M. Garzon, E. Drumwright, R.J. Deaton, and D. Renault. Virtual test tubes: a new methodology for computing. volume 0, page 116. IEEE Computer Society, 2000.

[36] M. Garzon, V. Phan, S. Roy, and A. Neel. In search of optimal codes for DNA computing. *DNA Computing*, pages 143–156, 2006.

[37] M. Garzon, R. Deaton, and M. Wittner. Encoding Genomes for DNA Computing. 2009.

[38] M.H. Garzon and R.J. Deaton. Codeword design and information encoding in DNA ensembles. *Natural Computing*, 3(3):253–292, 2004.

[39] N.L. Goddard, G. Bonnet, O. Krichevsky, and A. Libchaber. Sequence dependent rigidity of single stranded DNA. *Physical review letters*, 85(11): 2400–2403, 2000.

[40] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, 1982.

[41] O. Gotoh and Y. Tagashira. Stabilities of Nearest-Neighbor Doublets in Double-Helical DNA Determined by Fitting Calculated Melting Profiles to Observed Profiles. *Biopolimers*, 20:1033, 1981.

[42] A.R. Gruber, R. Lorenz, S.H. Bernhart, R. Neubock, and I.L. Hofacker. The Vienna RNA websuite. *Nucleic Acids Research*, 36, 2008.

[43] F. Guarnieri, M. Fliss, and C. Bancroft. Making DNA add. *Science*, 273 (5272):220, 1996.

[44] V. Gupta, S. Parthasarathy, and M.J. Zaki. Arithmetic and logic operations with DNA. In *Proceedings of the 3rd DIMACS Workshop on DNA Based Computers*, pages 212–220. Citeseer, 1997.

[45] A.J. Hartemink, D.K. Gifford, and J. Khodor. Automated constraint-based nucleotide sequence selection for DNA computation. *Biosystems*, 52(1–3): 227–235, 1999.

[46] T. Head, X. Chen, M.J. Nichols, M. Yamamura, and S. Gal. Aqueous Solutions of Algorithmic Problems: emphasizing knights on a 3X3. In *DNA computing: 7th Int. Workshop on DNA-Based Computers, Tampa*, page 191. Springer Verlag, 2002.

[47] C.V. Henkel, R.S. Bladergroen, C.I.A. Balog, A.M. Deelder, T. Head, G. Rozenberg, and H.P. Spaink. Protein output for DNA computing. *Natural Computing*, 4(1):1–10, 2005.

[48] D.I.T. Hinze. *Universelle Modelle und ausgewählte Algorithmen des DNA-Computing*. PhD thesis, Techn. Universitat Dresden, 2002.

[49] I.L. Hofacker, W. Fontana, P.F. Stadler, S.L. Bonhoeffer, M. Tacker, and P. Schuster. Fast Folding and Comparison of RNA Secondary Structures. *Monatsh. Chem.*, 125:167–188, 1994.

[50] Z. Ignatova, K.H. Zimmermann, and I.M. Martínez-Pérez. *DNA computing models*. Springer-Verlag New York Inc, 2008.

[51] H. Jacobson and W.H. Stockmayer. Intramolecular reaction in polycondensations. I. The theory of linear systems. *The Journal of Chemical Physics*, 18:1600, 1950.

[52] D.B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977.

[53] L. Kaderali. Selecting Target Specific Probes for DNA Arrays. Master's thesis, Köln, 2001.

[54] L. Kaderali and A. Schliep. Selecting signature oligonucleotides to identify

organisms using DNA arrays. *Bioinformatics*, 18:1340–1349, 2002.

[55] P. Kaplan, G. Cecchi, and A. Libchaber. Molecular computation: Adlemans experiment repeated. *Technical report, NEC research Institute*, 1995.

[56] N. Kummerfeldt. In silico autonomous DNA computing. Master's thesis, Hamburg Univ. Tech., 2010.

[57] T.B. Kurniawan, N.K. Khalid, Z. Ibrahim, M. Khalid, and M. Middendorf. An Ant Colony System for DNA sequence design based on thermodynamics. In *Proceedings of the Fourth IASTED Int. Conference on Advances in Comp. Science and Technology*, pages 144–149. ACTA Press, 2008.

[58] N. Le Novere. Melting, computing the melting temperature of nucleic acid duplex. *Bioinformatics*, 17:1226–1227, 2001.

[59] M. Leber, L. Kaderali, A. Schönhuth, and R. Schrader. A fractional programming approach to efficient DNA melting temperature calculation. *Bioinformatics*, 21(10):2375–2382, 2005.

[60] D. Li, X. Li, H. Huang, and X. Li. Scalability of the surface-based DNA algorithm for 3-SAT. *BioSystems*, 85(2):95–98, 2006.

[61] R.J. Lipton. DNA Solution of Hard Computational Problems. *Science*, 268:542–545, 1995.

[62] W. Liu, F. Zhang, and J. Xu. A DNA algorithm for the graph coloring problem. *J. Chem. Inf. Comput. Sci*, 42(5):1176–1178, 2002.

[63] X. Liu and S. Wang. Development of an in vivo computer for SAT problem. *Mathematical and Computer Modelling*, 2010.

[64] Y. Liu, J. Xu, L. Pan, and S. Wang. DNA solution of a graph coloring problem. *J. Chem. Inf. Comput. Sci*, 42(3):524–528, 2002.

[65] T.P. Mann and W.S. Noble. Efficient Identification of DNA hybridisation partners in sequence database. *Bioinformatics*, 22:e350–e358, 2006.

[66] A. Marathe, A.E. Condon, and R.M. Corn. On combinatorial DNA word design. *Journal of Computational Biology*, 8(3):201–219, 2001.

[67] N.R. Markham and M. Zuker. DINAMelt web server for nucleic acid melting prediction. *Nucleic Acids Research*, 33, 2005.

[68] I.M. Martinez-Perez. *Biomolecular computing models for graph problems and finite state automata*. PhD thesis, Hamburg Univ. Tech., 2007.

[69] I.M. Martinez-Perez, Z. Ignatova, G. Zhang, and K.-H. Zimmermann. Biomolecular autonomous solution of the Hamiltonian path problem via

hairpin formation. *IJBRA*, 1:389–398, 2006.

[70] I.M. Martínez-Pérez, G. Zhang, Z. Ignatova, and K.H. Zimmermann. Computational genes: a tool for molecular diagnosis and therapy of aberrant mutational phenotype. *BMC Bioinformatics*, 8(1):365, 2007.

[71] Mathews. Lab. webpage, last visited: Nov. 2008. URL `http://rna.urmc.rochester.edu/`.

[72] MeltTemp. Wisconsin Package, Accelrys Inc.(c), San Diego, CA. 2002.

[73] H. Nakagawa, K. Sakamoto, and Y. Sakakibara. Development of an in vivo computer based on Escherichia coli. *DNA Computing*, pages 203–212, 2006.

[74] 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(3):443–453, 1970.

[75] P. Pancoska, Z. Moravek, and U.M. Moll. Rational design of DNA sequences for nanotechnology, microarrays and molecular computers using eulerian graphs. *Nucleic Acids Research*, 32(15):4630–4645, 2004.

[76] M.J. Patitz. Simulation of self-assembly in the abstract tile assembly model with ISU TAS. *Arxiv preprint arXiv:1101.5151*, 2011.

[77] R. Penchovsky and J. Ackermann. DNA library design for molecular computation. *Journal of Computational Biology*, 10(2):215–229, 2003.

[78] N. Peyret. *Prediction of nucleic acid hybridisation: Parameters and algorithms*. PhD thesis, Wayne State University, Detroit, Michigan, 2000.

[79] B. Pfannkuche. In silico autonomous DNA computing. Master's thesis, Hamburg Univ. Tech., 2008.

[80] P.W.K. Rothemund, N. Papadakis, and E. Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology*, 2(12):e424, 2004.

[81] S. Roweis, E. Winfree, R. Burgoyne, N. Chelyapov, M. Goodman, P. Rothemund, and L. Adleman. A sticker based architecture for DNA computation. In *2nd DIMACS workshop on DNA based computers, Princeton*, pages 1–27, 1996.

[82] J. SantaLucia, Jr. A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. *Proceedings of the National Academy of Sciences*, 95(4):1460, 1998.

[83] J. SantaLucia, Jr. Lab. webpage, last visited: Nov. 2008. URL `http://ozone3.chem.wayne.edu/home/`.

[84] J. SantaLucia, Jr. and D. Hicks. The thermodynamics of DNA structural motifs. *Annu Rev Biophys Biomol Struct*, 33:415–440, 2004.

[85] S.-Y. Shin, D.-M. Kim, I.-H. Lee, and B.-T. Zhang. Evolutionary sequence generation for reliable DNA computing. In *Proceedings of the Congress on Evolutionary Computing*, pages 79–84, 2002.

[86] M.R. Shortreed, S.B. Chang, D.G. Hong, M. Phillips, B. Campion, D.C. Tulpan, M. Andronescu, A. Condon, H.H. Hoos, and L.M. Smith. A thermodynamic approach to designing structure-free combinatorial DNA word sets. *Nucleic acids research*, 33(15):4965, 2005.

[87] I. Simon. Recognizable sets with multiplicities in the tropical semiring. In *Mathematical Foundations of Computer Science 1988*, pages 107–120. Springer, 1988.

[88] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[89] T. Takaoka. Sub-cubic Cost Algorithms for the All Pairs Shortest Path Problem. *Algorithmica*, 20:309–318, 1995.

[90] F. Tanaka, M. Nakatsugawa, M. Yamamoto, T. Shiba, and A. Ohuchi. Towards a general-purpose sequence design system in DNA computing. In *Proceedings of the Congress on Evolutionary Computation, 2002*, volume 1, pages 73–78. IEEE, 2002.

[91] F. Tanaka, A. Kameda, M. Yamamoto, and A. Ohuchi. Design of nucleic acid sequences for DNA computing based on a thermodynamic approach. *Nucleic Acids Research*, 33(3):903, 2005.

[92] F. Tanaka, A. Kameda, M. Yamamoto, and A. Ohuchi. Specificity of hybridization between DNA sequences based on free energy. *DNA Computing*, pages 371–379, 2006.

[93] I. Tinoko, Jr., O.C. Uhlenbeck, and M.D. Levine. Estimation of secondary structure in ribonucleic acids. *Nature*, 230:362–367, apr 1971.

[94] S. Torgasin and K.H. Zimmermann. Dynamic programming algorithm for thermodynamically based prediction of DNA/DNA crosshybridisation [abstract]. In *Proceedings of the GCB 2008, Dresden*, page 82.

[95] S. Torgasin and K.H. Zimmermann. Algorithm for thermodynamically based prediction of DNA/DNA cross-hybridisation. *Int. J. Bioinformatics Research and Applications*, 6(1):82–97, 2010.

[96] R. Torres-Velázquez and V. Estivill-Castro. Local search for Hamiltonian Path with applications to clustering visitation paths. *Journal of the Oper-*

ational Research Society, 55(7):737–748, 2004.

[97] D. Tulpan. *Effective Heuristic Methods for DNA Strand Design.* PhD thesis, POLITEHNICA University of Bucharest, 2006.

[98] D. Tulpan, M. Andronescu, S.B. Chang, M.R. Shortreed, A. Condon, H.H. Hoos, and L.M. Smith. Thermodynamically based DNA strand design. *Nucl. Acids Res.*, 33(15):4951–4964, 2005.

[99] P. Villesen. FaBox: an online toolbox for fasta sequences. *Molecular Ecology Notes*, 7 (6):965–968, 2007.

[100] A.V. Vologodskii, B.R. Amirikyan, Y.L. Lyubchenko, and M.D. Frank-Kamenetskii. Allowance for heterogeneous stacking in the DNA helix-coil transition theory. *Journal of biomolecular structure & dynamics*, 2(1):131, 1984.

[101] S. Warshall. A theorem on boolean matrices. *J. ACM*, 9(1):11–12, 1962.

[102] P. Wasiewicz, J.J. Mulawka, W.R. Rudnicki, and B. Lesyng. Adding numbers with DNA. In *PROC IEEE INT CONF SYST MAN CYBERN*, volume 1, pages 265–270, 2000.

[103] M.S. Waterman and T.F. Smith. RNA Secondary Structure: A Complete Mathematical Analysis. *Mathematical Biosciences*, (42):257–266, 1978.

[104] E. Winfree. On the Computational Power of DNA Annealing and Ligation. In *DNA Based Computers, volume 27 of DIMACS*, pages 199–221. American Mathematical Society, 1995.

[105] E. Winfree, R. Schulman, and C. Evans. The Xgrow simulator. URL `http://www.dna.caltech.edu/Xgrow/`.

[106] J. Xu, X. Qiang, F. Gang, and K. Zhou. A DNA computer model for solving vertex coloring problem. *Chinese Science Bulletin*, 51(20):2541–2549, 2006.

[107] Z.X. Yin, J.Z. Cui, W. Liu, X.H. Shi, and J. Xu. Plasmid resolving the satisfiability problem with DNA computing models. *Journal of Computational and Theoretical Nanoscience, 4*, 7(8):1243–1248, 2007.

[108] Q. Zhang, B. Wang, X. Wei, X. Fang, and C. Zhou. DNA Word Set Design Based on Minimum Free Energy. *NanoBioscience, IEEE Transactions on*, PP(99):1, 2010.

[109] J. Zhao, L.L. Qian, Q. Liu, Z.Z. Zhang, and L. He. DNA addition using linear self-assembly. *Chinese Science Bulletin*, 52(11):1462–1467, 2007.

[110] K.H. Zimmermann, I.M. Martinez-Perez, Z. Ignatova, and Z. Gong. Solving the Maximum Clique Problem via DNA Hairpin Formation. 2006.

[111] M. Zuker and D. Sankoff. RNA Secondary Structures and Their Stability. *Bulletin of Mathematical Biology*, 46(4):591–621, 1984.

[112] M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Research*, 9(1):133–148, nov 1981.

[113] U. Zwick. A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. *Algorithmica*, 46(2):181–192, 2006.

Appendix A

Negative control DNA encodings

Table A.1: Negative control encodings 16 and 17 from Table 3.2

vertex	set 16	set 17
v_0	ggctggcatgcctagactgt	tactcatatggggttatacg
v_1	ggctgggtaatagaacctgt	ctccgcctgggcttagctta
v_2	catgcctagactgtacgcac	gatcctctgtttcctcagct
v_3	ctaaggacatctgtggctgg	ggctccacttgcttcgctta
v_4	gtaatagaactcgacggtcg	tatgggctagcggtccggtt
v_5	acgcacacattagcgacttc	gcccttgtagtctcgggtcc
v_6	acattcgacggtcgggctgg	ttcctgtaacttgcctctaa

Appendix B

Expressions used in Chapter 5

B.1 Distributive property of tropical matrix multiplication

$$A \odot (B \oplus C) = A \odot B \oplus A \odot C,$$

where A, B, and C are the matrices with the appropriate dimensions.
Proof:
The following reasoning shows that the general element of the left hand side is the same as that of the right hand side.
Consider the following basic issues:
distributivity of the tropical multiplication for the real numbers:

$$a \odot (b \oplus c) = a + min(b, c) = min(a + b, a + c) = a \odot b \oplus a \odot c,$$

and the definition of the tropical matrix multiplication:

$$A \odot B = \bigoplus_{k=1..n} A_{ik} \odot B_{kj}.$$

The following transformation proves the ojective.

$$\begin{aligned}
(A \odot (B \oplus C))_{ij} &= \bigoplus (A_{ik} \odot (B \oplus C)_{kj}) \text{– definition of matrix multiplication} \\
&= \bigoplus (A_{ik} \odot (B_{kj} \oplus C_{kj})) \text{– definition of matrix addition} \\
&= \bigoplus (A_{ik} \odot B_{kj} \oplus A_{ik} \odot C_{kj}) \text{– distributivity of the real numbers} \\
&= \bigoplus A_{ik} \odot B_{kj} \oplus \bigoplus A_{ik} \odot C_{kj} \text{– commutativity of the real numbers} \\
&= (A \odot B)_{ij} \oplus (A \odot C)_{ij} \text{ – definition of matrix multiplication}
\end{aligned}$$

where the sum $\bigoplus$ is taken from 1 to n. □

B.2 Proof for Equation (5.7) in Section 5.1 (page 93)

$$(M_2M_1)_0^{\odot k} = M_2(M_1M_2)_0^{\odot k-1}M_1 \oplus (M_2M_1)_0 \tag{5.7}$$

Consider that $\big(M_2M_1 \oplus U\big)$ is denoted as $(M_2M_1)_0$.
According to Equation (5.5) (page 93):

$$(M_2M_1)_0M_2 = M_2(M_1M_2)_0, \tag{5.5}$$

thus, $(M_2M_1)_0M_2M_1$ equals $M_2(M_1M_2)_0M_1$.

The objective is obtained through the following transformation:

$$\begin{aligned}
(M_2M_1)_0^{\odot k} &= (M_2M_1)_0^{\odot k-1}\big(M_2M_1 \oplus U\big) \\
&= (M_2M_1)_0^{\odot k-1}M_2M_1 \oplus \underbrace{(M_2M_1)_0^{\odot k-1}}_{(M_2M_1)_0^{\odot k-2}(M_2M_1\oplus U)} \\
&= \underbrace{(M_2M_1)_0^{\odot k-1}M_2M_1}_{M_2(M_1M_2)_0^{\odot k-1}M_1} \oplus \underbrace{(M_2M_1)_0^{\odot k-2}M_2M_1}_{M_2(M_1M_2)_0^{\odot k-2}M_1} \oplus \ldots \oplus \underbrace{(M_2M_1)_0^{\odot 1}M_2M_1}_{M_2(M_1M_2)_0^{\odot 1}M_1} \\
&\quad \oplus \underbrace{(M_2M_1)_0^{\odot 1}}_{M_2M_1\oplus U} \\
&= M_2\big[\,(M_1M_2)_0^{\odot k-1}M_1 \oplus (M_1M_2)_0^{\odot k-2}M_1 \oplus \ldots \oplus (M_1M_2)_0^{\odot 1}M_1 \oplus M_1\,\big] \oplus U \\
&= M_2\big[\,(M_1M_2)_0^{\odot k-1} \oplus (M_1M_2)_0^{\odot k-2} \oplus \ldots \oplus \underbrace{(M_1M_2)_0^{\odot 1} \oplus U}_{B_0\oplus U=B_0\,(5.3a)}\,\big]M_1 \oplus U \\
&= M_2\big[\,(M_1M_2)_0^{\odot k-2} \oplus (M_1M_2)_0^{\odot k-3} \oplus \ldots \oplus \underbrace{(M_1M_2)_0^{\odot 1} \oplus U}_{B_0\oplus U=B_0\,(5.3a)}\,\big](M_1M_2)_0M_1 \oplus U \\
&\;\;\vdots \\
&= M_2\big[\,(M_1M_2)_0^{\odot 1} \oplus U\,\big](M_1M_2)_0^{\odot k-2}M_1 \oplus U \\
&= M_2(M_1M_2)_0^{\odot k-1}M_1 \oplus U
\end{aligned}$$

□

B.3 Exponentiation by Squaring

Algorithm for finding x^n with $O(\log_2(n))$ time complexity

Procedure 5 $O(\log_2(n))$ algorithm for finding x^n

Input: x, n
Output: x^n

$res = 1$
while $n \neq 0$ **do**
  if $n \; mod \; 2 \neq 0$ **then**
    $res = res \cdot x$
    $n = n - 1$
  end if
  $x = x \cdot x$
  $n = n/2$
end while
return res

List of Figures

List of Tables

List of Publications

1. S. Torgasin and K.H. Zimmermann. Algorithm for thermodynamically based prediction of DNA/DNA cross-hybridisation. *Int. J. Bioinformatics Research and Applications*, 6(1):82–97, 2010.
2. C.S. Bassoy, S. Torgasin, M. Yang, and K.H. Zimmermann. Accelerating Scalar-Product Based Sequence Alignment using Graphics Processor Units. *Journal of Signal Processing Systems*, 61:117–125, 2010.
3. S. Torgasin and K.H. Zimmermann. Dynamic programming algorithm for thermodynamically based prediction of DNA/DNA crosshybridisation [abstract]. In *Proceedings of the GCB 2008, Dresden*, page 82.

Curicculum Vitae

Born on 15th of August 1975 in Frunze, Khirgizstan.

Sept. 1992 – July 1997	Diploma in Computer Science, University of Civil Aviation, Kiev, the Ukraine.
Nov. 1998 – Aug. 1999	Software engineer (apprentice), SEH Computertechnik GmbH, Bielefeld, Germany.
Sept. 2000 – Feb. 2004	Diploma in Life Science Informatics, University Bielefeld, Germany.
May 2004 – Aug. 2006	Tutor in informatics and genetics. Private.
Sept. 2006 – July 2011	Ph.D. student, Hamburg University of Technology, Germany.