

2008

DNA Hairpin Secondary Structure Design

Yan Zeng
Western University

Follow this and additional works at: <https://ir.lib.uwo.ca/digitizedtheses>

Recommended Citation

Zeng, Yan, "DNA Hairpin Secondary Structure Design" (2008). *Digitized Theses*. 4199.
<https://ir.lib.uwo.ca/digitizedtheses/4199>


This Thesis is brought to you for free and open access by the Digitized Special Collections at Scholarship@Western. It has been accepted for inclusion in Digitized Theses by an authorized administrator of Scholarship@Western. For more information, please contact wlsadmin@uwo.ca.

DNA Hairpin Secondary Structure Design

(Thesis format: Monograph)

by

Yan Zeng


Graduate Program
in
Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

© Yan Zeng 2008

THE UNIVERSITY OF WESTERN ONTARIO
THE SCHOOL OF GRADUATE AND POSTDOCTORAL STUDIES
CERTIFICATE OF EXAMINATION

Supervisor

Examiners

Dr. Lila Kari

Dr. Kaizhong Zhang

Dr. Mark Daley

Dr. Priti Krishna

The thesis by

Yan Zeng

entitled:

DNA Hairpin Secondary Structure Design

is accepted in partial fulfillment of the
requirements for the degree of
Master of Science

Date

Chair of the Examination Board
Dr. Hanan Lutfiyya

Abstract

In this thesis, we propose a bottom-up method to design single-stranded DNA sequences that form consecutive hairpin structures. This work was inspired by the hairpin-based DNA multi-state machine proposed by Takahashi *et al.* in 2004. They have successfully achieved this DNA multiple-hairpin structure in a laboratory experiment and proposed two possible applications. The first one is to construct a random access memory (RAM) by using the DNA machines as the access address for the data. The second one is to solve the maximum independent set problem (MISP). It is interesting thus to investigate how to design DNA sequences which form consecutive hairpin structures as mentioned above. We propose a bottom-up approach to construct consecutive hairpin structures, grounded on a so-called bond-free property, and several combinatorial constraints. A software is implemented to study the behavior of our bottom-up approach. We also calculate the maximal number of sequences that correctly fold into the desired multiple-hairpin structure. This calculation provides an estimation for the size of the memory that can be constructed using Takahashi *et al.*'s method. Lastly, by selecting suitable parameters, we successfully construct a set of sequences that can fold in to the desirable multiple-hairpin structure. For example, our software is able to generate 120 sequences that can fold into a four-hairpin structure where the length of each hairpin stem is 20, the length of each hairpin loop is 7 and the external segment is 20. We validate these sequences using the molecule secondary structure prediction package, *Vienna RNA secondary structure package*.

Keywords: DNA computing, DNA secondary structure, Multiple hairpins.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr. Lila Kari for her invaluable supervision, advice and guidance. She impressed me with her deep knowledge in DNA computing and her confidence and ability to solve any kind of difficulties.

During my master study, I took many excellent courses related to DNA computing and Bioinformatics, and also some fundamental courses related to Computer Science. I would like to thank the professors of these courses Dr. Yuri Boykov, Dr. Mark Daley, Dr. Lila Kari, Dr. Lynda Robbins, Dr. Roberto Solis-Oba, Dr. Sheng Yu, and Dr. Kaizhong Zhang for their dedication and time, because I really learned a lot from these courses.

Moreover, I would like to thank Shinnosuke Seki for suggesting I study the multiple-hairpin structures; Dr. Elena Czeizler and Dr. Eugen Czeizler for their careful proof reading of my thesis proposal; Bo Cui for providing me valuable advice about the thesis and his kind assistance in reading the thesis; and all other colleagues in the Biomcomputing Laboratory.

Last but not least, I wish to thank my parents and my brother, without whose endless support and encouragement my studies leading to the degree of Master of Science would not have been possible.

Dedication

to my family

Contents

Certificate of Examination	ii
Abstract	iii
Acknowledgements	iv
Dedication	v
Table of Contents	vi
List of Tables	ix
List of Figures	xi
1 Introduction	1
1.1 The Structure of DNA and RNA	1
1.1.1 The Structure of DNA	1
1.1.2 The Structure of RNA	3
1.2 The Objective of this Research	4
1.3 The Organization of the Thesis	5
2 Basic Notations and Background Information	7
2.1 Preliminary Definitions	7
2.2 Hamming Distance and Hamming Ball	8
2.3 Automata Theory	9

3	Literature Review	13
3.1	DNA Computing	13
3.2	DNA Sequence Design	15
3.2.1	The DNA Sequence Design Problem	15
3.2.2	Constraints for DNA Sequences Design	17
3.3	Approaches Related to DNA Word Design	18
3.3.1	The Stochastic Local Search Approach	19
3.3.2	The Unique Subsequence Approach	19
3.3.3	Word-Block Construction Scheme	22
3.3.4	Method Using Bond-free Languages	23
3.3.5	DNA Word Design Software	25
3.4	Hairpin Secondary Structure Design	26
3.4.1	Secondary Structure Design Problem	26
3.4.2	Methods Related to Structure Design Problem	28
3.4.3	The Hairpin Secondary Structure	29
3.5	Motivation of the Thesis	30
4	DNA Multiple-Hairpin Structure Design	36
4.1	Problem Statement and Solution	37
4.1.1	Problem Statement	37
4.1.2	Overview of Our Bottom-up Approach	39
4.2	Generate the Bond-free Language	42
4.2.1	Generate the Bond-free Language S^\otimes from a Set S	43
4.2.2	The Selection of the Set S	44
4.2.3	Calculate the Cardinality of $S^\otimes(l)$	46
4.2.4	A Bond-free Language Closed under Concatenation	49
4.3	Construct the Segments of the Multiple-hairpin Structure	52
4.3.1	Construct Subsets S_1 and S_2 from S	54

4.3.2	Construct Sets $B_{S_1}(s)$ and $B_{S_2}(l)$	55
4.4	Apply Additional Combinatorial Constraints	56
4.5	Generate a Set of Sequences Folding into a Given Multiple-hairpin Structure	58
4.6	Predict the Secondary Structures of the Result Sequences	60
5	Implementation and Experimental Results	64
5.1	Main Structure of the Software	64
5.2	Algorithms Related to Bond-free Languages	68
5.2.1	Algorithm to Select Set S	68
5.2.2	Method to Construct the Sets S_1 and S_2	71
5.2.3	Bond-free Languages Closed under Catenation	74
5.3	Bond-free Languages: Experimental Results	80
5.4	Constructing a Multiple-hairpin Structure: Experiments	84
5.5	Discussions	91
6	Conclusions and Future Work	93
6.1	Conclusions	93
6.2	Future Work	95
	Bibliography	98
	Curriculum Vitae	104

List of Tables

5.1	The mapping of DNA words of length 2 to quadruple code based on the mapping α that $\alpha(0) = A$, $\alpha(1) = C$, $\alpha(2) = G$, and $\alpha(3) = T$	70
5.2	The mapping of DNA words of length 2 to quadruple code based on the mapping β that $\beta(0) = G$, $\beta(1) = T$, $\beta(2) = A$, and $\beta(3) = C$	70
5.3	Sets S , S_1 and S_2 with parameters $k = 3$, $d_1 = 1$ and $d_2 = 0$	76
5.4	List of all the suffixes of S_1 , i.e., $w \in SU$, and the corresponding words v in S_1 satisfying $\text{Sub}_k(wv) \subseteq S$	77
5.5	Given a set S_1 , the table contains all the pairs of cur_SU and cur_BE we considered in the algorithm.	77
5.6	List of all the suffixes of S_2 , i.e., $w \in SU$, and the corresponding words v in S_2 satisfying $\text{Sub}_k(wv) \subseteq S$	78
5.7	Given set S_2 , all the pairs of cur_SU and cur_BE we considered in the algorithm.	78
5.8	Given parameters $k = 2$, $d_1 = 0$ and mapping α , this table gives the size of set $B(l)$ under various initial words and various lengths l	80
5.9	Given $k = 3$, $d_1 = 0$ and mapping α , this table gives the size of set $B(l)$ under various initial words and lengths l	81
5.10	Given $k = 3$, this table shows the cardinality of set $B(6)$ under various initial words and d_1	83
5.11	Given $k = 3$, mapping β and the initial word GGG, this table shows the words in S obtained under various d_1	84

5.12	The words in B_{stem} and $\theta(B_{stem})$ of Example Set 1 with $c = 5$ and $d_3 = 10$	85
5.13	The words in B_{loop} of Example Set 1.	87
5.14	The predictive results of Example Set 1.	87
5.15	The words in B_{stem} and $\theta(B_{stem})$ of Example Set 1 with the Hamming constraint $d_3 = 9$	89
5.16	The predictive results of Example Set 1 with $d_3 = 9$	89
5.17	The words in B_{stem} , $\theta(B_{stem})$ and some of words in B_{loop} of Example Set 2.	90
5.18	The predictive results of Example Set 2.	91

List of Figures

1.1	The structure of a double-stranded DNA and the structure of single-stranded RNA [38].	2
1.2	A DNA hairpin structure.	3
2.1	The DFA M in Example 1 accepts the language L over the alphabet $\{A, C, T\}$ consisting of all the words that are ending with AC.	10
2.2	A trie and an automaton representing the same set of words.	12
3.1	Single-stranded DNA molecules representing vertex u , vertex v and edge e from vertex u to vertex v that hybridize and form a partially-double-stranded DNA molecule.	14
3.2	Example of the undesirable hybridization.	16
3.3	DNA Holliday junction.	20
3.4	Tree structure used to build words of length n_s from shorter unique sequences of length $n_b = 4 < n_s$. There is an arrow from node u to node v iff $\text{Suff}_3(u) = \text{Pref}_3(v)$	21
3.5	Hybridization of two segments with mismatches.	24
3.6	An example of secondary structure.	27
3.7	A Multi-state DNA Machine.	30
3.8	Opening the first hairpin.	32
3.9	Opening the second hairpin.	32
3.10	The four hairpin structure is fully opened after applying four distinct keys.	33

3.11	Whiplash PCR simulates a state transition machine that transits from state A to state B by transition rule $A \rightarrow B$	35
4.1	Correct multiple-hairpin structure $hps(2, 13, 7, \theta)$	38
4.2	The bottom-up approach using to design the multiple-hairpin structure.	41
4.3	Trie constructed based on set $S = \{AA, AC, CA, CC\}$	43
4.4	The word $ACACTC \in S^\otimes$ is constructed by the overlaps of words AC, CA, AC, CT, and TC in S	45
4.5	The words w of length l end with the suffix $su = v \cdot e$ of length $n = k - 1$ and the words of length $l - 1$ end with suffix $su' = b \cdot v$ of length $n = k - 1$ where $\text{Pref}_{n-1}(su) = \text{Suff}_{n-1}(su') = v$ and $b \cdot v \cdot e \in S$	47
4.6	The trie represents the set $S = \{AA, AC, CA, CC, CT, TC\}$ such that the end set is $EN = \{AA, AC, CA, CC, TC\}$ and the begin set is $BE = \{AA, AC, CA, CC, CT\}$	50
4.7	The 2D secondary structure of RNA molecule GGGCUAUUAGCUCAGU-UGG and DNA molecule TTGGGCTATTAGCTCAGTTGG.	62
5.1	The flow chart of software.	66
5.2	The inheritance relations of templates <code>word</code> , <code>sort_set</code> , <code>hamming_set</code> , <code>bond_free_set</code> , <code>select_subset</code> , <code>trie</code> , <code>filter</code> and <code>check_struct</code>	67
5.3	Two kinds of overlaps between words w_1 and w_2	72
5.4	Correct DNA secondary structure with four consecutive hairpins.	86
5.5	The structures of the result sequences predicted by the Vienna RNA package.	88
5.6	Some structures of the result sequences, with $k = 4$, $d_1 = 0$ and $d_2 = 0$	91
6.1	The new multiple-hairpin structure with five hairpins.	96

Chapter 1

Introduction

1.1 The Structure of DNA and RNA

1.1.1 The Structure of DNA

Deoxyribonucleic acid (DNA) is a specific combination of *nucleotides* that encodes all the genetic information instructing the functioning and developing of all the living organisms. A single nucleotide consists of a phosphate group, a deoxyribose sugar and a nitrogenous base such that the nitrogenous base and the phosphate group covalently link to the 1' and the 5' carbon in the deoxyribose sugar, respectively. Depending on the nitrogenous bases, *adenine*, *guanine*, *cytosine* and *thymine*, there are four types of bases: *A*, *G*, *C* and *T*.

The consecutive nucleotides are linked to each other by the sugar-phosphate backbone between the 3'-hydroxyl group and the 5'-phosphate group, thus forming a single-stranded nucleotide strand. The conformation of the DNA strand leaves a 5'-phosphate group at one end of the strand and a 3'-hydroxyl group on the other end, inducing thus a direction of the DNA strand, from the 5' end to the 3' end. Since every DNA strand has an orientation, the strand 5'-ACCGT-3' is different from the strand 3'-ACCGT-5'.

DNA single strands can interact to each other due to hydrogen bonds that can form between nucleotides. The interactions between DNA nucleotides are highly specific, A

and T and respectively C and G being able to bind only to each other. This is known as the Watson-Crick complementarity. Two single-stranded DNA molecules that are complementary and of opposite orientation bind together and twist into a double helix structure by forming hydrogen bonds between corresponding complementary nucleotides in the opposite strands. For instance, the DNA strand 5'-CAGT-3' is complementary to the DNA strand 5'-ACTG-3' and Figure 1.1 shows the structure of a double-stranded DNA strand. A short DNA molecule which has not more than thirty bases is called an *oligonucleotide* [23]. The double-stranded DNA molecules will dehybridize, that is, the hydrogen bonds between nucleotides break apart and the double-stranded molecules break into single-stranded molecules, if the DNA molecules is heated. A temperature at which half of the oligonucleotides are hybridized and half of them are dehybridized is called *melting temperature*.

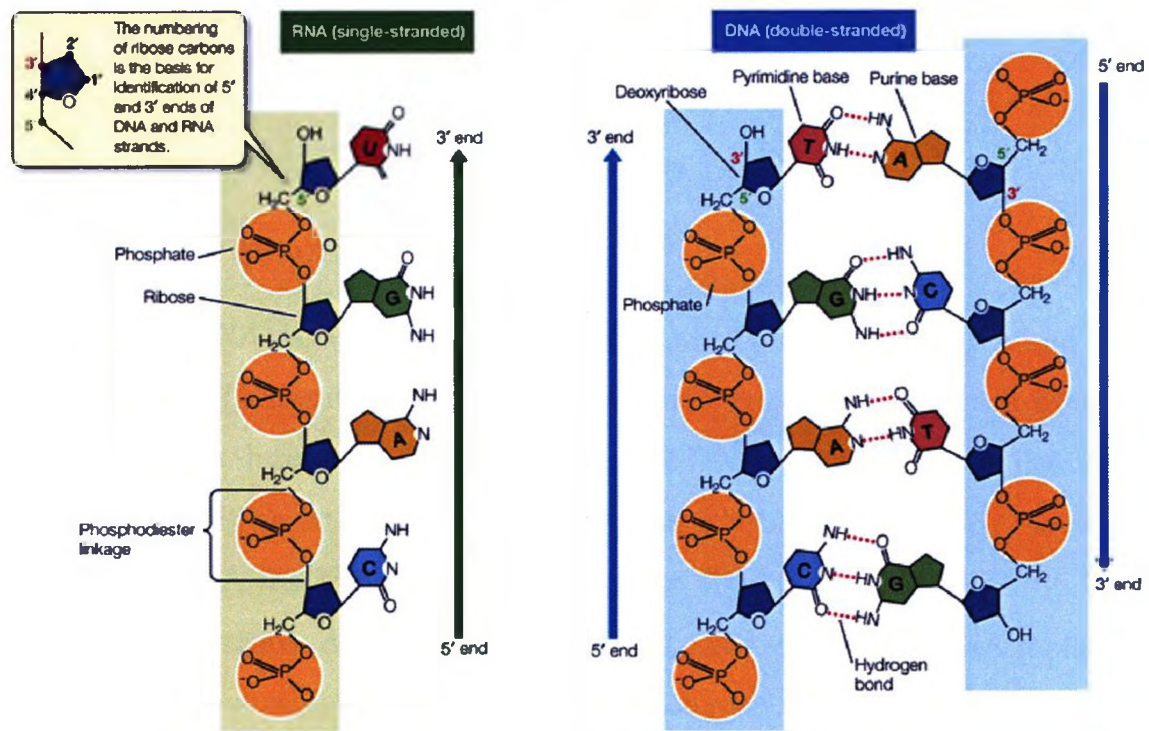


Figure 1.1: The structure of a double-stranded DNA and the structure of single-stranded RNA [38].

Due to the Watson-Crick complementarity, the hydrogen bonds of nucleotides can occur within a DNA strand resulting in many kinds of secondary structures. A single hairpin structure is a simple form of DNA secondary structure where two segments within a single strand are complementary to each other and thus bind together; an example is shown in Figure 1.2: the segment 5'-ACGTTTAGGT-3' binds to the segment 5'-ACCTAACGT-3'. The double-stranded segment in a hairpin is called a stem and the segment in between is called a loop, see Figure 1.2.

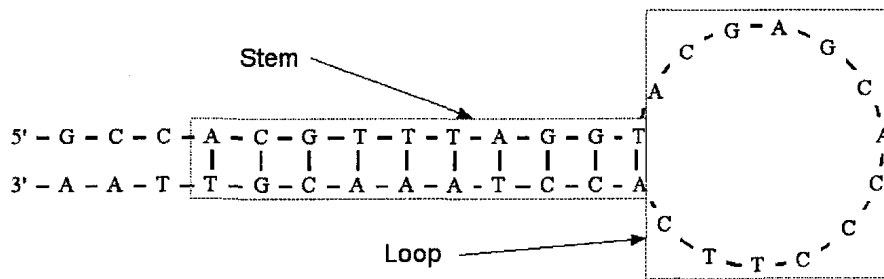


Figure 1.2: A DNA hairpin structure.

1.1.2 The Structure of RNA

Ribonucleic acid (RNA) is another kind of nucleic acid. Although both DNA and RNA are nucleic acids, RNA has its own specific characteristics. Concerning their chemical structures, there are three main differences between DNA and RNA molecules: RNA nucleotides have a different sugar group call *ribose*, RNA uses *uracil (U)* instead of thymine, and in RNA the nucleotide U may bind to G as well as to A.

Rather than the stable double helix structure of DNA molecules, RNA molecules are more often single-stranded. Segments within a long RNA strand may complement each other and the RNA strand may form many complex secondary structures such as pseudoknots, hairpin loops, bulge loops, internal loops and multiple loops.

1.2 The Objective of this Research

The Watson-Crick complementarity property has been intensively used in DNA computing research, both for implementation of DNA-based memories, and as a computational primitive used in bio-algorithms. This thesis falls within the first category by expanding on an idea proposed by Takahashi *et al.* who introduced a DNA hairpin-based RAM (Random Access Memory) in [44]. In the paper, data linked at the end of a four consecutive hairpin DNA structure can be accessed by fully opening all four hairpins one by one. We were intrigued by the idea of a RAM constructed by using multiple hairpins, and investigated a bottom-up approach to generate a set of sequences folding into such consecutive hairpin structures.

Our bottom-up approach is grounded in the “bond-free property” initially proposed by Kari *et al.* in [26] and extended by Cui and Konstantinidis in [10]. If a set of words satisfies the bond-free property, then any two subwords w and v of fixed length of words in the set have the property that the Hamming distance between w and the Watson-Crick complement of v is greater than a given positive integer d . The Hamming distance constraint, which will be explained in more details in Section 3.2.2, ensures that the DNA single strands represented by w and v are dissimilar to each other. In our design, constituent parts of a hairpin (stem, loop) are constructed by combining shorter “building blocks”, each of which belong to a set satisfying the bond-free property. The construction ensures that the DNA folds indeed into the expected hairpin and avoids other secondary structures. If several single hairpins are obtained, the multiple-hairpin structure is achieved by catenating the single hairpins together. To complement the design, combinatorial constraints such as the Hamming distance constraint, the GC content constraint and the continuity constraint are adopted to ensure the correctness and stability of the structures.

We develop a C++ program based on the bottom-up approach described above, to generate sequences that fold into the desirable multiple-hairpin structures used

in [44]. In order to check the correctness of our results, we use the Vienna RNA secondary structure package to predict the structure of our result sequences.

1.3 The Organization of the Thesis

This thesis is organized as follows. In Chapter 2, we give some basic definitions and notations: word, language, subword, prefix, suffix, antimorphic involution, Hamming distance, Hamming ball etc. We also include background information related to automata and tries which will be used later to construct distinct segments (stem and loop) of hairpins.

Chapter 3 consists of a literature review of DNA computing research and DNA word design problem. The DNA word design problem is one of the central topics of DNA computing research. This is because (unlike electronic data which has a fixed address) data-encoding DNA single strands can interact with each other in either programmed or undesirable ways. Various approaches to this problem are described in Chapter 3. In addition, we mention relevant research on DNA/RNA secondary structure design.

In Chapter 4, we describe the problem we address in this thesis and describe our bottom-up approach that constructs the constituent parts of a hairpin by combining shorter “blocks” chosen from sets satisfying the bond-free property. Bond-free languages have already been investigated in [26] and [10]. Here we improve the method of selecting bond-free “blocks”, we use the method to produce bond-free DNA word sets by constructing the corresponding tries, and modify the method to make sure the concatenation of bond-free DNA words still maintains the bond-free property. Several combinatorial constraints are then used to filter out undesirable words generated by the tries. Finally, we discuss issues related to the Vienna RNA secondary structure prediction package.

The main algorithms related to our bottom-up approach are presented in Chap-

ter 5. To test the performance and quality of our algorithm, we give experimental results in the same chapter. Also, the way to choose the parameters is discussed in the end of this chapter. Our result is that, by choosing suitable parameters, we successfully generated a set of sequences that can each form the desired consecutive hairpin structure. For example, our software can generate 120 sequences that each fold into a four-hairpin structure where the length of each hairpin stem is 20, the length of each hairpin loop is 7 and the length of the external segment is 20. We validate the correct secondary structure of our generated sequences by using the Vienna RNA package.

Finally, we present the conclusions of our approach and possible future directions of research in Chapter 6.

Chapter 2

Basic Notations and Background Information

A set of DNA sequences can be considered as a language over the DNA alphabet $\{A, C, G, T\}$. Therefore, in this chapter, we first give some formal definitions of language theory in Section 2.1. Furthermore, a formal definition of Hamming distance is given in Section 2.2. Finally, background information related to automata and tries is given in Section 2.3.

2.1 Preliminary Definitions

An *alphabet* Σ is a finite nonempty set of symbols and a *word* w is a sequence of symbols from the alphabet. The length of a word w is denoted by $|w|$. For example, the DNA alphabet is denoted by $\Sigma_{\text{DNA}} = \{A, C, G, T\}$ and the sequence AACGT is a word over the DNA alphabet; the binary alphabet is denoted by $\Sigma_{\text{binary}} = \{0, 1\}$ and the sequence 1110111 is a word over the binary alphabet. Due to the polarity of DNA molecules, by a DNA word w we denote the DNA molecule from the 5'-end to the -3' end. For instance, the DNA molecule 5'-AAGCGGT-3' is denoted by the word AAGCGGT and the DNA molecule 3'-AAGCGGT-5' is denoted by the word TGGCGAA. The word of length 0 is the *empty word* denoted by λ . By Σ^* , Σ^+ and Σ^k , we denote the set of all words, the set of all words except the empty word,

and the set of all words of length k over the alphabet Σ , respectively. A *language* L over the alphabet Σ is a subset of Σ^* and the cardinality of L is denoted by $|L|$.

The Watson-Crick complementarity can be considered as a mapping θ from Σ_{DNA} to Σ_{DNA} where $\theta(A) = T$, $\theta(T) = A$, $\theta(C) = G$, and $\theta(G) = C$. Following the definitions in [18], a *morphism* (*antimorphism*) of Σ^* is a mapping α from Σ^* to Σ^* satisfying $\alpha(uv) = \alpha(u)\alpha(v)$ ($\alpha(uv) = \alpha(v)\alpha(u)$) where $u, v \in \Sigma^*$. An *involution* $\theta: \Sigma \rightarrow \Sigma$ is a mapping with the property that, for every $w \in \Sigma^*$, we have $\theta(\theta(w)) = w$. Thus, the Watson-Crick complementarity of DNA sequences can be formalized mathematically by an antimorphic involution. For example, $\theta(\text{ACTG}) = \theta(G)\theta(T)\theta(C)\theta(A) = \text{CAGT}$.

Given a word $w \in \Sigma^*$, the *prefix* set of w is $\text{Pref}(w) = \{u \mid \exists v \in \Sigma^*, w = uv\}$. In a similar way, the *suffix* set of w is $\text{Suff}(w) = \{u \mid \exists v \in \Sigma^*, w = vu\}$ and the *subword* set of w is $\text{Sub}(w) = \{u \mid \exists x, v \in \Sigma^*, w = xuv\}$. By $\text{Pref}_k(w)$ and $\text{Suff}_k(w)$, we denote the prefix and suffix of w of length k , respectively. By $\text{Sub}_k(w)$, we denote the set of distinct subwords of w of length k . For example, given the word $w = \text{TCGTCGAA}$, $\text{Pref}_3(w) = \text{TCG}$, $\text{Suff}_3(w) = \text{GAA}$, and $\text{Sub}_3(w) = \{\text{TCG}, \text{CGT}, \text{GTC}, \text{CGA}, \text{GAA}\}$. Also, given a set S , $\text{Sub}_k(S)$, $\text{Pref}_k(S)$ and $\text{Suff}_k(S)$ denote the set of distinct subwords, prefixes and suffixes of length k of words in S , respectively. For example, if $S = \{\text{AAAA}, \text{ACGT}, \text{GCGC}\}$, then $\text{Sub}_2(S) = \{\text{AA}, \text{AC}, \text{CG}, \text{GT}, \text{GC}\}$.

2.2 Hamming Distance and Hamming Ball

The *Hamming Distance* was first introduced in the field of information theory to count the number of bits being changed during the transmission of information through a noisy channel [30]. It is a convenient method to measure the difference between sequences of equal length. Given two sequences $u = u_1u_2\dots u_n$ and $v = v_1v_2\dots v_n$ of length n , the Hamming distance $h(u, v)$ of u and v is the number of positions where the symbols are not the same, i.e., $u_i \neq v_i$, and $1 \leq i \leq n$. For instance, if $u = \text{AACGGT}$ and $v = \text{GGCGGT}$, then $h(u, v) = 2$ since these two sequences are

different at position one and two. The Hamming distance notion can be extended to sets. Given two sets of words of the same length, S_1 and S_2 , we say $h(S_1, S_2) > d$, if for any two words $u \in S_1$ and $v \in S_2$ we have $h(u, v) > d$.

The definition of a *Hamming ball* is based on the Hamming distance. For a word u , $H_d(u) = \{v \mid h(u, v) \leq d, \forall v \in \Sigma^k \text{ and } u \in \Sigma^k\}$ denotes a set of words of certain length k over an alphabet Σ such that the Hamming distance between any words v in the set and the word u is less than or equal to the distance d . For example, given the DNA alphabet $\Sigma_{\text{DNA}} = \{A, C, G, T\}$ and the word $u = AAA$, the Hamming ball $H_1(u)$ is $H_1(AAA) = \{AAA, CAA, GAA, TAA, ACA, AGA, ATA, AAC, AAG, AAT\}$. Indeed, each word in the Hamming ball is different in at most one position from the given word u . Likewise, $H_d(S)$ denotes the set of words such that each word w in this set satisfies $h(w, u) \leq d$ for all $u \in S$.

2.3 Automata Theory

A finite automaton is one of the simplest abstract models of computing devices. Formally, a *Deterministic Finite Automaton (DFA)* is a quintuple $M = (Q, \Sigma, \delta, s, F)$ consisting of five components:

- Q is a finite set of states,
- Σ is an input alphabet,
- $\delta : Q \times \Sigma \rightarrow Q$ is the set of transition rules,
- s is the initial state, and
- $F \subseteq Q$ is the set of final states.

Given a DFA M and an input word $w = w_1w_2\dots w_n$, $w_i \in \Sigma$, $1 \leq i \leq n$, the DFA reads the input word w as follows. Begin with the initial state s in the DFA and the first symbol w_1 of w . If there is a transition rule $\delta(s, w_1) = S_1$, then the automaton

transits to state S_1 and the current reading symbol becomes w_2 . The DFA will repeat this process until it finishes reading all the symbols of the word w or no transition rule can be applied. If the DFA is in a final state when it finishes reading the whole word w , then the word is accepted by the DFA, otherwise, it is rejected by the DFA. Here we present an example of how an automaton works.

Example 1 Consider a DFA $M = (Q, \Sigma, \delta, s, F)$ where $Q = \{s_0, s_1, s_2\}$, $\Sigma = \{A, C, T\}$, $s = s_0$, $F = \{s_2\}$, and the set of transition rules is:

$\{ \delta(s_0, A) = s_1, \delta(s_0, C) = s_0, \delta(s_0, T) = s_0, \delta(s_1, A) = s_0, \delta(s_1, C) = s_2, \delta(s_1, T) = s_0, \delta(s_2, A) = s_0, \delta(s_2, C) = s_0, \delta(s_2, T) = s_0 \}$.

The language L accepted by the DFA M , depicted in Figure 2.1, consists of all the words over the alphabet $\{A, C, T\}$ ending with AC . Given a word $w = ACTAC$,

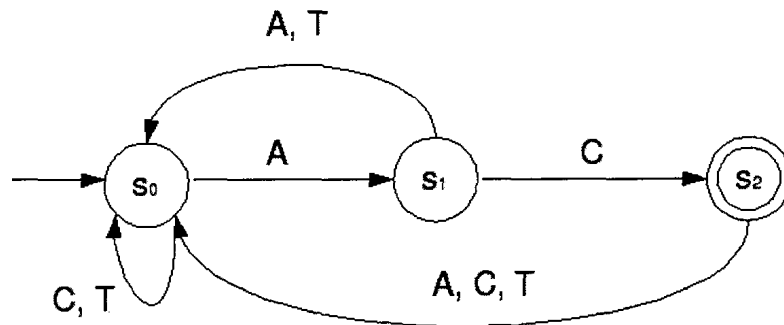


Figure 2.1: The DFA M in Example 1 accepts the language L over the alphabet $\{A, C, T\}$ consisting of all the words that are ending with AC .

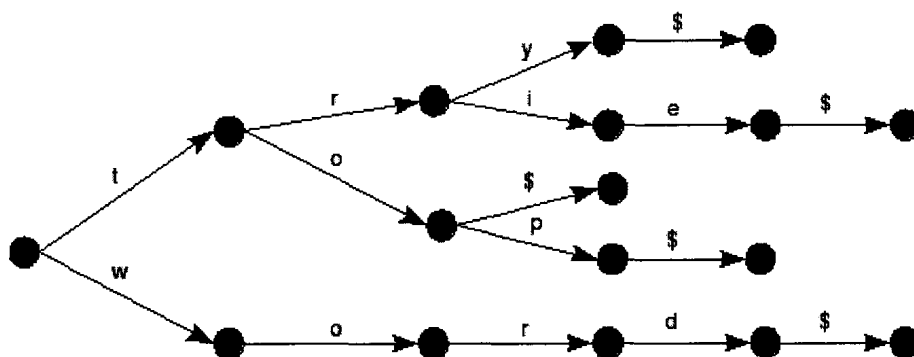
then w can be accepted by the “derivation”: $s_0ACTAC \vdash s_1CTAC \vdash s_2TAC \vdash s_0AC \vdash s_1C \vdash s_2$. In contrast, given the input word $w = ACAT$, w cannot be accepted since $s_0ACAT \vdash s_1CAT \vdash s_2AT \vdash s_0T \vdash s_0$, i.e., the automaton finishes reading the word and stops at the state s_0 which is not a final state.

The tree-like data structure *trie*, [2], is used to save a set of strings over an alphabet. Using this tree-like structure may reduce the space needed to store strings

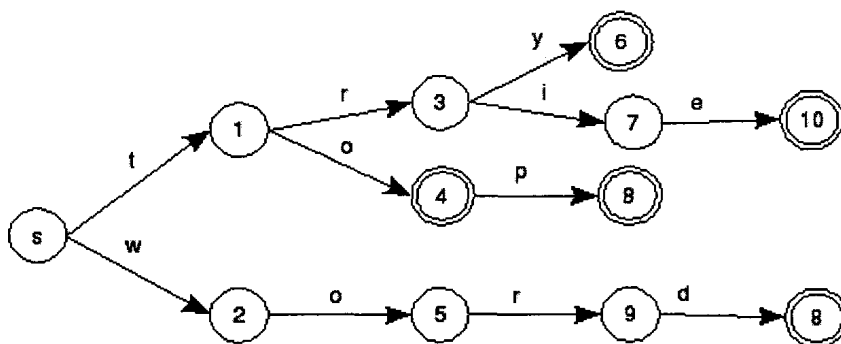
with a common prefix. Figure 2.2(a) gives an example of trie storing the set of strings $\{\text{try, trie, to, top, word}\}$. The strings *'try'*, *'trie'*, *'to'* and *'top'* have the common prefix *'t'*, therefore, they have a common ancestor node in the trie. A string in the given set can be recognized by searching the corresponding trie from the root to a leaf. For example, the string *'word'* can be recognized by going through the edges labeled *'w'*, *'o'*, *'r'*, and *'d'*. This may lead to the problem when a string in the set is the prefix of another word in the set, such as the string *'to'* being the prefix of the string *'top'*. To address this issue, a mark such as \$ was added to the end of each string.

A trie structure can also be considered as a DFA and the DFA corresponding to the trie mentioned above is shown in Figure 2.2(b), where the root corresponds to the start state and all the leaves correspond to terminal states.

In this thesis, we will construct tries that accept bond-free languages L with the property that any two subwords w, v of words in L of certain length satisfy $h(w, \theta(v)) > d$. The bond-free languages that we construct are closed under catenation.



(a) The trie structure represents the set of strings $S = \{\text{try, trie, to, top, word}\}$.



(b) The DFA over the alphabet $\{t, r, y, l, e, o, p, d\}$ accepts the word set $\{\text{try, trie, to, top, word}\}$.

Figure 2.2: A trie and an automaton representing the same set of words.

Chapter 3

Literature Review

3.1 DNA Computing

In DNA computing, computational problems are solved by applying molecular manipulation techniques to information-encoding DNA molecules. The first DNA computing experiment was carried out by Leonard Adleman [1] in 1994 to solve a seven-node instance of the Hamiltonian Path Problem.

The *Hamiltonian Path Problem (HPP)* determines whether there is a path that goes through each vertex exactly once in a given graph. HPP is an NP-complete problem¹ and Adleman has solved a 7-node instance of HPP by applying DNA molecular operations [23] to the graph encoded by DNA molecules. In Adleman's experiment, the Hamiltonian path sought starts with vertex v_{in} and ends with vertex v_{out} and the graph was encoded in the following way: each vertex v_i was encoded by a 20-base single-stranded DNA molecule, each edge e_k from vertex v_i to vertex v_j was encoded by a 20-base single-stranded DNA molecule such that $e_k = \theta(\text{Suff}_{10}(v_i)\text{Pref}_{10}(v_j))$ where θ is the antimorphic involution mentioned in Section 2.1.

As an illustration in Figure 3.1, if e is an edge from vertex u to vertex v in a given graph and vertices u and v are represented by single-stranded DNA molecules

5'-CGGTTACTTACTAACTTGGC-3' and 5'-GCGTATAGTACCGGAATCTC-3',

¹An NP-complete problem is a particular type of NP (non-deterministic polynomial time) problem with the property that all other NP problems can be reduced to it in polynomial time [23].

respectively, then edge e is represented by DNA molecule



Figure 3.1 also shows how these molecules bind to each other and form a partially-double-stranded structure.

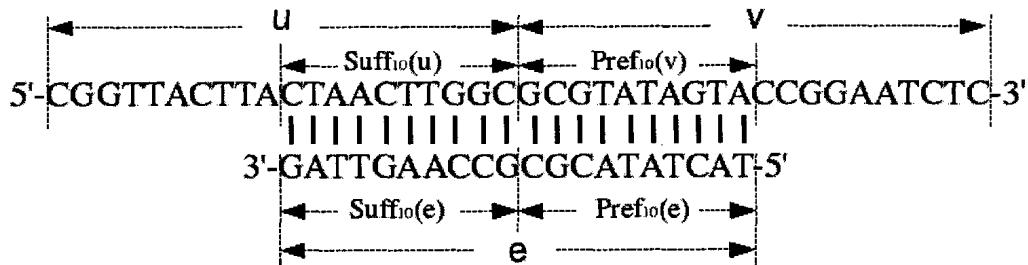


Figure 3.1: Single-stranded DNA molecules representing vertex u , vertex v and edge e from vertex u to vertex v that hybridize and form a partially-double-stranded DNA molecule.

In a solution test tube, each single-stranded DNA molecule representing a node or an edge was presented in multiple copies and they could hybridize to each other freely in random order to form distinct paths by the *ligation reaction*. The result of this step was that all possible paths in the graph were formed simultaneously. If there were any solutions of HPP in the graph, they could then be extracted by several molecular manipulations. The steps to weed out the incorrect candidate paths were as follows. First, keep the paths that begin and end with v_{in} and v_{out} , respectively. This step can be achieved by using *polymerase chain reaction (PCR)* to amplify the DNA strands starting with v_{in} and ending with v_{out} . Second, keep the paths that enter exactly n vertices where n is the total number of vertices in the graph. A technique called *gel electrophoresis* was used to separate the DNA strands by length, the length of the desirable path here being $140 = 20 \times 7$. Third, keep all the paths

that enter each vertex exactly once. By *affinity purification*, DNA strands containing certain subsequences can be extracted from heterogeneous solution of DNA strands. Therefore, the presence of vertices can be checked one by one by affinity purification. Finally, if there are paths remaining in the solution, they are the result of the HPP, otherwise, there is no solution path.

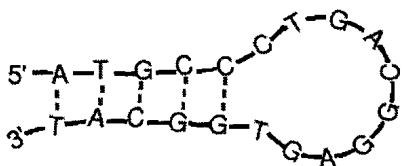
3.2 DNA Sequence Design

The correct solution to Adleman's experiment was based on the assumption that all the DNA sequences hybridize in the correct way. However, this is not always the case: DNA sequences can bind to themselves, to other strands inaccurately, or form some other undesirable bonds. Considering the DNA sequences used in Adleman's experiment as examples, the following hybridization may be unwelcome: If a segment in a sequence encoding a graph node is Watson-Crick complementary to another segment in the same sequence, then it may form a secondary structure such as Figure 3.2(a). If segments in different sequences representing the vertices or the edges are complementary to each other, then they may hybridize together and form double-stranded sequences such as Figure 3.2(b) and Figure 3.2(c). In both cases, the computation is compromised since the desired Hamiltonian path may never form.

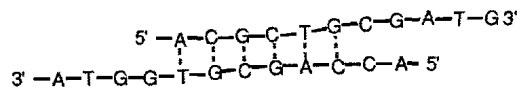
In this section, we present the DNA sequence design problem (Section 3.2.1) and some constraints that can be used to obtain DNA sequences that are free of undesirable hybridizations (Section 3.2.2).

3.2.1 The DNA Sequence Design Problem

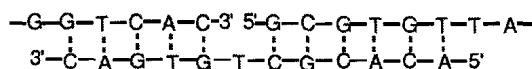
As seen in previous example, in DNA computing, obtaining the correct computing output is based on the occurrence of the precise hybridizations between DNA molecules, and no or few occurrences of unwanted hybridizations. The undesirable hybridizations within a set of DNA molecules interfere with the computing process.



(a) Undesirable bond of one DNA strand



(b) Undesirable bond between two strands



(c) Undesirable bond among DNA strands

Figure 3.2: Example of the undesirable hybridization.

Hence, the DNA sequence design problem arose and became a crucial problem in DNA computing.

The DNA sequence design problem [18] is: *How to encode a given problem by using single-stranded or double-stranded DNA that avoid most of the undesirable hybridizations and maintain the desirable hybridizations, so as to obtain the correct result by a succession of molecular operations?*

Many approaches have been proposed to tackle the DNA sequence design problem theoretically and experimentally. As DNA sequences can be thought of as words over the DNA alphabet, theoretical approaches have formalized the DNA sequence design problem as the “DNA word design problem”, and properties of DNA languages were investigated in [24], [20], that ensure that certain kinds of undesirable bonds would not occur. Others have estimated the upper and lower bound of the size of the DNA language with given constraints [32]. On the other hand, experimental approaches have generated DNA sequence libraries by using computer algorithms, and then constructing the DNA libraries in the laboratory. Mauri and Ferretti [33] give a survey of the methods related to DNA sequences design.

3.2.2 Constraints for DNA Sequences Design

To cope with the DNA word design problem, a necessary step is to eliminate the words that will form the undesirable bonds, as well as keep the words that will form the desirable bonds. Applying combinatorial constraints to the DNA words can reduce unwanted hybridizations. Therefore, various constraints were proposed to aid in the word design problem. A survey of constraints was given by [39] and Tulpan summarized these constraints in [46]. Here, we enumerate some of them.

- C1. *Hamming distance constraint:* This constraint is widely used in the word design problem, by which the words in the set may be dissimilar to each other, and then increase the correct hybridizations when apply molecule operations. A set of words S of length k is said to satisfy the Hamming distance constraint, if $h(u, v) > d, \forall u, v \in S$ where $k, d (k > d)$ are given positive integer parameters. For instance, the Hamming distance of the set $S = \{\text{ACGTTA}, \text{GGTGGC}, \text{CTACAT}\}$ is larger than 5 since the Hamming distance between any two words is 6.
- C2. *Reverse-complement constraint:* This is also a common constraint used in the word design problem, by which the words in a set may not hybridize to each other. A set of words S of length k satisfies the reverse-complement constraint, if $h(u, \theta(v)) > d, \forall u, v \in S$ and $k, d (k > d)$ are given positive integer parameters, where θ is the antimorphic involution mentioned in Section 2.1. Using the same example set S mentioned above, the reverse-complement distance of the set S is larger than 2 since the minimum distance are $h(\text{ACGTTA}, \theta(\text{CTACAT})) = h(\text{ACGTTA}, \text{ATGTAG}) = 3$ and $h(\text{CTACAT}, \theta(\text{ACGTTA})) = h(\text{CTACAT}, \text{TAACGT}) = 3$.
- C3. *Slide Hamming distance constraint:* If the length of words in the set is fairly long, undesirable bonds may occur within a word or between segments of two

words. To address this problem, the slide Hamming distance constraint was proposed, which makes sure that any slides of words of certain length are not similar to any other slides of words in the set.

- C4. *GC-content constraint:* Since the hybridization between G and C has three hydrogen bonds and there are only two hydrogen bonds when A binds to T, a double-stranded DNA will be more stable if the number of Gs and Cs in the strand is higher than a certain threshold. Furthermore, it is desirable to make the GC-content of all DNA words in certain range, so that the corresponding DNA molecules may have similar melting temperature.
- C5. *Continuity constraint:* If one base appears in one DNA strand continuously, the number of repetitions should be lower than a given threshold. This constraint prevents the unstable hybridizations of DNA strands [45].
- C6. *Alphabet size:* As mentioned in [34], if we use the alphabet set {A, C} instead of {A, C, G, T} for encoding information as DNA strands, this will prevent most undesirable bonds between words in the set. Also, the self-complementary DNA sequences in a DNA word set can be reduced if we use the three-base alphabet {A, C, T} [5]. Therefore, it would be a practical approach to use a subset of the DNA alphabet to design DNA words.

3.3 Approaches Related to DNA Word Design

Exhaustive search of the whole solution space of the DNA word design problem is not computationally feasible because the solution space grows exponentially with the length of the word. Many researchers proposed their own methods to deal with this word design problem, and we will describe some of them in this section.

3.3.1 The Stochastic Local Search Approach

The *stochastic² local search* algorithm is an efficient method that tackles problems with huge solution space. In [47], a stochastic local search approach was utilized to design sets of sequences with combinatorial constraints which include the GC content constraint (C4), the Hamming distance constraint (C1) and the reverse-complement Hamming distance constraint (C2). The algorithm starts with a random initial set of words of fixed length. The procedure iteratively picks up two words that violate the given constraints and modifies one symbol in either word which may let the new word satisfy the constraints. This is done until either the set of words satisfies all the constraints or until the iterations exceeds certain number of times.

3.3.2 The Unique Subsequence Approach

The idea of using unique short subsequences to generate longer DNA sequences with certain desirable properties was initially proposed by Seeman in [43] and [42] to construct *DNA junction* structures. In a junction structure, each sequence contributes to two double helices by binding to two other sequences within the structure. Figure 3.3 shows a well-known DNA junction called DNA Holliday junction.

In order to construct a stable and correct junction, Seeman applied particular rules to the short subsequences that he used as building blocks to create the final DNA strands. We emphasize three of them: First of all, subsequences of certain length could only be used at most once within the junction structure. Second, the Watson-Crick complement of any subsequence located at a bend could not be used inside the structure. Third, self-complementary subsequences were not allowed in any sequences. (A self-complementary sequence is a DNA sequence s satisfying $\theta(s) = s$.) The unique subsequence criteria made sure that the desirable hybridizations were

²A stochastic process is a process in which the current state does not depend on the previous states.

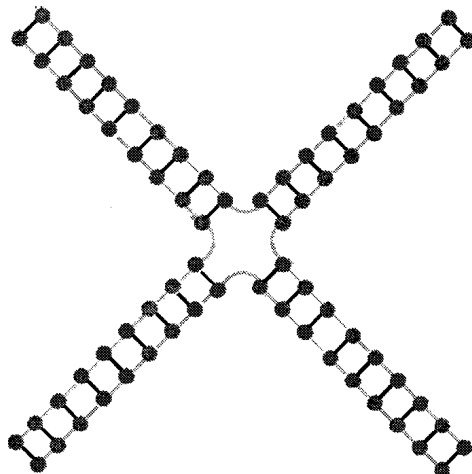


Figure 3.3: DNA Holliday junction.

more specific, and therefore more likely to form stable and correct junction structures.

The concept of “unique subsequences” was extended by Feldkamp *et al.* [13] to solve the DNA word design problem. Formally, they tried to design a set of words S of equal length n_s , with the property that any subwords of length $n_b < n_s$ of words in S can only be used at most once in S . By this criterion, the subwords of length n_b are *unique* in the set S , and the subwords of length $n_b - 1$ are the common subwords. The authors built then the set of sequences of length n_s by searching a tree structure. The tree structure had the following properties: Each vertex of the tree denoted a unique subsequence of length n_b . If there was an arrow from vertex u to vertex v , then $\text{Suff}_{n_b-1}(u) = \text{Pref}_{n_b-1}(v)$. For $n_b = 4$, Figure 3.4 shows a tree starting with vertex TAGC. Vertex TAGC would have arrows to vertices AGCA, AGCC, AGCG and AGCT, since TAGC has suffix AGC and the other subsequences have it as the prefix. However, the subsequence AGCT would be forbidden because it is self-complementary, that is $\theta(\text{AGCT}) = \text{AGCT}$. When searching the tree, each vertex can only be traversed once, and vertices representing the Watson-Crick complement of the respective subsequences being used cannot be used. In the Figure 3.4, the path shown in bold represents the sequence TAGCAC.

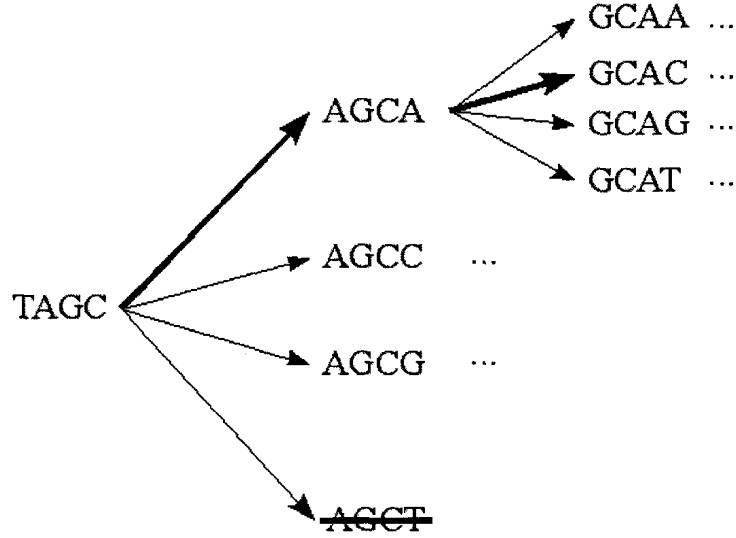


Figure 3.4: Tree structure used to build words of length n_s from shorter unique sequences of length $n_b = 4 < n_s$. There is an arrow from node u to node v iff $\text{Suff}_3(u) = \text{Pref}_3(v)$.

In [13], the cardinality of the word set obtained in this way was analyzed. If the length of the subsequence n_b is even, then the number of useful unique subsequences is

$$N_{\text{useful}}(n_b) = \frac{4^{n_b} - 4^{n_b/2}}{2}$$

and if n_b is odd, then

$$N_{\text{useful}}(n_b) = \frac{4^{n_b}}{2}$$

where 4^{n_b} is the total number of subsequences of length n_b and $4^{n_b/2}$ is the total number of the self-complementary subsequences of length n_b . Thus, an upper bound of the cardinality of a set of length n_s can be

$$N_{\text{seqs}}(n_s, n_b) = \lfloor \frac{N_{\text{useful}}(n_b)}{n_s - n_b - 1} \rfloor$$

When constructing the DNA sequence set, there is a trade-off between the diversity of the subsequences and the cardinality of the set. If n_b is small, the possible unique

subsequence candidates are very few, making the cardinality of the set of words of length n_s too small. For instance, if $n_b = 3$ and $n_s = 10$, then $N_{useful}(n_b) = 32$ and $N_{seqs}(n_s, n_b) = N_{seqs}(10, 3) = 5$. In this example, we may only generate 5 sequences of length 10, if we use the unique subsequence of length 3. On the other hand, if n_b is large, then the common subsequences of length $n_b - 1$ are also long, which may cause undesirable hybridizations. For example, if $n_b = 10$, then the length of the common subsequence is 9 which is long enough to allow undesirable bond within a subsequence or between the subsequences.

3.3.3 Word-Block Construction Scheme

Reif *et al.* constructed a DNA library of sequences of equal length for use in DNA computing experiments by concatenating shorter subsequences of equal length [36]. Here, a library is a collection of physical DNA molecules. The subsequences of equal length were assigned multiple times to design the DNA sequences rather than using the unique subsequences mentioned in the previous subsection. In this design method, the alphabet $\{A, C, T\}$ was used instead of the ordinary four-base DNA alphabet, which can reduce the chances of unwanted hybridizations.

This method consisted of several steps. Firstly, a set of words S of equal length over the three-base alphabet was generated, satisfying the GC-content constraint (C4). Here, Reif *et al.* chose the length of the sequences as 4. Since there was no symbol G in the alphabet, the authors fixed the number of bases C in each word w in the sets of DNA words. Secondly, the words generated in the previous step were assigned to distinct sets which were called “blocks” in [36], and all the blocks satisfied the Hamming distance constraint (C1): Each word w could only be assigned to one block, and the Hamming distance between any two blocks was larger than a given threshold. Finally, all the blocks were arranged in a particular order such that each DNA sequence used in the final DNA library was a concatenation of words selected orderly from each block. For example, if words were assigned to n blocks, and each block

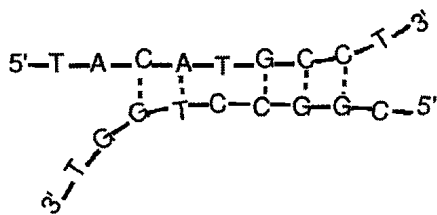
had m words, a word could be denoted by $w_{i,j}$ where i denotes the index of block that the word is in, and j denotes the index of the word in the block. Thus, a DNA sequence in the final library could be represented by $w_{1,m_1}w_{2,m_2}\dots w_{n,m_n}$. When the words were concatenated, new short words could be created at the adjacency places and they may cause undesirable bonds. Reif *et al.* implemented additional algorithms to handle this problem.

Compared to the unique subword approach, the Word-Block construction scheme can construct a much larger DNA library. If there are n blocks and m words in each block, then the cardinality of the DNA library S is $|S| = m^n$. For example, if $n = 10$ and $m = 4$, there will be 4^{10} sequences in the DNA library.

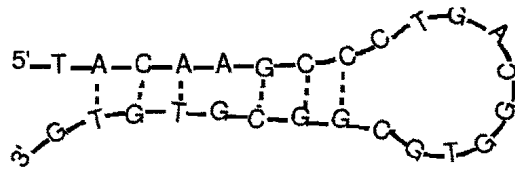
3.3.4 Method Using Bond-free Languages

Another issue related to the DNA word design problem is that undesirable bonds may occur even when sequences are only partially Watson-Crick complementary to each other. Figure 3.5(a) shows that sequence 5'-TACATGCCT-3' may hybridize to the sequence 5'-CGGCCTGGT-3' although bases at position 1, 2, 5 and 9 are mismatched. Figure 3.5(b) shows another example. The hybridization may occur between two segments 5'-ACAAGCC-3' and 5'-GGCGTGT-3' within a single-stranded sequence with one mismatch in the middle of the segments. According to this observation, it is therefore desirable to design DNA words based on the subwords with good properties, such as the bond-free property which will be mentioned in this subsection.

The notion of *bond-free language* was defined by [26] to deal with this type of undesired imperfect bond. A language L is a $(\theta, H_{d,k})$ -*bond-free language*, if any two subwords u and v of length k of words $w \in L$ satisfy the Hamming distance constraint $h(u, \theta(v)) > d$, where θ denotes an antimorphic involution (the mathematical formalization of Watson-Crick complementarity, defined in Section 2.1), k denotes the length of the subwords, and d denotes the Hamming distance. By choosing appropriate parameters d and k , any two subwords will not bind to each other. Thus



(a) Hybridization between two sequences.



(b) Hybridization within a sequence.

Figure 3.5: Hybridization of two segments with mismatches.

the bond-free language which is made up by words whose subwords of length k satisfy this constraint will avoid the imperfect segment hybridization situations shown in Figure 3.5. This is because, for an imperfect hybridization to occur between segments u and v , the matched bonds have to “outnumber” the mismatched bonds. By imposing $h(u, \theta(v)) > d$, we ensure that u and $\theta(v)$ differ by sufficiently many bases so that a hybridization is impossible.

To describe the relation between the subword set and the final DNA language, the *subword closure operation* \otimes was proposed in [26]. Given a set S which contains only the words of length k , the *subword closure* S^\otimes of S is a language which contains all the words of length larger than or equal to k , such that for each word w in the language S^\otimes , $\text{Sub}_k(w) \subseteq S$. For example, given $S = \{\text{AAT}, \text{ATG}, \text{TGC}, \text{GCA}\}$, word $w = \text{AATGC} \in S^\otimes$ since all the subwords AAT, ATG and TGC are from S , but the word $w = \text{AATTG} \notin S^\otimes$ since the subwords ATT and TTG are not in the set S .

Generally, we are seeking a DNA language consisting of words of equal length, say l . Also, since concatenating information-encoding DNA strands is often needed during biocomputation, a desirable property of a DNA language is that it be closed under concatenation. Using the $(\theta, H_{d,k})$ -bond-free language S^\otimes as an example, if the language S^\otimes is not closed under concatenation, then for $w, v \in S^\otimes$, some of the words of length k in $\text{Sub}_k(wv)$ might not be in S , and a segment may hybridize to the segment at the joint of w and v . Keeping the focus on these two considerations, Cui and Konstantinidis [10] proposed a method to generate a bond-free language of words

of fixed length that is closed under concatenation. This method will be mentioned later in Section 4.2 and the details of the method are described in [9].

3.3.5 DNA Word Design Software

Various software packages related to DNA word design were proposed, such as design a set of words for a specific DNA computing algorithm, design a set of words that satisfies certain constraints, and design a set of words that may fold into a specific structure.

The unique subsequence approach proposed in [13] and described in Section 3.3.2 was implemented as a software package called *DNASequenceGenerator*. The authors also considered other criteria such as the GC content constraint, and melting temperature, when constructing the sequences. An extension of this work, *DNASequence-Compiler* [12] was introduced to study and construct sequences with the property that the concatenation of sequences still maintain the “unique subsequences” property. A component of the software *Nucleic Acid Computing Simulation Toolkit (NACST)* called *NACST/Seq* [29] introduced a non-dominated sorting genetic algorithm(NSGA) which aimed at generating a good set of sequences meeting multiple constraints. Based on the theoretical properties of DNA words studied by Hussini *et al.* [18], and Jonoska *et al.* [20], *CODEGEN* [28] was implemented to generate a DNA word set with certain properties and to test whether a given set of DNA words satisfies these properties.

Another main approach to DNA sequences design is used in DNA nanotechnology. *Uniquimer* [49] was a software with graphic interfaces to design DNA sequences for DNA self-assembly. Iimura *et al.* [19] implemented a system to construct specific 4×4 tiles. Not only did they use the GC content constraint and the melting temperature constraint to design the sequences, they also evaluated the sequences based on the free energy. Free energy is a widely used criterium in the sequence design problem. If a DNA sequence can theoretically fold into several possible secondary structures,

in practice it will fold as to minimize a thermodynamic parameter called “Gibbs free energy”. In the system, a hill-climbing algorithm was adopted to maximize the free energy of sequences that did not need to hybridize, and minimize the free energy of sequences that were expected to hybridize.

3.4 Hairpin Secondary Structure Design

In the following, we first introduce the DNA/RNA folding problem (Given a DNA/RNA sequence, what is the secondary structure that it will form?) and some of its solutions, and then we describe the inverse folding problem (Given a secondary structure, find a RNA sequence that will fold into that structure.) in Section 3.4.1. Several approaches to solve the inverse folding problem are introduced in Section 3.4.2. Finally, in Section 3.4.3, we describe the formal definition of a single hairpin structure and the hairpin frame.

3.4.1 Secondary Structure Design Problem

An unstable single-stranded RNA or DNA sequence will hybridize to itself if two of its segments are Watson-Crick complementary to each other and form a more stable structure. A complex pseudoknot-free³ secondary structure may contain many kinds of simple secondary structures, for example, hairpin loop, multiple loop, internal loop, bulge, and external bases, see Figure 3.6. RNA molecules are more often single stranded and unstable single-stranded RNA sequences may fold into complex secondary structures to reduce their free energy. This raised the RNA secondary structure prediction problem which also known as the folding problem.

Mfold [50] and *RNAfold* in *Vienna RNA Package* [17] were proposed by Zuker and Hofacker *et al.*, respectively, to predict the secondary structure of a given DNA/RNA sequence. The main method was to fold the given sequence into a structure with

³Pseudoknot is a specific RNA structure [37] that we do not consider in this thesis.

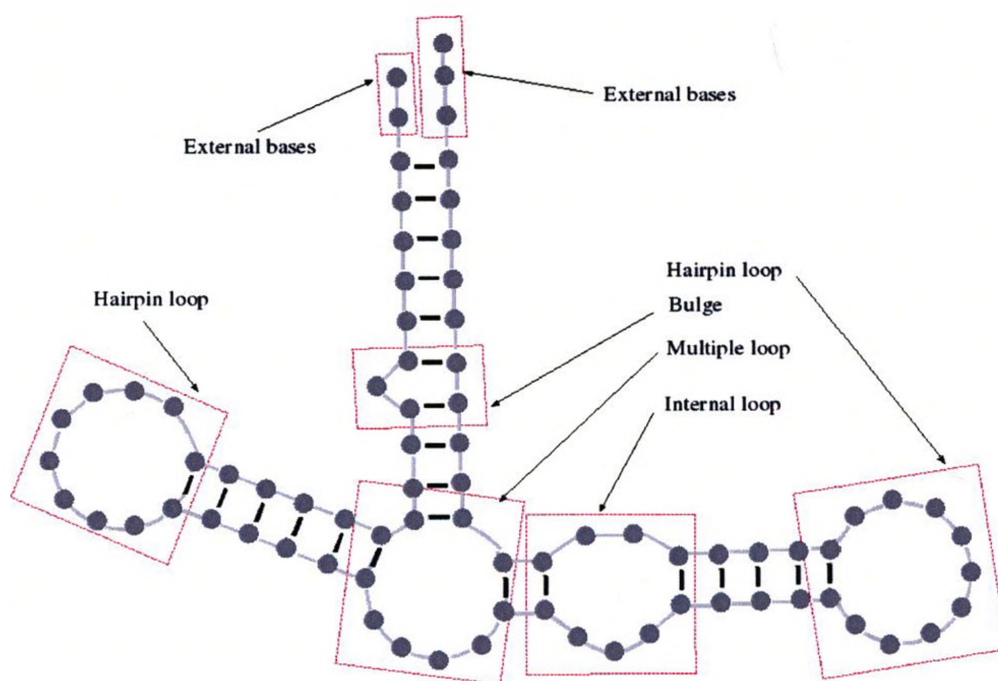


Figure 3.6: An example of secondary structure.

minimum free energy, since a structure with lower free energy will be more stable and immobile. To address this problem, dynamic programming was implemented to assign substructures to the given sequence to optimize the free energy. The process was based on the assumption that the energy of the whole structure was built up by the energy of small substructures [17] such as stems, hairpin loops, bulges, internal loops, multiple loops, external bases, etc. The thermodynamic parameters of small substructures were collected in experiments at a temperature of 37°C. Regarding the DNA word design problem, these two software packages can be used to predict whether given DNA words will form any secondary structures.

On the other hand, the research in self assembly and the interest in the molecular machine design gave rise to the inverse folding problem. Given a secondary structure, the problem is to find a RNA sequence that will fold into the target structure. Several approaches have been presented and there are some common techniques adopted by

these approaches [17], [3], and [6]. First of all, minimum free energy is used as a criterion to evaluate the sequences. If a sequence has the lowest free energy value when it folds into a given structure, then this sequence is the best sequence. Secondly, the whole solution space of the problem is exponential relative to the length of the given structure, so it is difficult and unpractical to check every solution sequence. With this in mind, the stochastic local search is a general technique used to find a solution. Also, the energy of the whole structure is added up by the energy of the substructures [17] and it is easier and more efficient to optimize the substructures and then combine them together to build the final structure, so most of the algorithms try to design sequences of substructures and then link them together. In the following section, we will introduce some of these approaches to solve the inverse folding problem.

3.4.2 Methods Related to Structure Design Problem

The inverse RNA folding problem was investigated in [17] and a heuristic algorithm was implemented as a function called *RNAinverse* in the *Vienna RNA package*. Given a complicated secondary structure, *RNAinverse* began the process with an initial sequence and iteratively optimized the substructures by mutating an unpaired base or a base pair. Since the sequence is measured by its free energy, if a mutation reduces the free energy of the substructure, then the mutation will be maintained, otherwise the mutation will be discarded. Afterwards, all the optimal subsequences are linked to form the objective secondary structure.

In [3], the author introduced *RNA Secondary Structure Designer (RNA-SSD)*. Although the basic method was analogous to *RNAinverse*, *RNA-SSD* put more effort into designing the initial sequence. Three observations were applied to the initial sequence. Generally, the original sequence was assigned according to the unpaired bases and the paired bases. If a base would hybridize to another base in the given structure, then the pair was assigned at the same time. Furthermore, a certain GC content was imposed on the paired bases to make the structure more stable.

Conversely, if a base was an unpaired base such as a base at the loop structure, then the base was assigned to make sure it would not bind to other base in the loop. Finally, the unique subsequence concept mentioned in [42] was also applied to the initial sequence.

INFO-RNA, introduced in [6], is a new approach to solve the folding problem, based on two steps. In the initial step, dynamic programming was used to calculate the minimum free energy of bases folding into the given complex structure. This step can generate a sequence with minimum free energy based on the specific structure, but the sequence can also fold into other structures which have even lower free energy. In order to obtain a better sequence, *INFO-RNA* introduced stochastic local searching algorithm in the second step. In this step, they obtained the sequence by mutating the initial sequence by one base if it is an unpaired base or two bases if they are a base pair.

3.4.3 The Hairpin Secondary Structure

A single hairpin structure is a simple form of DNA (or RNA) secondary structure where two segments within a single strand are complementary to each other and thus binding together; an example is shown in Figure 1.2. The authors in [27] proposed a formal definition of all the hairpin with stem length at least k , where k is a positive integer:

$$hp(\theta, k) = \{ xvy\theta(v)z \mid x, z, v \in \Sigma^*, y \in \Sigma^+, \text{ and } |v| \geq k \}.$$

In a hairpin $hp(\theta, k)$, v and $\theta(v)$ are complementary to each other and they are of length at least k . Take Figure 1.2 as an example: $v = \text{ACGTTTAGGT}$ will bind to $\theta(v) = \text{ACCTAAACGT}$, since they are Watson-Crick complementary to each other.

To model multiple-hairpin structures, in [25], a hairpin frame was defined by:

$$u = x_1v_1y_1\theta(v_1)z_1x_2v_2y_2\theta(v_2)z_2\dots x_nv_ny_n\theta(v_n)z_n.$$

where $x_i, v_i, y_i, \theta(v_i), z_i \in \Sigma^*$. For each segment $x_i v_i y_i \theta(v_i) z_i$, v_i is complementary to $\theta(v_i)$ and form a single hairpin. Therefore, u has n single hairpins and u is a *hp-frame* of degree n . Note that other formal definitions of hairpin structures and hairpin frames have been proposed in the literature, for example by using trajectories [11].

3.5 Motivation of the Thesis

Using the above notations, we can denote by

$$u = x v_1 y_1 \theta(v_1) v_2 y_2 \theta(v_2) v_3 y_3 \theta(v_3) v_4 y_4 \theta(v_4) \quad (3.1)$$

a four-consecutive-equal-size-hairpin structure where $x, v_i, y_i, \theta(v_i) \in \Sigma_{DNA}^+$, $|x| =$

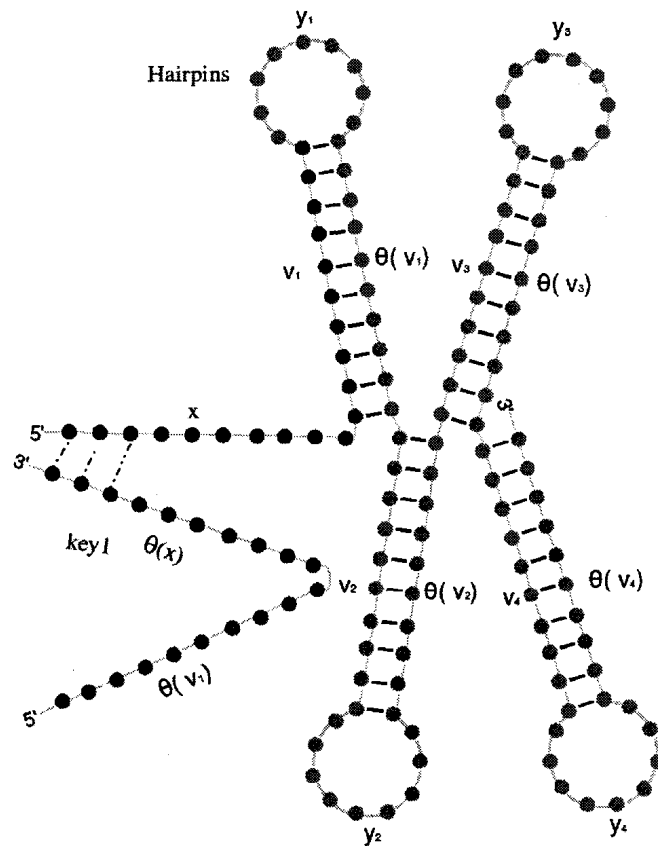


Figure 3.7: A Multi-state DNA Machine.

$|v_i| = |\theta(v_i)| = s$ and $|y_i| = l$ ($1 \leq i \leq 4$). This structure was designed and used in [48] to serve as a multi-state machine, see Figure 3.7.

For a single hairpin structure, the folded and unfolded structure can be considered as two different states. Hence, a consecutive hairpin structure, i.e., a structure containing several consecutive equal-size hairpins, can work as a multi-state machine. In order to make the hairpins unfold, additional single stranded DNA sequences are used as the *keys* to open each hairpin structure. A key can open the hairpin by branch migration, a process by which an unpaired region of a strand displaces one strand from a double-helix by forming the same base pairs as the complement of strand being replaced, provided the new structure which is more energetically favourable.

The process of opening all four hairpins of u is as follows: The key $\theta(v_1)\theta(x)$ is used to open the first hairpin $xv_1y_1\theta(v_1)$. Here the key $\theta(v_1)\theta(x)$ is a linear DNA sequence whose first part, $\theta(v_1)$, is complementary to the stem portion v_1 in the first hairpin, and the second part, $\theta(x)$, is complementary to the external segment x . Figure 3.8 shows how the key $\theta(v_1)\theta(x)$ opens the first hairpin $xv_1y_1\theta(v_1)$. The part $\theta(x)$ of the key will first anneal to the external segment x , see Figure 3.8(a). Thus, by branch migration, the part $\theta(v_1)$ of the key will hybridize to the stem v_1 and force the double-stranded stem to dehybridize into two single-stranded segments. Finally, the first hairpin is fully opened, see Figure 3.8(b).

To open the second hairpin, the key $\theta(v_2)v_1$ is used and the process is similar to the previous one, see Figure 3.9. The opening of the first hairpin makes the segment $\theta(v_1)$ single stranded, therefore, it behaves similarly to the external segment x . The part v_1 of the key will anneal to the segment $\theta(v_1)$ of the stem, see Figure 3.9(a), and then force the double stranded stem to dehybridize into two single stranded segments. At the same time, the part $\theta(v_2)$ of the key will hybridize to the stem v_2 . Thus, the second hairpin is opened, see Figure 3.9(b).

By the same token, the third and fourth hairpin can be opened orderly by applying keys $\theta(v_3)v_2$ and $\theta(v_4)v_3$. After applying all four keys, the multiple-hairpin structure

Figure 3.9: Opening the second hairpin.

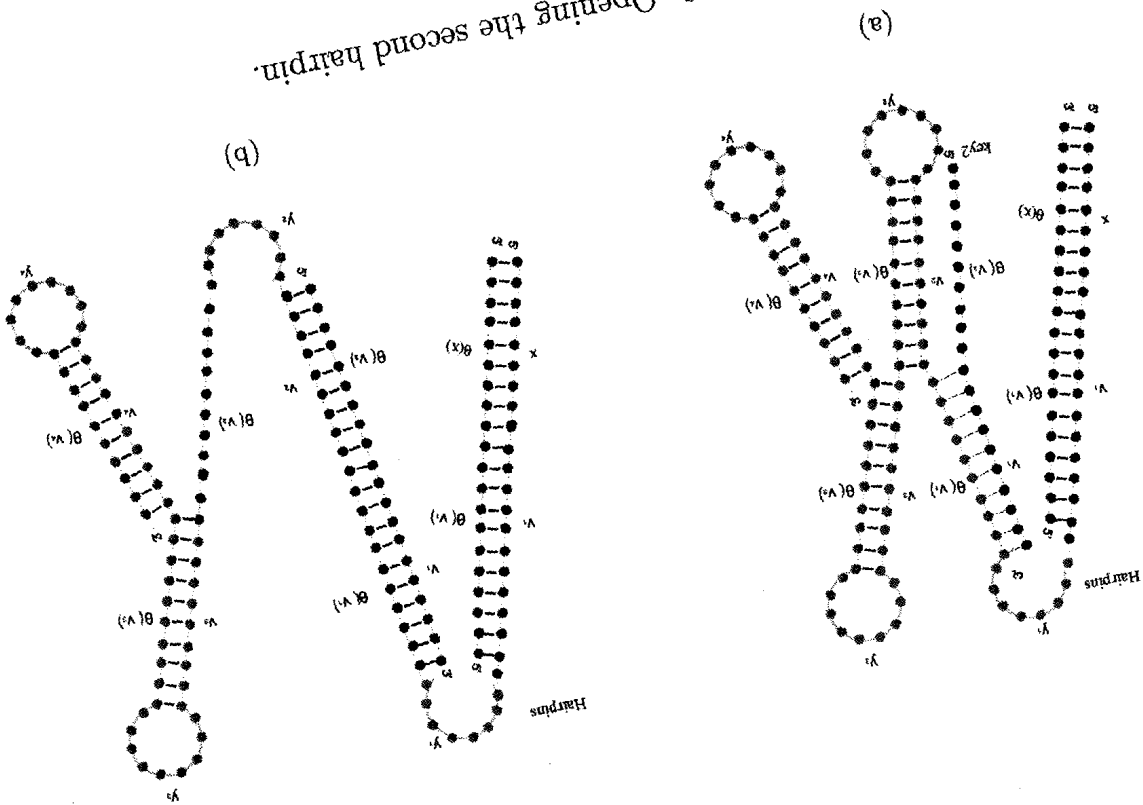
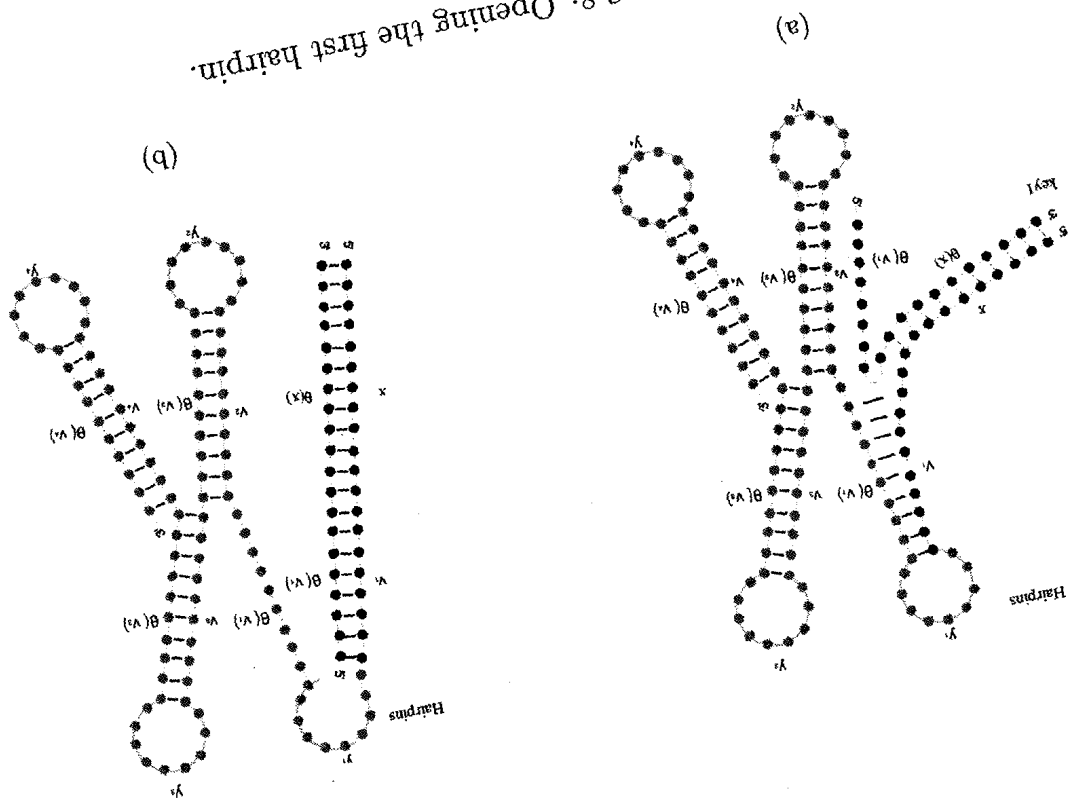


Figure 3.8: Opening the first hairpin.



is fully opened and Figure 3.10 shows this result.

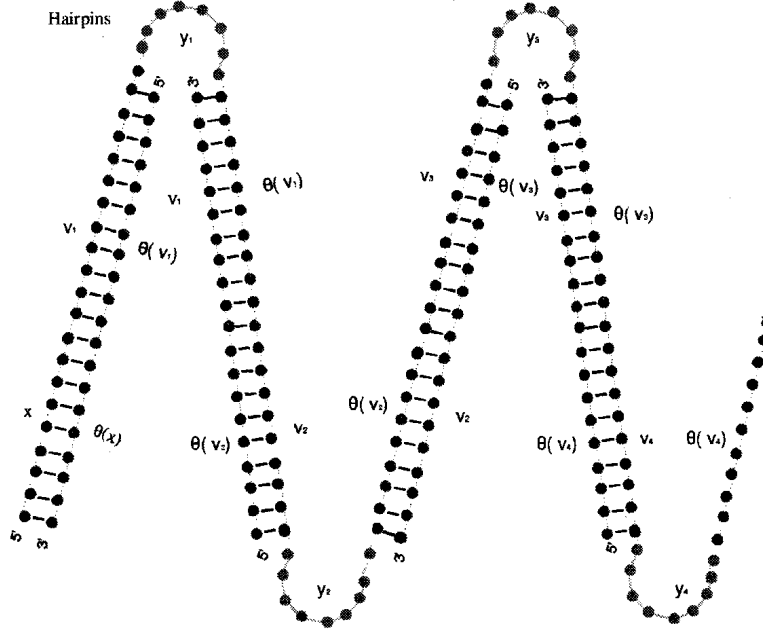


Figure 3.10: The four hairpin structure is fully opened after applying four distinct keys.

Using the methods proposed in [48], the authors in [22] succeeded to effectively construct a DNA sequence with four consecutive hairpins. Moreover, in [44], two applications of the multi-state machine were proposed. The multi-state machine can work as an address section of the memory and each hairpin can represent a bit. The data linked to the address can be reached by opening all the hairpins sequentially. It is clear that a hairpin in the structure (except the first hairpin) can be opened only if the hairpin on its left is open, since the key needs to hybridize to a single stranded segment for the branch migration. Therefore, by this conformation, a distinct address can be obtained by just changing the order of the single hairpins. If the structure is changed by introducing “spacers” x_2, x_3, x_4 between hairpins:

$$u' = x_1 v_1 y_1 \theta(v_1) x_2 v_2 y_2 \theta(v_2) x_3 v_3 y_3 \theta(v_3) x_4 v_4 y_4 \theta(v_4),$$

where $x_i, v_i, y_i, \theta(v_i) \in \Sigma_{DNA}^+$, $|x_i| = |v_i| = |\theta(v_i)| = s$ and $|y_i| = l$ ($1 \leq i \leq 4$). In

the new conformation, each single hairpin $x_i v_i y_i \theta(v_i)$ has a segment x_i to which the corresponding key may hybridize to allow branch migration. Thus, the hairpins can be opened independently by applying the corresponding key. By this new conformation, the machine can be used to solve the Maximum Independent Set Problem (MISP) [44].

There are many other applications of the hairpin structures. For example, in [4], the authors propose a “smart drug” wherein the main computation component is single hairpin structure with a sticky end. The stem of the hairpin encodes the “diagnostic rules” and the loop encodes the “drug” to be released.

Furthermore, Sakamoto *et al.* [40] used *Whiplash PCR (WPCR)* to solve some NP-complete problems such as *CNF-SAT*, the vertex cover problem, the direct sum cover problem and HPP. The basic idea is to use WPCR to simulate a state transition machine by iteratively forming hairpin structures. The machine consists of a single stranded DNA molecule of the form

$$5' - \text{stopper} \theta(s'_1) \theta(s_1) \dots \text{stopper} \theta(s'_m) \theta(s_m) \dots \text{stopper} \theta(s'_n) \theta(s_n) s_j s_k \dots s_m - 3'$$

where state transition rules $\theta(s'_i) \theta(s_i)$ are encoded at the 5' end separated by *stopper*

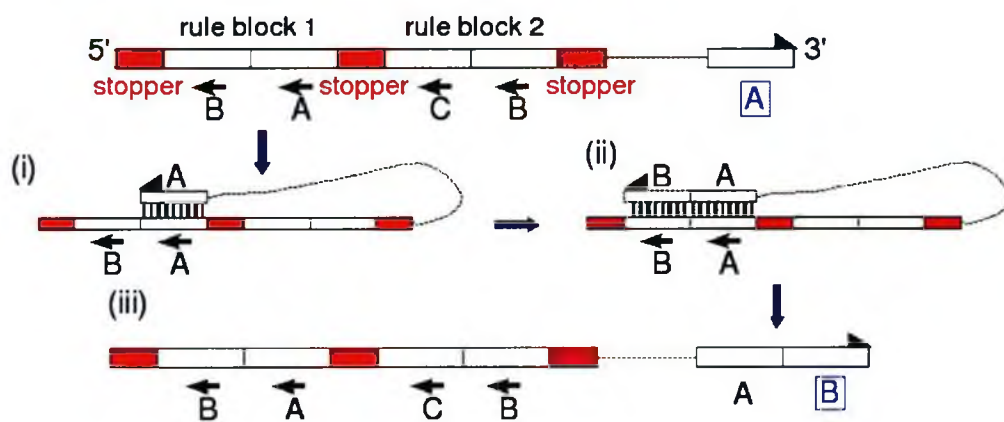


Figure 3.11: Whiplash PCR simulates a state transition machine that transits from state A to state B by transition rule $A \rightarrow B$.

sequences and the current state s_m is encoded at the 3' end of the sequence. Figure 3.11 gives an example. If the current state is A and there is a transition rule $A \rightarrow B$ (encoded as $\theta(B)\theta(A)$ and presented as $\overleftarrow{B}\overleftarrow{A}$ in Figure 3.11), then the process of state transition is as follows: First, the current state A hybridizes to the state $\theta(A)$ in a transition rule $\theta(B)\theta(A)$ and forms a hairpin structure. Second, by PCR, nucleotides which form the DNA strand B are added to the 3' end of the sequence. Therefore, the machine transits from state A to state B . Third, the hairpin structure is unfolded and becomes a single stranded sequence.

Chapter 4

DNA Multiple-Hairpin Structure Design

As described previously in Section 3.4.2, various methods were used to design DNA sequences that form given structures. The primary approach was using stochastic local search to find one solution sequence that will fold into the given structure. Here we want to investigate the possibility to generate a set of sequences which will fold into the particular multiple-hairpin structure described in [48]. Our approach is mainly based on the bond-free property [26].

In this chapter, we first state the problem of this thesis, and then we give a general idea of our bottom-up approach in Section 4.1. In Section 4.2, we describe a method to generate a bond-free language $B(l)$ of words of equal length l closed under concatenation, that is, the reverse-complement Hamming distance between any two subwords w and v of length k of words in $(B(l))^*$ is larger than d . In Section 4.3, we describe a method to generate two bond-free languages $B_{S_1}(s)$ and $B_{S_2}(l)$ such that $(B_{S_1}(s))^*$, $(B_{S_2}(l))^*$ and the concatenation $B_{S_1}(s) \cdot B_{S_2}(l)$ are also bond-free languages. Furthermore, in Section 4.4, we describe a filtration procedure by which we can weed out the undesirable sequences in the sets $B_{S_1}(s)$ and $B_{S_2}(l)$ and obtain the sets B_{stem} and B_{loop} , respectively. In Section 4.5, we explain the way we arrange the sequences in B_{stem} and B_{loop} to construct the multiple-hairpin structure, and give an estimation for the number of the sequences that can be constructed. Finally, in Section 4.6, we

introduce the Vienna RNA Package by which we can predict the structures of the result sequences of our bottom-up approach.

4.1 Problem Statement and Solution

In order to clarify the problem we want to solve, we give a formal statement of the consecutive hairpin structure design problem in Section 4.1.1. and a further discussion of the problem is given in the same section. Thus, we describe the bottom-up approach to solve this problem in Section 4.1.2.

4.1.1 Problem Statement

The DNA multiple-hairpin structure design problem can be described as follows.

Problem Statement: *Given a multiple-hairpin structure,*

$$hps(n, s, l, \theta) = \{xv_1y_1\theta(v_1)\dots v_ny_n\theta(v_n) \mid x, v_i, \theta(v_i) \in \Sigma^s, y_i \in \Sigma^l \text{ and } 1 \leq i \leq n\} \quad (4.1)$$

where θ is an antimorphic involution, can we find a set of DNA sequences that will fold into this particular structure correctly? Furthermore, for given parameters n , s and l , what is the cardinality of the set of sequences that form this particular structure?

Figure 4.1 shows a multiple-hairpin structure in $hps(2, 13, 7, \theta)$ which contains an external segment and 2 hairpin structures. In the remainder of this thesis, for each multiple-hairpin structure, the external segment x will denote the single-stranded segment at the 5' end; for each single hairpin structure, the loop y_i and the stem v_i and $\theta(v_i)$ are the single-stranded segment and the double-stranded segments, respectively. An example of the external segment, loop and stem are highlighted in Figure 4.1.

As mentioned above, Figure 4.1 shows a correct multiple-hairpin structure $hps(2, 13, 7, \theta)$. Based on the observation of this multiple-hairpin structure, we emphasize

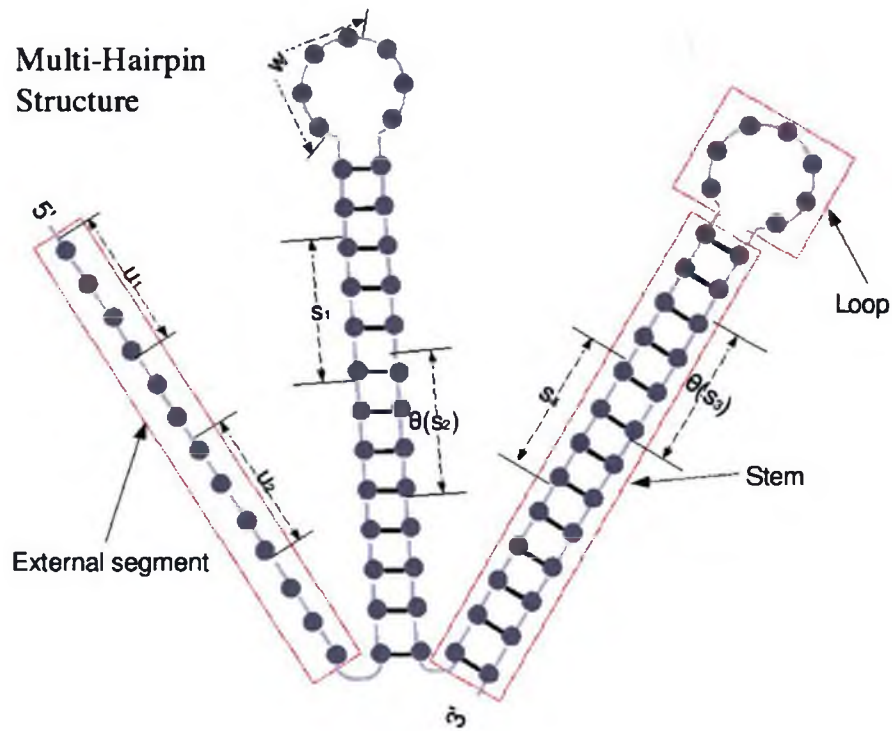


Figure 4.1: Correct multiple-hairpin structure $hps(2, 13, 7, \theta)$.

four kinds of fundamental unwelcome bonds related to the structure:

- U1: Any two segments of continuously unpaired bases bind to each other. For example, in Figure 4.1 such undesirable bonds would include u_1 binding to u_2 within the external segment, or u_1 in the external segment binding to w in the first loop.
- U2: Any segments of continuously unpaired bases binds to any segments of stems v_i . This may include two situations: (a) Any segments of unpaired bases in the external segment binds to any segments of any stems v_i and any two segments in the stems v_i and v_j bind to each other. (b) For each single hairpin structure $v_i y_i \theta(v_i)$, any segments in the loop y_i binds to any segments in the stem v_i . For example, in Figure 4.1 such undesirable bonds would include u_1 from the external segment binding to s_1 from v_1 of the first stem, s_1 from v_1 of the first

stem binding to s_4 from v_2 of the second stem (a), or w from the first loop binding to s_1 from v_1 of the first stem (b).

U3: For each single hairpin structure $v_i y_i \theta(v_i)$, any segments of unpaired bases in the loop y_i binds to any segments of the stem $\theta(v_i)$. For example, in Figure 4.1, such undesirable bonds include w from the loop binding to $\theta(s_2)$ from $\theta(v_1)$ of the first stem.

U4: Any segments of the external segment binds to any segment of stems $\theta(v_i)$ and any segments of stems v_i binds to any segments of stems $\theta(v_j)$. For example, in Figure 4.1, such undesirable bonds include u_1 or u_2 from the external segment binding to $\theta(s_2)$ from $\theta(v_1)$ of the first stem, or s_1 from v_1 of the first stem binding to $\theta(s_3)$ from $\theta(v_2)$ of the second stem.

Before describing in detail the bottom-up method we propose, we give some observations on the multiple-hairpin structure. Obviously, the sequences of a stem, both v_i and $\theta(v_i)$, should be considered together, i.e., if either the sequence v_i or $\theta(v_i)$ is fixed, the other sequence can be obtained by Watson-Crick complementarity. Therefore, in the thesis, we will only consider the design of sequences v_i of the stem and obtain the complement by using the antimorphic involution.

4.1.2 Overview of Our Bottom-up Approach

Many approaches proposed to design a sequence that may fold into a specific structure are based on the evaluation of the free energy. In contrast to these approaches, we investigate a bottom-up approach to design sequences that may fold into a consecutive hairpin structure based only on the combinatorial constraints, especially the bond-free property.

It is straightforward to observe that the problem is difficult to solve by exhaustive search. Instead, our approach adapts the idea of Seeman *et al.* [43] that uses small

pieces of single-stranded DNA sequences with good properties to construct the DNA sequences which will form the desirable complex secondary structure. A drawback of Seeman's method is that the size of the structure is very dependent on the cardinality of the subword set. As noted in Section 3.3.2, the unique subsequence approach can only generate 5 DNA words of length 10 if the length of the subword is 3. Instead of using each subword word at most once, we attempt to use the subwords with good properties multiple times to construct the multiple-hairpin secondary structure.

Since U1 - U4 represent the undesirable bonds between small segments in the structure, in the remainder of this paper, we call a subsequence of length k in the structure a *block*. In Figure 4.1, for example, segments $u_1, u_2, s_1, w, \theta(s_2), \theta(s_3)$ and s_4 are blocks of length $k = 4$. To avoid the undesirable bonds U1 - U4 observed, we employ both the reverse-complement Hamming distance constraint (C2) and the Hamming distance constraint (C1). The undesirable bonds U1 and U2 can be reduced by applying the reverse-complement Hamming distance constraint to any two blocks u and v in the multiple-hairpin structure, i.e., $h(u, \theta(v)) > d_1$. Furthermore, if we apply the Hamming distance constraint to any two blocks u and v in the loop y_i and the stem v_i , respectively, that is $h(u, v) > d_2$, then the undesirable bond U3 can be reduced. A similar approach deals with U4, in fact, since the length of the stems and the external segment are identical, we will generate the corresponding blocks together. Due to this consideration, we will drop U4 and instead apply the Hamming distance constraint only to the whole sequences of the stems v_i and the external segment x .

We now present our new bottom-up approach of constructing DNA sequences for the multiple-hairpin structure.

Initially, a set of words S of length k is constructed such that $h(u, \theta(v)) > d_1$ and $u, v \in S, k > d_1$, see Figure 4.2 a). We want all the blocks in the structure to be from the set S . By choosing proper parameters k and d_1 , we will generate a proper set S and prevent the undesirable bonds U1 and U2.

In the second step, two sets S_1 and S_2 are constructed such that $S_1 \subseteq S, S_2 \subseteq S$

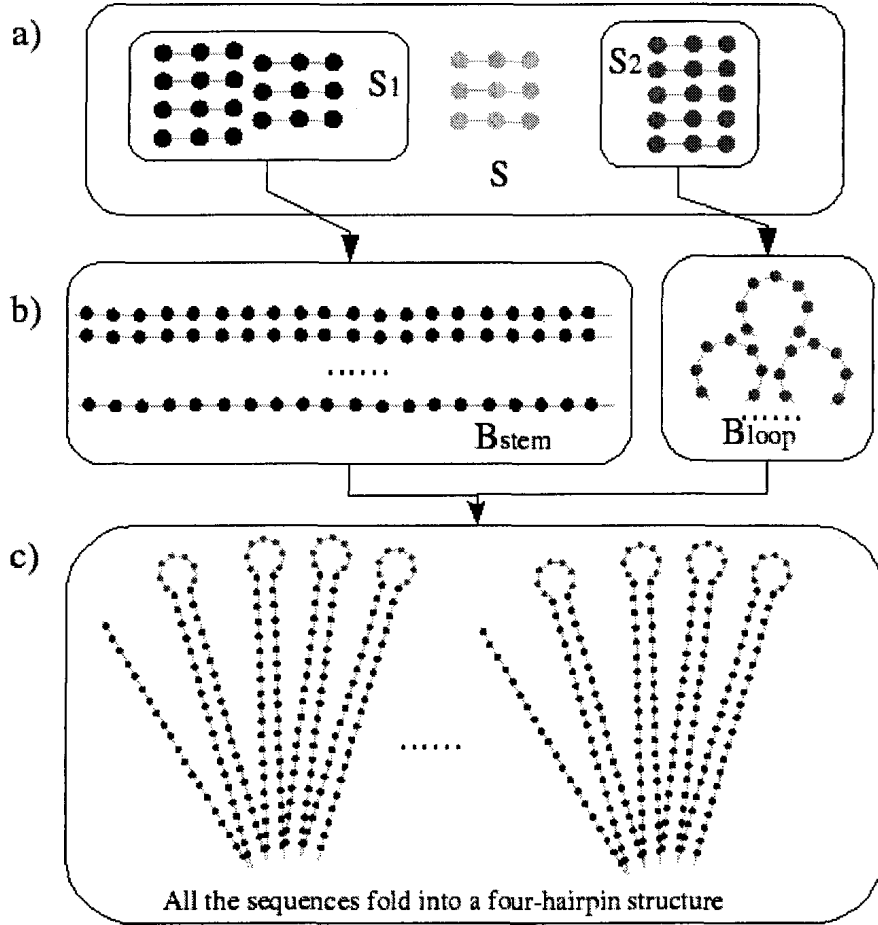


Figure 4.2: The bottom-up approach using to design the multiple-hairpin structure.

and $S_1 \cap S_2 = \emptyset$, as well as $H(S_1, S_2) > d_2$, see Figure 4.2 a). The blocks in the stems and the loops will be from the sets S_1 and S_2 , respectively. By choosing a proper parameter d_2 , the blocks in stems will be distinct from those in the loops. Therefore, we can prevent the undesirable bond U3 by which the blocks in the loop bind to blocks in $\theta(v_i)$ of the stem.

In the third step, a set of words B_{stem} of length s and a set of words B_{loop} of length l are constructed that all the blocks in B_{stem} and B_{loop} are from S_1 and S_2 , respectively, see Figure 4.2 b). Furthermore, the blocks in $(B_{stem})^*$, $(B_{loop})^*$ and $B_{stem} \cdot B_{loop}$ should all be in S . By selecting a sequence v_i from B_{stem} and a sequence y_i from B_{loop} , a single

hairpin structure can be obtained by concatenating v_i to y_i and then to $\theta(v_i)$. Since the length of the stem and external segment are equal to s , the external segment x can also be selected from B_{stem} . In order to avoid U4 and other undesirable sequences, before obtaining the sets B_{stem} and B_{loop} , we apply the combinatorial constraints to the stem set $B_{s_1}(s)$ of words of length s and the loop set $B_{B_2}(l)$ of words of length l to weed out undesirable sequences.

Finally, the external segment and all the single hairpin structures will be concatenated together to form the multiple-hairpin structure, see Figure 4.2 c).

While the general idea above is simple, many details related to the implementation of this method need to be addressed. In the following sections, we will mention several major problems and our proposed solutions.

4.2 Generate the Bond-free Language

One of the main problem in the previous algorithm is how to construct B_{stem} and B_{loop} starting with the blocks from S_1 and S_2 , respectively. We will use for this purpose a method (Section 4.2.1) that, given a set S of words of length k , outputs a set S^\otimes of words of length l that is a $(\theta, H_{d,k})$ -bond-free language. This method will be used to generate the set S_1^\otimes of words of length s , $S_1^\otimes(s)$, and the set S_2^\otimes of words of length l , $S_2^\otimes(l)$, such that $B_{stem} \subset S_1^\otimes(s)$ and $B_{loop} \subset S_2^\otimes(l)$.

In Section 4.2.2 we propose a refined method to select the set S by which we can generate a fair large set $S^\otimes(l)$. In order to evaluate the cardinality of $S^\otimes(l)$ without generating all the words in $S^\otimes(l)$, we describe the method of calculating the cardinality of $S^\otimes(l)$ in Section 4.2.3.

Since the sets B_{stem} and B_{loop} are concatenated to form the multiple-hairpin structure, we describe the method to construct the set $B(l) \subset S^\otimes(l)$ of words of certain length l which is closed under concatenation in Section 4.2.4. Furthermore, the method of calculating the cardinality of the set $B(l)$ is provided in the same section.

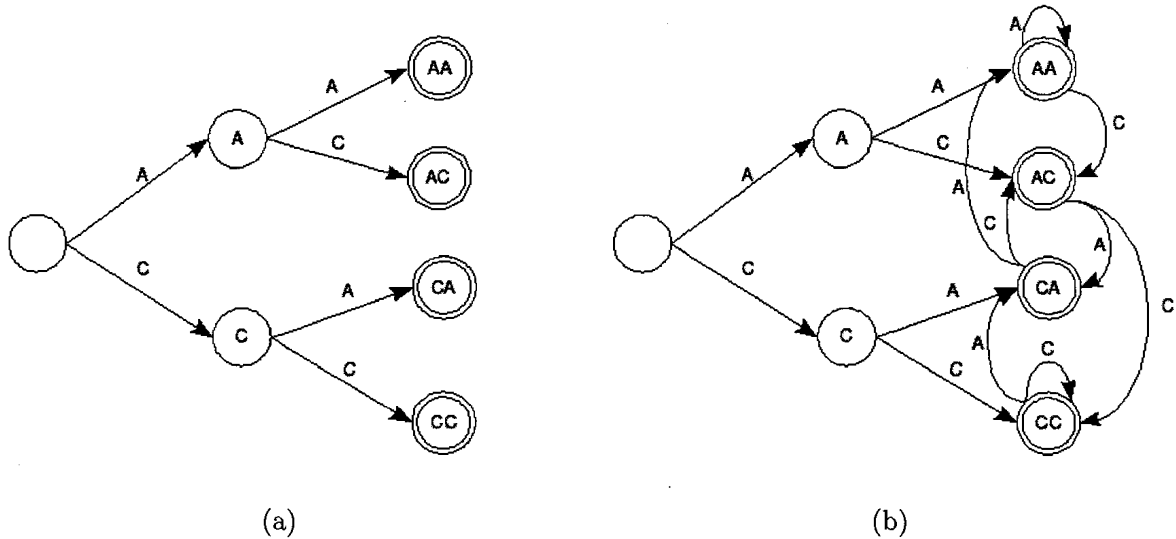


Figure 4.3: Trie constructed based on set $S = \{AA, AC, CA, CC\}$.

4.2.1 Generate the Bond-free Language S^\otimes from a Set S

As mentioned in Section 3.3.4, a language L is $(\theta, H_{d,k})$ -bond-free if any two subwords u and v of length k of words in L satisfy $h(u, \theta(v)) > d$. The authors of [26] gave a method to construct a bond-free language starting from a set of words of equal length. Given a set S of words of length k , then S^\otimes is $(\theta, H_{d,k})$ -bond-free if and only if it satisfies

$$\theta(S) \cap H_d(S) = \emptyset \quad (4.2)$$

where θ is the antimorphic involution, \otimes is the subword closure operation mentioned in Section 3.3.4. Equation (4.2) implies that $h(u, \theta(v)) > d$, $|u| = |v| = k$ for all $u, v \in S$.

Given a set of words S that meets the constraint of Equation (4.2), the bond-free language S^\otimes can be obtained by searching the corresponding automaton, [9], constructed based on the set S .

The automaton accepting S^\otimes is mainly a trie, defined as in Section 2.3. The way to construct the automaton that accepts S^\otimes is as follows: Initially, build a trie according to the set S and set all the leaves as the final states. Here, a path from

the root to a leaf corresponds to a word in the set S . For instance, given the set $S = \{AA, AC, CA, CC\}$ satisfying $\theta(S) \cap H_1(S) = \emptyset$, the trie corresponding to S is shown in Figure 4.3(a). Secondly, iteratively add an edge from leaf u to leaf v if $\text{Suff}_{k-1}(u) = \text{Pref}_{k-1}(v)$ and label it with the last symbol in leaf v . For the second step, we can find eight pairs of leaves (u, v) satisfying $\text{Suff}_{k-1}(u) = \text{Pref}_{k-1}(v)$ in the trie in Figure 4.3(a) and all the pairs (u, v) are (AA, AA) , (AA, AC) , (AC, CA) , (AC, CC) , (CA, AA) , (CA, AC) , (CC, CA) and (CC, CC) . Then we can add eight new edges and the result is shown in Figure 4.3(b).

To obtain a word of length $l \geq k$ from this automaton, we simply follow a path from the start state to a final state which goes through l edges.

Grail [35] introduced by Raymond and Wood is a C++ library to construct and manipulate automata. In [9], Cui implemented an algorithm to construct the trie mentioned above by applying functions in *Grail*. We included this algorithm from Cui's thesis to construct tries, needed in our construction.

4.2.2 The Selection of the Set S

The main problem described here is to find a set S of words of certain length k that satisfies the equation (4.2). Given parameters k and d which are the length of the words and the reverse-complement Hamming distance between any two words, respectively, we need to find a subset S of words over the whole solution space of size 4^k , such that $\theta(S) \cap H_d(S) = \emptyset$. Although the way to select a good word set S is still not completely solved, Equation (4.2) gives us a clue: whenever a word w is selected into the set S , the reverse-complement of words located in the Hamming ball $H_d(w)$ centred at w should not be in the set S . Cui, [9], adopted this idea to heuristically obtain the set S . The process picked a word w in the solution space, and removed all the words u that satisfied $h(\theta(w), u) \leq d$ in the solution space. If given an initial word w_{initial} , all the words were sorted in the alphabetical order, the basic step could be applied recursively from the word w_{initial} to the last word, and from the first word

to the word before word w_{initial} . After this procedure, the remaining words were the words in S .

Using the method mentioned above, we can produce a good set of words. Here are some observations about the set S generated by the method. According to the experimental data from [9], the words in the set S are dependent on the given start word and the order in which we check the solution space. One way to improve the result is to change the way we check the solution space.

By observation, we have got the following relations between S^\otimes and S . If the word w is in S^\otimes , then $\text{Sub}_k(w) \subseteq S$. Also, a word $w \in S^\otimes$ can be achieved by the overlap of words in the set S . For example, given the set $S = \{\text{AA}, \text{AC}, \text{CA}, \text{CC}, \text{CT}, \text{TC}\}$, the word $w = \text{ACACTC} \in S^\otimes$ is achieved by the consecutive overlaps of words AC, CA, AC, CT and TC in S , see Figure 4.4. If there are no overlaps of words in the set S which cannot be used to construct longer words, then S^\otimes is only a finite set. For example, if $S = \{\text{ACC}, \text{GGT}, \text{GAT}\}$, then $S^\otimes = S$. Based on these observations, if we check the words in the solution space according to the overlapping property rather than the alphabetical order, the result set S has better properties since the cardinality of subword closure of S , $|S^\otimes|$, is larger than the previous experimental data in [9]. The experiment data of the new method will be shown in Chapter 5.

A C A C T C
 A C
 C A
 A C
 C T
 T C

Figure 4.4: The word $\text{ACACTC} \in S^\otimes$ is constructed by the overlaps of words AC, CA, AC, CT, and TC in S .

4.2.3 Calculate the Cardinality of $S^\otimes(l)$

Following [10], we use $S^\otimes(l)$ to denote all the words of length l in the set S^\otimes . The cardinality of the set S^\otimes of words of a certain length l , $|S^\otimes(l)|$, is one of the fundamental properties according to which we evaluate the set S^\otimes . If it is too small, it is difficult to construct a suitable DNA word set. On the other hand, if the cardinality $|S^\otimes(l)|$ is fairly large, we can select a subset of $S^\otimes(l)$ with good properties by applying the combinatorial constraints such as the Hamming distance constraint (C1), the GC content constraint (C4), the continuity constraint (C5) and etc. Another reason to calculate the cardinality of $S^\otimes(l)$ will be mentioned in the next subsection.

The cardinality $|S^\otimes(l)|$ can be calculated by searching the whole trie, but the process is very time consuming. We adopt the recursion algorithm from [9] to calculate the cardinality of $S^\otimes(l)$ without generating the set $S^\otimes(l)$. Given a set of words S of length k , a word $w \in S^\otimes(l)$ can be presented in the form $w = a_1a_2\dots a_{l-n}a_{l-n+1}\dots a_{l-1}a_l$ ending with the suffix $su = \text{Suff}_n(w) = a_{l-n+1}\dots a_{l-1}a_l$ of length $n = k - 1$. Here, two situations are considered: First, the length of the words in $S^\otimes(l)$ is longer than the length of the words in S , i.e., $l > k$. If we consider all the words of length l ending with a particular suffix $su = v \cdot e = a_{l-n+1}\dots a_{l-1}a_l$ of length $n = k - 1$ ($v \in \Sigma^*$, $e \in \Sigma$), then these words can only come from the set of words $S^\otimes(l-1)$ of length $l-1$ ending with the suffixes $su' = b \cdot v = a_{l-n}a_{l-n+1}\dots a_{l-2}a_{l-1}$ ($v \in \Sigma^*$, $b \in \Sigma$) such that $\text{Suff}_{n-1}(su') = \text{Pref}_{n-1}(su) = v = a_{l-n+1}\dots a_{l-2}a_{l-1}$ and $b \cdot v \cdot e \in S$, see Figure 4.5. Since we consider the suffixes of length $n = k - 1$, and all the subwords of length k of words $w \in S^\otimes(l)$ should be in S , then the overlap of $su' = b \cdot v$ and $su = v \cdot e$ should be in S , that is, $b \cdot v \cdot e \in S$. Second, if the length of the words in $S^\otimes(l)$ is equal to the length of words in S , i.e., $l = k$, then the words in $S^\otimes(l)$ ending with a particular suffix su of length $n = k - 1$ are the words in S ending with the suffix su .

Let us consider the set of words of length $k = 3$ and $n = 2$, $S = \{\text{AGG, GGG, TGG, GAG, GAT, GGA, GGT, TGT, GTA, GTG, TTG, GTT, TTT}\}$, as an example. All the distinct suffixes su of length $n = 2$ are GG, AG, AT, GA, GT, TA, TG,

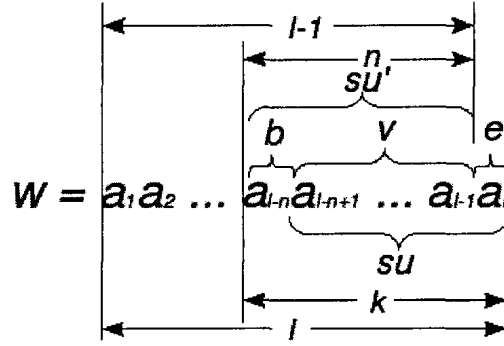


Figure 4.5: The words w of length l end with the suffix $su = v \cdot e$ of length $n = k - 1$ and the words of length $l - 1$ end with suffix $su' = b \cdot v$ of length $n = k - 1$ where $\text{Pref}_{n-1}(su) = \text{Suff}_{n-1}(su') = v$ and $b \cdot v \cdot e \in S$.

and TT. Let $S_{su}^\otimes(l)$ denote the set of words of length l ending with suffix su . Then we may obtain the set of words of length l ($l > k$) ending with suffix su as follows:

$$\begin{aligned}
S_{GG}^\otimes(l) &= [S_{AG}^\otimes(l-1) \cdot G] \cup [S_{GG}^\otimes(l-1) \cdot G] \cup [S_{TG}^\otimes(l-1) \cdot G], \\
S_{AG}^\otimes(l) &= S_{GA}^\otimes(l-1) \cdot G, \\
S_{AT}^\otimes(l) &= S_{GA}^\otimes(l-1) \cdot T, \\
S_{GA}^\otimes(l) &= S_{GG}^\otimes(l-1) \cdot A, \\
S_{GT}^\otimes(l) &= [S_{GG}^\otimes(l-1) \cdot T] \cup [S_{TG}^\otimes(l-1) \cdot T], \\
S_{TA}^\otimes(l) &= S_{GT}^\otimes(l-1) \cdot A, \\
S_{TG}^\otimes(l) &= [S_{GT}^\otimes(l-1) \cdot G] \cup [S_{TT}^\otimes(l-1) \cdot G], \text{ and} \\
S_{TT}^\otimes(l) &= [S_{GT}^\otimes(l-1) \cdot T] \cup [S_{TT}^\otimes(l-1) \cdot T].
\end{aligned}$$

Thus, the set of words of length l ($l = k$) ending with suffix su are as follows:

$$\begin{aligned}
S_{GG}^\otimes(l) &= \{\text{AGG}, \text{GGG}, \text{TGG}\}, \\
S_{AG}^\otimes(l) &= \{\text{GAG}\}, \\
S_{AT}^\otimes(l) &= \{\text{GAT}\}, \\
S_{GA}^\otimes(l) &= \{\text{GGA}\}, \\
S_{GT}^\otimes(l) &= \{\text{GGT}, \text{TGT}\},
\end{aligned}$$

$$\begin{aligned}
S_{TA}^{\otimes}(l) &= \{\text{GTA}\}, \\
S_{TG}^{\otimes}(l) &= \{\text{GTG}, \text{TTG}\}, \text{ and} \\
S_{TT}^{\otimes}(l) &= \{\text{GTT}, \text{TTT}\}.
\end{aligned}$$

If we want to produce the set of words of length $l = 4$ ending with suffix $su = \text{GG}$, $S_{GG}^{\otimes}(l)$, then we may need three sets of words of length $l = 3$ ending with suffixes AG , GG , and TG , respectively. We can consider these three suffixes as su' such that $\text{Suff}_2(su') = \text{Pref}_2(su) = v = \text{G}$ and the overlap of su' and su is the word in S , that is, AGG , GGG and TGG , respectively. Therefore, $S_{GG}^{\otimes}(4) = [S_{AG}^{\otimes}(3) \cdot G] \cup [S_{GG}^{\otimes}(3) \cdot G] \cup [S_{TG}^{\otimes}(3) \cdot G] = [\{\text{GAG}\} \cdot G] \cup [\{\text{AGG}, \text{GGG}, \text{TGG}\} \cdot G] \cup [\{\text{GTG}, \text{TTG}\} \cdot G] = \{\text{GAGG}, \text{AGGG}, \text{GGGG}, \text{TGGG}, \text{GTGG}, \text{TTGG}\}$.

Therefore, given a set of words S of equal length k , the set S^{\otimes} of length l ($l > k$) ending with a particular suffix $su = v \cdot e$ of length $n = k - 1$ is

$$S_{su=v \cdot e}^{\otimes}(l) = \bigcup_{su' \in E} S_{su'=b \cdot v}^{\otimes}(l-1) \cdot e,$$

where set E is the set of all the distinct suffixes $su' = b \cdot v$ of length $n = k - 1$ in S such that $\text{Pref}_{n-1}(su) = \text{Suff}_{n-1}(su') = v$ and the overlap of su' and su are in S , i.e., $b \cdot v \cdot e \in S$, and $b, e \in \Sigma, v \in \Sigma^*$.

Let us consider the second case, if $l = k$, then

$$S_{su}^{\otimes}(l) = S_{su}$$

where S_{su} is the set of words in S that end with su . Furthermore, S^{\otimes} of length l is

$$S^{\otimes}(l) = \bigcup_{su \in E_{su}} S_{su}^{\otimes}(l),$$

where set E_{su} is the set of all the distinct suffixes of length n in S .

As a result, we can calculate the cardinality of $S^{\otimes}(l)$ by the following method: First, we find out set $E_{su} = \text{Suff}_{k-1}(S)$ that contains all distinct suffixes su of length $n = k - 1$ in set S . In the second step, we select the suffix set E of length $n = k - 1$

for each $su \in E_{su}$ ($su = v \cdot e$) such that $\text{Pref}_{k-2}(su) = \text{Suff}_{k-2}(su')$ ($su' = b \cdot v \in E$) and $b \cdot v \cdot e \in S$, $b, e \in \Sigma$, $v \in \Sigma^*$. Third, we can calculate the cardinality of $S^\otimes(l)$ by

$$|S^\otimes(l)| = \begin{cases} \sum_{su \in E_{su}} |S_{su}| & \text{if } l = k \\ \sum_{su \in E_{su}} \sum_{su' \in E} |S_{su'}^\otimes(l-1)| & \text{if } l > k \end{cases} \quad (4.3)$$

Here, we can calculate the cardinality of $S^\otimes(l)$ of the example set as follows:

If $l = 3$, then $|S_{GG}| = 3$, $|S_{AG}| = 1$, $|S_{AT}| = 1$, $|S_{GA}| = 1$, $|S_{GT}| = 2$, $|S_{TA}| = 1$, $|S_{TG}| = 2$, $|S_{TT}| = 2$, and $|S^\otimes(3)| = 13$.

If $l = 4$, then $|S_{GG}^\otimes(l)| = 6$, $|S_{AG}^\otimes(l)| = 1$, $|S_{AT}^\otimes(l)| = 1$, $|S_{GA}^\otimes(l)| = 3$, $|S_{GT}^\otimes(l)| = 5$, $|S_{TA}^\otimes(l)| = 2$, $|S_{TG}^\otimes(l)| = 4$, $|S_{TT}^\otimes(l)| = 4$, and $|S^\otimes(4)| = 26$.

4.2.4 A Bond-free Language Closed under Concatenation

Give the example set $S = \{AA, AC, CA, CC\}$, the concatenations of the words in S^\otimes are still in language S^\otimes , but this is not always the case. Here is a counterexample. Given the set $S = \{AA, AC, CA, CC, CT, TC\}$ such that $\theta(S) \cap H_0(S) = \emptyset$, both words ACCT and TCAC are the words in S^\otimes . However, the word ACCTCAC, which is the concatenation of the words ACCT and TCAC, is not a word in S^\otimes since the subword TT is not in the set S .

To tackle this problem, the authors of [9] and [10] proposed a method to generate a language $B \subset S^\otimes$ which is closed under concatenation, i.e., $B^* \subset S^\otimes$. Here is the method to construct language B in [10]:

$$B = S^\otimes \cap BE \cdot \Sigma^* \cap \Sigma^* \cdot EN \quad (4.4)$$

where $BE \subseteq S$, $EN \subseteq S$ and $\text{Sub}_k(uv) \subseteq S$ if $u \in EN$ and $v \in BE$. Also,

$$B(l) = S^\otimes(l) \cap B \quad (4.5)$$

is a subset of B such that all the words are of length l . Equation (4.4) is based on the idea that only a subset of words concatenating to each other are still maintaining

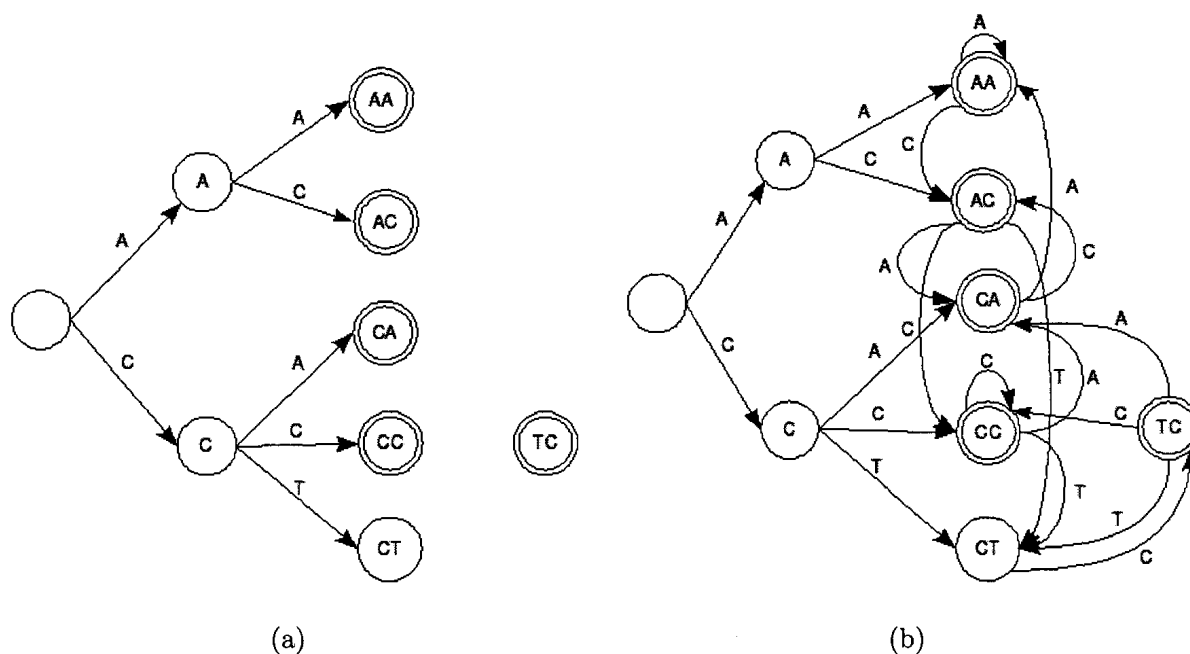


Figure 4.6: The trie represents the set $S = \{AA, AC, CA, CC, CT, TC\}$ such that the end set is $EN = \{AA, AC, CA, CC, TC\}$ and the begin set is $BE = \{AA, AC, CA, CC, CT\}$

the subword closure property. The method described in [10] is to select a subset $EN \subseteq S$ and a subset $BE \subseteq S$ such that all the words in the language B end with and start with words in EN and BE , respectively. The words that violate the concatenation closure property are caused by the new subwords at the connecting place, so all the words in EN can concatenate to all the words in BE and still remain in set S^\otimes . Regarding the counterexample that the word $ACCTTCAC \notin S^\otimes$, if we choose $EN = \{AA, AC, CA, CC, TC\}$ as the end set and $BE = \{AA, AC, CA, CC, CT\}$ as the begin set, then a word $w = uv$ is in S^\otimes where $u \in EN$ and $v \in BE$.

Since the language B should be closed under concatenation, the corresponding trie should be modified to satisfy this property. The new method to construct the trie, which is a little bit different from the one mentioned before is described as follows: Firstly, we only generate the trie structure based on the begin set BE , and the words in the set S but not in BE will be the isolated vertices in the graph. Also, we mark

all vertices representing the words in the end set EN as the final states. Figure 4.6(a) shows the result of the first step, using the set $S = \{AA, AC, CA, CC, CT, TC\}$.

The second step is the same as previously, in which we draw an arrow from vertex u to vertex v and label it with the last symbol in v if $\text{Suff}_{k-1}(u) = \text{Pref}_{k-1}(v)$ and u, v represent words in set S . The final trie corresponding to the example is shown in Figure 4.6(b).

Furthermore, the way to calculate the cardinality of $S^\otimes(l)$ should be modified. Again, we use set $S = \{AA, AC, CA, CC, CT, TC\}$ as an example. All the distinct suffixes $su = v \cdot e$ of length $n = 1$ in S are A, C and T, i.e., $E_{su} = \{A, C, T\}$. In this example, the length of words in S is $k = 2$ and the length of the suffix is $n = k - 1 = 1$, therefore, $su = e$ where v is empty, i.e., $v = \lambda$. We can obtain $S_{su}^\otimes(l)$ of length l ($l > k$) of each $su \in E_{su}$ as follows:

$$\begin{aligned} S_A^\otimes(l) &= [S_A^\otimes(l-1) \cdot A] \cup [S_C^\otimes(l-1) \cdot A], \\ S_C^\otimes(l) &= [S_A^\otimes(l-1) \cdot C] \cup [S_C^\otimes(l-1) \cdot C] \cup [S_T^\otimes(l-1) \cdot C] \text{ and} \\ S_T^\otimes(l) &= S_C^\otimes(l-1) \cdot T. \end{aligned}$$

All the words in B are starting with words in BE , so let BE_{su} denote the set of distinct words in BE ending with suffix su of length $k - 1$. If $BE = \{AA, AC, CA, CC, CT\}$, then the set of words of length l ($l = k$) ending with suffix su are as follows:

$$\begin{aligned} S_A^\otimes(l) &= BE_A = \{AA, CA\}, \\ S_C^\otimes(l) &= BE_C = \{AC, CC\}, \text{ and} \\ S_T^\otimes(l) &= BE_T = \{CT\}. \end{aligned}$$

Since the words in the set B are ending with words in set EN , let set EN_{su} denote the set of all the distinct suffixes of length $k - 1$ in the set EN . If $EN = \{AA, AC, CA, CC, TC\}$, then $EN_{su} = \{A, C\}$. Therefore, the cardinality of $B(l)$ is

$$|B(l)| = \begin{cases} \sum_{su \in EN_{su}} |BE_{su}| & \text{if } l = k \\ \sum_{su \in EN_{su}} \sum_{su' \in E} |S_{su'}^\otimes(l-1)| & \text{if } l > k \end{cases} \quad (4.6)$$

For each particular $su \in EN_{su}$ ($su = v \cdot e$, $v \in \Sigma^*$, and $e \in \Sigma$), the set E is the set of all the distinct suffixes $su' = b \cdot v$ ($b \in \Sigma$ and $v \in \Sigma^*$) of length $k - 1$ in S such that $\text{Pref}_{k-2}(su) = \text{Suff}_{k-2}(su')$ and $b \cdot v \cdot e \in S$. To clearly state the method, we give some examples of $|B(l)|$ with $n = 1$ and $k = 2$.

If $l = 2$, then $|BE_A| = 2$, $|BE_C| = 2$, $|BE_T| = 1$, and $|B(2)| = |BE_A| + |BE_C| = 4$.

If $l = 3$, then $|S_A^\otimes(3)| = 4$, $|S_C^\otimes(3)| = 5$, $|S_T^\otimes(3)| = 2$, and

$$|B(3)| = |S_A^\otimes(3)| + |S_C^\otimes(3)| = 9.$$

If $l = 4$, then $|S_A^\otimes(4)| = 9$, $|S_C^\otimes(4)| = 11$, $|S_T^\otimes(4)| = 5$, and

$$|B(4)| = |S_A^\otimes(4)| + |S_C^\otimes(4)| = 20.$$

The example mentioned above is to calculate $|B(l)|$ based on the given set BE and set EN . But there are many ways to choose the set BE and the set EN . A straightforward method to select the sets BE and EN is to evaluate them by $|B(l)|$. [10] gave two guidelines on how to choose the sets BE and EN . Guideline 1 states that the words assigned to EN are based on the suffixes of length $k - 1$, that is, for each assignment, a set whose words all end with the same suffix of length $k - 1$ will be assigned to EN . For each distinct suffix u of length $k - 1$, there is a subset B_u of words in S such that $\text{Sub}_k(u \cdot B_u) \subseteq S$. Different suffixes u may have different sets B_u . This property leads to Guideline 2: all the distinct suffixes u are sorted by the cardinality of sets B_u s and the words are assigned to EN based on this order. For each assignment of words to EN , BE is obtained by the intersection of sets B_u s.

4.3 Construct the Segments of the Multiple-hairpin Structure

In this section, we first investigate the constructions of S_1 and S_2 in Section 4.3.1 and then introduce a method to construct the languages $B_{S_1}(s)$ and $B_{S_2}(l)$ that satisfy three additional properties in Section 4.3.2.

As previously stated, if we choose proper parameters k and d_1 , we can produce a set of words S satisfying $\theta(S) \cap H_{d_1}(S) = \emptyset$ and therefore forbid undesirable hybridizations U1 and U2. Generally, researchers are more interested in designing DNA secondary structures with regular and special substructures. In our problem, the desirable DNA structure (4.1) mainly consist of single-hairpin structures $(v_i y_i \theta(v_i))$ of equal size. For each single hairpin structure, an expected subsequence will fold into the structure $v_i y_i \theta(v_i)$ without any shift hybridization. To prevent the shift hybridization U3, we can design the stem and the loop separately and make them dissimilar to each other. Here, we introduce a step to assign the words of S to two distinct subsets S_1 and S_2 such that

$$h(S_1, S_2) > d_2. \quad (4.7)$$

By choosing a suitable parameter d_2 , $h(u, v) > d_2$ ($u \in S_1$, $v \in S_2$ and $|u| = |v| = k$), we can ensure that any subwords of length k of words in S_1^\otimes and S_2^\otimes respectively are at a distance larger than d_2 . If we obtain both sets S_1 and S_2 , then they can be used to construct the stems and the loops, respectively. Therefore, any blocks in the stems will be at least $d_2 + 1$ bases different from the blocks in the loops.

If we take two words w of length s and v of length l from S_1^\otimes and S_2^\otimes , respectively, to construct the stem and loop in a single hairpin structure, the concatenation wv is still $(\theta, H_{d_1, k})$ -bond-free. In addition, it is desirable if the catenation of two words in the same set S_i^\otimes is still $(\theta, H_{d_1, k})$ -bond-free. Taking into account these considerations, given S_1 and S_2 , we will construct the languages B_{S_1} and B_{S_2} such that:

P1: For each $S_i \subseteq S$ where $i = 1$ or 2 , $B_{S_i} \subseteq S_i^\otimes$.

P2: For each $S_i \subseteq S$ where $i = 1$ or 2 , $(B_{S_i})^* \subseteq S^\otimes$.

P3: $B_{S_1} \cdot B_{S_2} \subseteq S^\otimes$.

By property P1, the language B_{S_i} ($i = 1, 2$) is $(\theta, H_{d_1, k})$ -bond-free. Since $S_i \subset S$ and $\theta(S) \cap H_{d_1}(S) = \emptyset$, if words x and y are any two words in S_i , then $x, y \in S$, thus,

$h(x, \theta(y)) > d_1$. Also, B_{S_i} is a subset of the subword closure of S_i , i.e., $B_{S_i} \subset S_i^\otimes$ and all the subwords of length k are from S_i , i.e., $\text{Sub}_k(B_{S_i}) \subseteq S_i$. If the language B_{S_i} satisfies P2, then the language $(B_{S_i})^*$ is also $(\theta-H_{d_1,k})$ -bond-free. This property is straightforward since $\text{Sub}_k(B_{S_i}) \subseteq S$ and $\theta(S) \cap H_{d_1}(S) = \emptyset$. Property P3 can ensure the concatenation of B_{S_1} to B_{S_2} , $B_{S_1}B_{S_2}$, is $(\theta-H_{d_1,k})$ -bond-free. By $B_{S_i}(l)$, we denote a subset of B_{S_i} that contains all the words of length l , that is $B_{S_i}(l) = B_{S_i} \cap \Sigma^l$.

Therefore, in Section 4.3.1, we describe the constructions of S_1 and S_2 , and then in Section 4.3.2, we introduce a method to construct the languages $B_{S_1}(s)$ and $B_{S_2}(l)$ that satisfy all three properties P1 - P3.

4.3.1 Construct Subsets S_1 and S_2 from S

When we try to assign the words in S to either S_1 or S_2 , we make two observations. (1) Given parameter d_2 , if there are subsets S_1 and S_2 satisfying $h(S_1, S_2) > d_2$ (4.7), then $S_1 \cup S_2 \subseteq S$. That is, not all the words of S can be assigned to either S_1 or S_2 and still satisfy the Hamming distance constraint. (2) Based on the way we assign the words in S , distinct results can be achieved.

Regarding to the first observation, here is an example. Given the set $S = \{\text{GGG}, \text{TGG}, \text{AGG}, \text{GTG}, \text{TTG}, \text{GAG}, \text{GGT}, \text{TGT}, \text{GTT}, \text{TTT}, \text{GAT}, \text{GGA}, \text{GTA}\}$ where $k = 3$ and $d_1 = 1$, then the subsets can be $S_1 = \{\text{TGG}, \text{AGG}, \text{GTG}, \text{GAG}\}$ and $S_2 = \{\text{GGT}, \text{TTT}, \text{GGA}\}$ and $h(S_1, S_2) > 1$. All other words ($\text{GGG}, \text{TTG}, \text{TGT}, \text{GTT}, \text{GAT},$ and GTA) in S can be put in neither S_1 nor S_2 , otherwise, the condition $h(S_1, S_2) > 1$ will fail.

For the second observation, we give an example to assign the words in S solely based on the Hamming distance constraint. That is, we assign the first word to S_1 and iteratively assign word w to S_2 if $h(S_1, w) > 1$, otherwise to S_1 . Thus, we may obtain $S_1 = \{\text{GGG}, \text{TGG}, \text{AGG}, \text{GTG}, \text{TTG}, \text{GAG}, \text{GGT}, \text{TGT}, \text{GTT}, \text{TTT}, \text{GAT}, \text{GGA}, \text{GTA}\}$ and $S_2 = \emptyset$. This assignment is unsuitable, since one of the subsets is empty.

Based on the observations and properties P1 - P3, we obtain three insights of the assignment. First, the initial several words w assigned to S_1 and S_2 , respectively, are important since sets S_1 and S_2 should satisfy $H(S_1, S_2) > d_2$. The sets S_1 and S_2 are related to each other, if the initial several words are assigned improperly, it is difficult to assign the words remaining in the set S to these two sets. Also, sets B_{S_1} and B_{S_2} that their subwords of length k are from S_1 and S_2 , respectively, should be closed under concatenation. We have already evaluated S and have the suitable EN and BE of S , therefore, when we assign the first several words, it is better to assign the words in EN and BE . Secondly, by the same token of Section 4.2.2, we select the subsets mainly based on the overlapping property and the Hamming constraint. If we assign the word only based on the Hamming distance constraint, it is very easy to obtain a set S_i such that $B_{S_i}(l)$ is empty. Finally, we may want to assign more words to S_1 , because S_1 is used to construct the words in stems and the stability of the multiple-hairpin structure depends mainly on the stable and correct base pairings.

4.3.2 Construct Sets $B_{S_1}(s)$ and $B_{S_2}(l)$

In this subsection, we will introduce the main method to construct set $B_{S_1}(s)$ and $B_{S_2}(l)$ that satisfy properties P1, P2 and P3. These two sets will be used to construct the stem and the loop, respectively. If these two languages satisfy properties P1 - P3, then $B_{S_1}(s)$ and $B_{S_2}(l)$ are $(\theta-H_{d_1,k})$ -bond-free language. Furthermore, all languages $(B_{S_1}(s))^*$, $(B_{S_2}(l))^*$ and $B_{S_1}(s) \cdot B_{S_2}(l)$ are $(\theta-H_{d_1,k})$ -bond-free.

Sets $B_{S_1}(s)$ and $B_{S_2}(l)$ satisfying properties P1 and P2 can be obtained by modifying the method used to generate the language $B(l)$ reported in Section 4.2.4. This method is based on the condition

$$B(l) = S^{\otimes} \cap BE \cdot \Sigma^* \cap \Sigma^* \cdot EN \cap \Sigma^l$$

where $EN \subseteq S$, $BE \subseteq S$ and $\text{Sub}_k(EN \cdot BE) \subseteq S$. Thus, analogous to $B(l)$, we may

construct $B_{S_i}(l_i)$ where $i= 1$ or 2 by

$$B_{S_i}(l_i) = S_i^\otimes \cap BE \cdot \Sigma^* \cap \Sigma^* \cdot EN \cap \Sigma^{l_i}$$

where $EN \subseteq S_i$, $BE \subseteq S_i$ and $\text{Sub}_k(EN \cdot BE) \subseteq S$. In the same manner, we may find subsets EN and BE of S_i such that $\text{Pref}_k(w) \in BE$ and $\text{Suff}_k(w) \in EN$ if w is a word in $B_{S_i}(l_i)$. In addition, if words $w \in EN$ and $u \in BE$, then any subwords of length k of wu are in S instead of S_i . By this, the constraint is less strict and more words may be assigned to sets EN and BE .

Using the method mentioned above, we can produce $B_{S_1}(s)$ and $B_{S_2}(l)$ that satisfy properties P1 and P2. In order to satisfy property P3, that $B_{S_1}(s) \cdot B_{S_2}(l) \subset S^\otimes$, we may refine the end set of $B_{S_1}(s)$ and the begin set of $B_{S_2}(l)$. By EN_1 and BE_2 , we denote the end set of $B_{S_1}(s)$ and the begin set of $B_{S_2}(l)$, respectively. If word $w \in B_{S_1}(s)$, then the suffix of w of length k should be in set EN_1 . Also, if $v \in B_{S_2}(l)$, then the prefix of v of length k should be in set BE_2 . Therefore, we only need to check the subword closure of $EN_1 \cdot BE_2$, i.e., $\text{Sub}_k(EN_1 \cdot BE_2) \subseteq S$. We want to retain most of the words in EN_1 to preserve the cardinality of $B_{S_1}(s)$. The method we used is straightforward: we find subsets $BE'_2 \subseteq BE_2$ and $EN'_1 \subseteq EN_1$ such that $\text{Sub}_k(EN'_1 \cdot BE'_2) \subseteq S$ and $|EN'_1|$ is maximal. The detailed algorithm will be shown in Section 5.2.3.

4.4 Apply Additional Combinatorial Constraints

In this section, we mention additional constraints such as the Hamming distance constraint, the GC content constraint, the continuity constraint that use to weed out the undesirable sequences in the sets $B_{S_1}(s)$ and $B_{S_2}(l)$ in order to obtain the sets B_{stem} and B_{loop} , respectively.

In Grail [35], the function *enumerate* can find out a subset of words of distinct lengths accepted by an automaton. By the modification of this function, [9], we can

generate all the words of certain length accepted by an automaton. Generally, not all the words accepted by an automaton are useful for the DNA secondary structure design. For distinct segments of the structure, we mention several kinds of unacceptable words and the corresponding constraints mentioned in Section 3.2.2 to forbid them.

Firstly, words of certain length accepted by the automaton may be too similar to each other, which reduces the chances of correct hybridization. Let us use the set $S = \{AA, AC, CA, CC\}$ as an example. The set of words of length 7 accepted by the corresponding automaton contains words ACACCAA and CCACCAA that differ by one base only. To address this problem, we can apply constraint C1, the Hamming distance constraint, when we select the words to construct the secondary structure. That is, when we put a new word into the desirable word set, we always make sure the Hamming distance between the new word and the word set is above certain threshold d_3 .

Secondly, the number of bases G and C in the words generated by the automaton are widely different from each other and this may effect the melting temperature of the DNA molecules and increase the difficulty to control the molecular operations. For example, the automaton corresponding to S accepts words AAAAAAA and CCCCCCC. The word AAAAAAA contains no base G or C but word CCCCCCC contains only the base C. Therefore, when we select the desirable words, we will make sure that the GC content of the words is within a certain range from GC_{lower} to GC_{upper} . This is constraint C4 mentioned in Section 3.2.2.

Thirdly, if a DNA word provided to construct the stem has a long contiguous segments of the same base, it is unacceptable since it may make the structure unstable. Using the same example mentioned, the words AAAAAAA and CCCCCCC have contiguous stretches of A and C, respectively. This problem can be solved if we apply the continuity constraint (C5) to remove all the words containing a segment of consecutive single bases of length larger than certain parameter c .

Finally, we may apply a constraint to sequences constructing the loops, that the

first and the last base of the sequences cannot be Watson-Crick complementary pair. This constraint is adopted because these two bases link to the stems and it is sure that they will bind to each other if they are Watson-Crick complementary to each other. We called this constraint C7.

To summarize, the unwelcome words generated by the automaton can be filtered out by applying the combinatorial constraints. As mentioned in 3.2.2, many kinds of constraints were proposed, however, we just adopt some of them to filter out undesirable sequences. For the stem and the loop, we may use different combinatorial constraints. A more strict combinatorial constraints would be applied to the stem to produce B_{stem} , since the sequences in the stem will decide the final structure that the whole sequence folds into. We implemented an algorithm to filter out all the words that violated the constraints. The main process is straightforward. Constraints that are only related to each particular word may be checked first, such as constraints C4, C5 and C6. Constraint C1 will only be checked if the word satisfied all the other given constraints. This process will apply to the words generated by the automaton in a certain order. Finally, we can obtain a set of words accepted by the automaton corresponding to the given set S such that all of them satisfy the given combinatorial constraints.

4.5 Generate a Set of Sequences Folding into a Given Multiple-hairpin Structure

After the filtration procedure described in Section 4.4, we may obtain a subset B_{stem} of $B_{S_1}(s)$ and a subset B_{loop} of $B_{S_2}(l)$ satisfying the respective combinatorial constraints. In this section, we describe the method by which we generate the multiple-hairpin structure from the sets B_{stem} and B_{loop} .

The required DNA multiple-hairpin structure is of the form (4.1):

$$hps(n, s, l, \theta) = \{xv_1y_1\theta(v_1)\dots v_ny_n\theta(v_n) \mid x, v_i, \theta(v_i) \in \Sigma^s, y_i \in \Sigma^l \text{ and } 1 \leq i \leq n\}.$$

In the structure, the length of the external base segment $|x|$ and the stems $|v_i|$ and $|\theta(v_i)|$ are identical and equal to s . In [48], the authors constructed a four hairpin structure and the length of the stem s and the loop l are equal to 20 and 7, respectively. In this paper, we will mainly construct structures similar to the one in [48], that is, the number of hairpin n is 4, the length of stem s is 20 and the length of loop segment l is 7. The general idea to construct the sequence folding into the multiple-hairpin structure is to assign words in the set B_{stem} to the external segment and each stem, and assign words in the set B_{loop} to each loop.

By the observation of the structure predicted by the Vienna RNA package [17], only some of words obtained this way will fold into the required multiple-hairpin structures. Furthermore, if we assign a word in B_{stem} multiple times, the resulting sequence can easily fold into other secondary structures. This is because a stable structure is based on the stability of the paired bases. If we assign a word in B_{stem} multiple times to distinct stems in the same multiple-hairpin structure, then each stem may have multiple choice of complements which will form perfect hybridizations. In order to address this issue, we will assign distinct words in the set B_{stem} to each stem and the external segment. On the other hand, if we assign the words in B_{loop} to loop segments multiple times within a multiple-hairpin structure, it seems the sequence will still fold into the desirable DNA secondary structure. Therefore, the words in B_{loop} may be assigned multiple times if the set B_{loop} does not have enough words.

In brief, we assign the words to the structure as follows: For each segment of length s , we may pick an unused word in B_{stem} . For each segment of length l , we may pick a word in B_{loop} . Since we are interested in the number of sequences that will fold into the given multiple-hairpin structure, we give the upper bound of the sequences we can generate. If the number of single hairpins in the multiple-hairpin

structure is n , the cardinality of set $B_{stem}(s)$ is m ($m > n$) and the cardinality of the B_{loop} is at least 2, then we can generate

$$P(m, n + 1) = m(m - 1)(m - 2) \dots (m - n) = \frac{m!}{(m - n - 1)!} \quad (4.8)$$

sequences of the form (4.1). For example, if there are 5 words in set B_{stem} and the structure we require is a four-hairpin structure, then we may have $\frac{5!}{(5-4-1)!} = 5! = 120$ sequences which are expected to fold into the four hairpin structure; if there are 20 words in set B_{stem} and then we may have $\frac{20!}{(20-4-1)!} = 1860480$ sequences. If we produce a large B_{stem} and an acceptable B_{loop} , we may generate a huge number of sequences. According to our consideration, all the sequences are candidates that may form the desirable multiple-hairpin structure. However, it is not sure whether all of them will fold into the required DNA structures in a laboratory experiment. In order to check the performance of the sequences, we will use the molecule structure prediction software to foresee the two dimensional structures of the sequence as seen in the following section.

4.6 Predict the Secondary Structures of the Result Sequences

Vienna RNA Package is used to foresee the secondary structure of the result sequences we generated. In this section, we describe several problems we encounter when using the package, and our solutions.

Both software packages, *Mfold* and *Vienna RNA Package*, can be used to predict the secondary structure of single DNA/RNA sequences by folding the sequence into a structure with minimum free energy, and their accuracy is dependent on the thermodynamic parameters used in the experiment. The *Vienna RNA Package* reported firstly in [17] is implemented in the programming language C which can be easily included in our program, so we used this package to predict the secondary

structure of the DNA sequences we generated. As a general rule, the folding result is presented by using brackets and dots, where a pair of brackets denotes a base pair and the dot denotes an unpaired base. For example, the secondary structure of the RNA molecule UUGGGCUAUUAGCUCAGUUGG with minimum free energy -5.90kcal/mol^1 at temperature 37°C can be represented by

UUGGGCUAUUAGCUCAGUUGG
 ((((((.....)))))).....

where the segment UUGGGC binds to the segment GCUCAG, forming base pairs, and others are unpaired bases. Although the notation is very easy to understand, it is not intuitive for the user to figure out the real secondary structure. To address this issue, the Vienna RNA package included a method to produce a pictorial representation of the two dimensional secondary structure of the folding result. The two dimensional structure of the example RNA molecule is shown in Figure 4.7(a). Since both of them are useful, we included both methods to present the sequence folding result in our software.

As described in Section 3.4.1, the prediction of secondary structure is based on calculating the minimal free energy and the energy data is collected by experiments. The secondary structure prediction is primarily used to foresee the RNA structure and only the RNA thermodynamic parameters were provided by the Vienna RNA Package. To cope with this problem, we adopt the DNA energy parameters provided by Santalucia in [41], now it is included in the Mfold package. If we change all bases U to T in the previous RNA molecule example, then we may obtain DNA molecule TTGGGCTATTAGCTCAGTTGG. Based on the parameters provided by Santalucia, we have the DNA secondary structure with minimum free energy -1.72kcal/mol at temperature 37°C . The structure is shown as follows:

¹kcal/mol: kilo-calorie (a unit of energy) per mole (a unit of amount of substance).

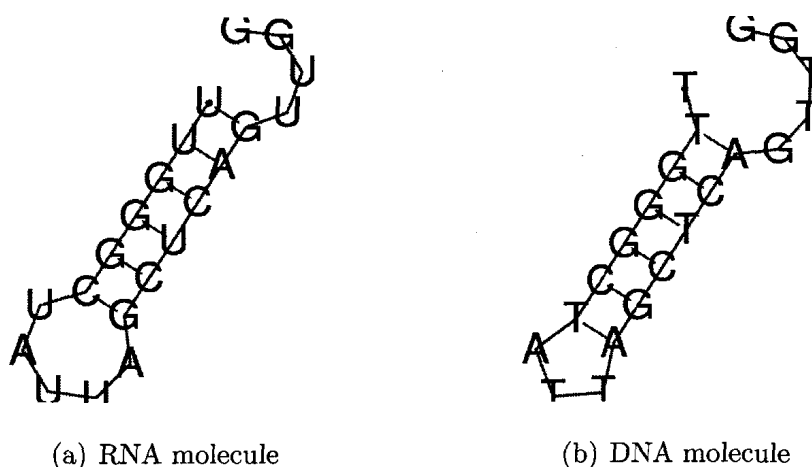


Figure 4.7: The 2D secondary structure of RNA molecule GGGCUAU-UAGCUCAGUUGG and DNA molecule TTGGGCTATTAGCTCAGTTGG.

TTGGGCTATTAGCTCAGTTGG
 .((((((...)))))).....

and the two dimensional structure is shown in Figure 4.7(b). It is clear that the thermodynamic parameters of RNA and DNA are different and we may want to use DNA thermodynamic parameters to predict the secondary structure of our result DNA sequences.

However, we encountered two problems when we used the DNA energy data. First of all, Vienna RNA package and Mfold used two different data formats to display their energy parameters, so we implemented an algorithm to convert the data format in Mfold to the one in Vienna RNA package. Secondly, the secondary structure of DNA/RNA molecular depends on several environment conditions such as temperature and ionic conditions. Generally, all the thermodynamic parameters are measured at temperature $T = 37^{\circ}\text{C}$ and the ionic conditions are $Na^{+} = 1.0\text{m}$ and $Mg^{++} = 0.0\text{m}$. By applying these environment conditions, the result of Vienna RNA package is consistent to Mfold. If use a temperature other than 37°C , then the resulting minimum free energy may be different from the one in Mfold since the calculation of the energy parameters is different between Vienna RNA package and

Mfold package. Therefore, we will evaluate our data mainly at temperature 37°C.

Furthermore, we forbid all possibility of nucleotide G hybridize to nucleotide T, and vice versa. As mentioned in Section 1.1.2, RNA has four kinds of bases and the nucleotide will form the non-Watson-Crick complementarity by which U may bind to A as well as G. In this paper, we follow the general idea of Watson-Crick complementarity which is $\theta(A) = T$, $\theta(C) = G$, $\theta(G) = C$, and $\theta(T) = A$. Also all the equations we used are based on the Watson-Crick complementarity. Therefore, the method we used cannot prevent the undesirable bond between G and T. Regarding this issue, we change the default parameter of the Vienna RNA package in such way that it forbids any nucleotides G binding to nucleotides T and vice versa, when predicting the secondary structure of a given sequence.

Chapter 5

Implementation and Experimental Results

In Chapter 4, we have described our bottom-up approach and the details of problems we want to address. In order to validate our approach, we implemented a software which we briefly describe in Section 5.1. Our bottom-up approach is mainly based on the bond-free property, therefore, we describe the main algorithm related to the bond-free languages in Section 5.2.

The bond-free languages $B(l)$ closed under concatenation have already been investigated in [9]. Here we improve the method to generate the set S by which we can produce larger bond-free languages $B(l)$. The experimental results of constructing the bond-free languages are given in Section 5.3. The experimental results of our bottom-up approach to construct the consecutive hairpin structure is given in Section 5.4. Finally, in Section 5.5, we discuss the way to select the parameters used in our bottom-up approach.

5.1 Main Structure of the Software

A C++ software is implemented to study and characterise the behavior of this approach. The whole procedure of our software can be divided into five steps.

First, a set of words S of equal length over the DNA alphabet satisfying the bond-

free property, that is, $h(w, \theta(v)) > d_1, \forall w, v \in S$, is selected by providing parameters k and d_1 . By parameters k and d_1 , we denote the length of the words in S and the reverse-complement Hamming distance of any two words in this set, respectively.

In the second step, we put the words into two subsets S_1 and S_2 such that Hamming distance between these sets is larger than a parameter d_2 .

In the third step, we construct two $(\theta-H_{d_1,k})$ -bond-free languages $B_{S_1}(s)$ and $B_{S_2}(l)$ from S_1 and S_2 , respectively. That is, all subwords of length k of words in the language B_{S_i} are from the set S_i ($i = 1, 2$). Both languages $B_{S_1}(s)$ and $B_{S_2}(l)$ are closed under concatenation and the concatenation of $B_{S_1}(s)$ and $B_{S_2}(l)$ is also $(\theta-H_{d_1,k})$ -bond-free. By this step, we may prevent the words at the jointed place from violating the bond-free property.

In the fourth step, we apply combinatorial constraints to these two languages $B_{S_1}(s)$ and $B_{S_2}(l)$ to filter out undesirable words and obtain two sets of words B_{stem} and B_{loop} .

Finally, a single hairpin structure is obtained by the concatenation of three words $v \in B_{stem}$, $y \in B_{loop}$ and $\theta(v)$, and the consecutive hairpin structure is obtained by concatenating the external segment and the single hairpin structures. Furthermore, in the final step, the software package Vienna RNA is included to evaluate the real structure of the sequences we generated.

To clearly state the process of our software, Figure 5.1 shows the flow chart of this bottom-up approach. In the flow chart, we consider several conditions to make sure the sets we generate are working properly. For example, we evaluate the set S by calculating the cardinality of the language B mentioned in Section 4.2.4. If the cardinality of language B of words of certain length n is too small, then the algorithm will terminate.

The main algorithms carried out in our software consist of templates `word`, `sort_set`, `hamming_set`, `bond_free_set`, `select_subset`, `trie`, `filter` and `check_struct`. All the templates are developed according to sets with particular properties and the

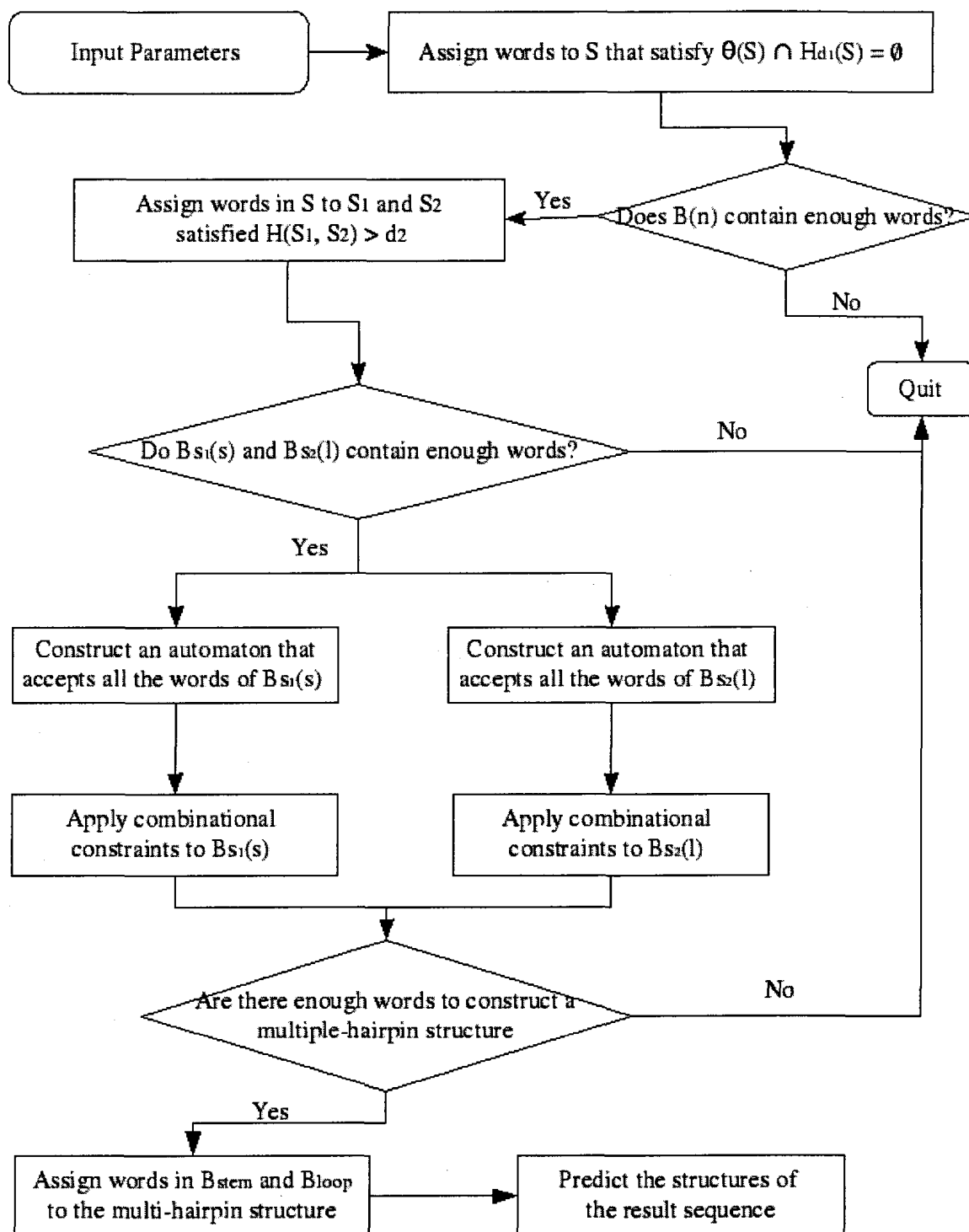


Figure 5.1: The flow chart of software.

elementary processes are the operations on the words. Hence, the templates are implemented by using inheritance and the respective relations are shown in Figure 5.2.

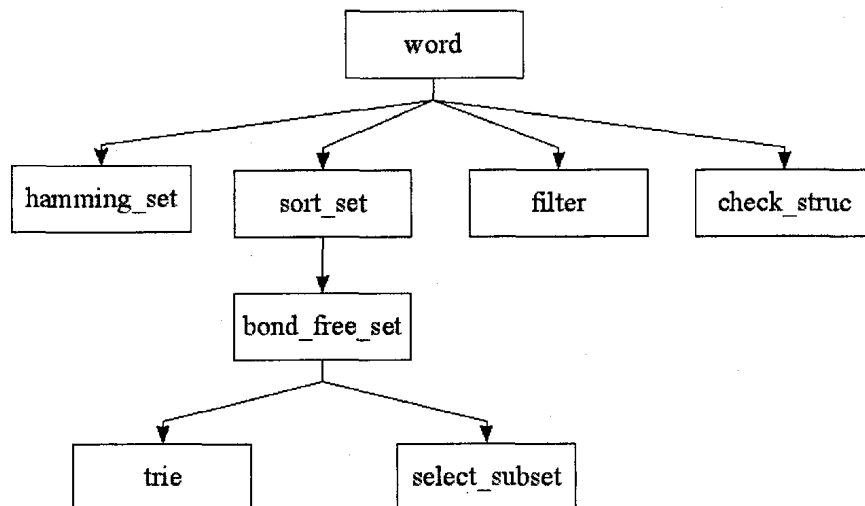


Figure 5.2: The inheritance relations of templates `word`, `sort_set`, `hamming_set`, `bond_free_set`, `select_subset`, `trie`, `filter` and `check_struct`.

The basic template `word` contains all fundamental operations on words. For example, obtaining the antimorphic involution image of a given word, obtaining the subword of a DNA word of certain length, the conversion between indexes and DNA words, getting the Hamming distance between two words of equal length, the test of equality of two words, the test of equality of two subwords, and etc. The template `hamming_set` is a function to generate the set S of words of fixed length k that satisfies $\theta(S) \cap H_{d_1}(S) = \emptyset$. Since all the sets S , S_1 and S_2 should be sorted in a certain order to construct the trie, algorithm *Radix sort* is adopted in the template `sort_set` to sort the sets. The template `bond_free_set` is the main algorithm to construct the concatenation closure of a bond-free language, such as the set B , B_{S_1} or B_{S_2} of words of certain length. In the template `trie`, we construct a trie to generate bond-free languages such that their end sets EN and begin sets BE are decided in the template

`bond_free_set`. The template `select_subset` is the algorithm to assign words of S to sets S_1 and S_2 that satisfy $H(S_1, S_2) > d_2$. Since we produce sets S_1 and S_2 based on the evaluation of S , the template `select_subset` is a derive class of the template `bond_free_set`. In the template `filter`, the combinatorial constraints which consist of the Hamming distance constraint (C1), the GC content constraint (C4), the continuity constraint (C5) and the end constraint (C7), are implemented to filter out the undesirable words of a given set.

In addition, we include and modify several templates from the library Grail to help the construction of automata in the template `trie`: they are the templates `array`, `String`, `list`, `set`, `inst`, `state`, and `fm`. Finally, when we evaluate the behavior of our result in the template `check_struct`, we involve several classes in the Vienna RNA Package version 1.7.1 and the DNA thermodynamic parameters measured by SantaLucia [41].

5.2 Algorithms Related to Bond-free Languages

In this section, we will report the algorithms we have implemented that relate to bond-free languages. We first describe the algorithm to select the set S in Section 5.2.1. Secondly, the algorithm constructing the sets S_1 and S_2 is reported in Section 5.2.2. Finally, in Section 5.2.3, we introduce the algorithm of constructing the bond-free languages B_{S_1} and B_{S_2} which are closed under catenation, and also $B_{S_1} \cdot B_{S_2}$ is bond-free.

5.2.1 Algorithm to Select Set S

We implement an improved algorithm to select a set S satisfying equation (4.2), based on the overlapping property mentioned in Section 4.2.2. If the word length is k and the alphabet is Σ , then the size of the solution space is $|\Sigma|^k$. Whenever we put a word w into the set S , we remove all the words u in the solution space such that

$h(w, \theta(u)) \leq d_1$. The main process is to put a word w into S and then iteratively put word v into the set S if its prefix of length $k-1$ is the same as the suffix of the previous word put into S , i.e., $\text{Suff}_{k-1}(w) = \text{Pref}_{k-1}(v)$. This process will continue until all the words are checked. The details of the algorithm are shown in the following:

Algorithm: Generate a set S such that $\theta(S) \cap H_{d_1}(S) = \emptyset$

Input: Σ - the alphabet set;

k - the length of words;

d_1 - the Hamming distance $h(w, \theta(v)) > d_1$, and $w, v \in S$; and

initial - the index of an initial word.

Output: A set of words S satisfying $\theta(S) \cap H_{d_1}(S) = \emptyset$.

Procedure:

```

1  for  $i = 0$  to  $|\Sigma|^k - 1$       ▷ Mark all the words as YES
2      do  $w[i] = \text{YES}$ 
3   $i = \text{initial}$ 
4  while  $i < |\Sigma|^k$           ▷ Check the solution space from initial to  $|\Sigma|^k - 1$ 
5      do if  $w[i] = \text{YES}$ 
6          then  $w[i] = \text{CHECK}$  and
               $w[j] = \text{NO}$  such that  $h(w[i], \theta(w[j])) \leq d_1$ 
7               $\text{current} = i$ 
8              while find  $\text{Suff}_{k-1}(w[\text{current}]) = \text{Pref}_{k-1}(w[j])$ 
9                  do if  $w[j] = \text{YES}$ 
10                     then  $w[j] = \text{CHECK}$  and
                           $w[l] = \text{NO}$  such that  $h(w[j], \theta(w[l])) \leq d_1$ 
11                      $\text{current} = j$ 
12                 else  $i++$ 
13   $i = 0$ 
14  while  $i < \text{initial}$           ▷ Check the solution space from 0 to initial - 1
15      do 5 - 12
16  return all the words such that  $w[i] = \text{CHECK}$ 

```

When we implement this algorithm, we use a mapping from $\Sigma_{quad} = \{0, 1, 2, 3\}$ to $\Sigma_{DNA} = \{A, C, G, T\}$ in the obvious way. Using this method, each DNA word can be represented by an index. For example, if α denotes a mapping from Σ_{quad} to Σ_{DNA} such that $\alpha(0) = A$, $\alpha(1) = C$, $\alpha(2) = G$, and $\alpha(3) = T$, then all the DNA words of length 2 can be represented by the indexes shown in Table 5.1. Given the mapping β :

Index	$Index_4$	word	Index	$Index_4$	word	Index	$Index_4$	word	Index	$Index_4$	word
0	00	AA	4	10	CA	8	21	GA	12	30	TA
1	01	AC	5	11	CC	9	22	GC	13	31	TC
2	02	AG	6	12	CG	10	23	GG	14	32	TG
3	03	AT	7	13	CT	11	24	GT	15	33	TT

Table 5.1: The mapping of DNA words of length 2 to quadruple code based on the mapping α that $\alpha(0) = A$, $\alpha(1) = C$, $\alpha(2) = G$, and $\alpha(3) = T$.

$\Sigma_{quad} \rightarrow \Sigma_{DNA}$ defined by $\beta(0) = G$, $\beta(1) = T$, $\beta(2) = A$, and $\beta(3) = C$, all the DNA words of length 2 can be represented by the indexes in quadruple alphabet shown in Table 5.2.

$Index_4$	DNA word	$Index_4$	DNA word	$Index_4$	DNA word	$Index_4$	DNA word
00	GG	10	TG	21	AG	30	CG
01	GT	11	TT	22	AT	31	CT
02	GA	12	TA	23	AA	32	CA
03	GC	13	TG	24	AC	33	CC

Table 5.2: The mapping of DNA words of length 2 to quadruple code based on the mapping β that $\beta(0) = G$, $\beta(1) = T$, $\beta(2) = A$, and $\beta(3) = C$.

According to the experimental results, this algorithm is independent from the way we define the mapping, but the desirable result is dependent on the way we define the mapping. Here is an example. Given parameters $k = 3$ and $d_1 = 1$, the set S

obtained by our algorithm based on mappings α and β are {AAA, CAA, GAA, ACA, CCA, AGA, AAC, CAC, ACC, CCC, AGC, AAG, ACG} and {GGG, TGG, AGG, GTG, TTG, GAG, GGT, TGT, GTT, TTT, GAT, GGA, GTA}, respectively. If we choose the first set, then the ratio of the base A versus the other bases is high in the set S^\otimes . On the other hand, if we pick the second set, then the ratio of the base G versus the other bases is high in the set S^\otimes . The ratio of GC may affect the result of hybridization, so we will choose a suitable mapping when testing the result.

5.2.2 Method to Construct the Sets S_1 and S_2

According to our observations, it is difficult to find an optimal and simple algorithm to construct two sets S_1 and S_2 such that $h(S_1, S_2) > d_2$ and $S_1 \subset S$, $S_2 \subset S$. The sequences that will fold into the correct multiple-hairpin structure depend on the stable folding of the stems, so we are more interested in getting sets S_1 and S_2 that can make B_{S_1} fairly large. Furthermore, according to Section 4.3, S_1 and S_2 are used to construct $B_{S_1}(s)$ and $B_{S_2}(l)$ that satisfy properties P1 - P3. Therefore, when we construct S_1 and S_2 , we may select words that ensure these properties. As discussed in Section 4.3.1, simple algorithm cannot produce proper subsets S_1 and S_2 satisfying $h(S_1, S_2) > d_2$ and properties P1 - P3.

The way to producing sets S_1 and S_2 is not quite clear, we get the way to produce the words by a “trial and error” method. Our method of producing sets S_1 and S_2 is based on the observations of the data we obtained in the experiment. The observations are shown as follows.

First, it is better to select words based on both the Hamming distance constraint and the overlapping property. Given words w_1 and w_2 , if $\text{Pref}_{k-1}(w_1) = \text{Suff}_{k-1}(w_2)$, then the overlap segment of w_1 and w_2 is at the beginning of w_1 , see Figure 5.3(a); On the other hand, if $\text{Suff}_{k-1}(w_1) = \text{Pref}_{k-1}(w_2)$, then the overlap segment of w_1 and w_2 is at the end of w_1 , see Figure 5.3(b). In the method, we select words from S to S_1 and S_2 iteratively by checking the overlapping property and the Hamming distance

$$\begin{array}{rcc}
 \mathbf{w}_1 = \mathbf{a}_1 \dots \mathbf{a}_{k-2} \mathbf{a}_{k-1} \mathbf{a}_k & & \mathbf{w}_1 = \mathbf{a}_1 \mathbf{a}_2 \dots \mathbf{a}_{k-1} \mathbf{a}_k \\
 | \dots | | & & | \dots | | \\
 \mathbf{w}_2 = \mathbf{a}_0 \mathbf{a}_1 \dots \mathbf{a}_{k-2} \mathbf{a}_{k-1} & & \mathbf{w}_2 = \mathbf{a}_2 \dots \mathbf{a}_{k-1} \mathbf{a}_k \mathbf{a}_{k+1}
 \end{array}$$

(a) The overlap region is at the beginning of w_1 .

(b) The overlap region is at the ending of w_1 .

Figure 5.3: Two kinds of overlaps between words w_1 and w_2

constraint. This process will continue until it exceeds a certain number of times.

Second, at the initial state, the words assigned to sets S_1 and S_2 , respectively, are important since the following words are selected if they have segments that can overlap with the word already selected, and moreover, satisfy the Hamming distance constraint. Also, the sets S_1 and S_2 are used to construct the stems and the loops which should satisfy properties P1 - P3. When we produce S , we also evaluate the cardinality of $B(l)$ of words of length l and obtain the best end set EN and the best begin set BE which can produce large $B(l)$. Therefore, the words initially selected into S_1 and S_2 are from these two sets.

Finally, the unused words in the set S will be checked to see whether they can be selected to either the set S_1 or the set S_2 . By this step, we ensure we use most of the words in S to construct the sets S_1 and S_2 . The main method to produce sets S_1 and S_2 is as follows:

Algorithm: Generate sets S_1 and S_2 such that $H(S_1, S_2) > d_2$ and $S_1, S_2 \subset S$.

Input: S - the set satisfying $\theta(S) \cap H_{d_1}(S) = \emptyset$;

$size$ - the cardinality of S ;

k - the length of words in S ;

d_2 - the Hamming distance that $h(w, v) > d_2$ and $w \in S_1, v \in S_2$;

EN - the end set of S ; and

BE - the start set of S .

Output: Subsets S_1 and S_2 of S satisfying $H(S_1, S_2) > d_2$.

Procedure:

Initially select several words into S_1 and S_2 , respectively

for $n \leftarrow 0$ **to** MAX \triangleright MAX is a parameter indicating the number of times.

do $Overlap(S, S_1, S_2, either)$ and $Overlap(S, S_2, S_1, end)$

for $i \leftarrow 0$ **to** $size - 1$

do if $S[i] \notin S_1$ and $h(S[i], S_2) > d_2$

then $S_1 = S_1 \cup S[i]$

if $S[i] \notin S_2$ and $h(S[i], S_1) > d_2$

then $S_2 = S_2 \cup S[i]$

By the following algorithm $Overlap(S', I, O, type)$, we can easily assign a word in S' to set I such that both the Hamming distance constraint $H(I, O) > d_2$, and the overlapping property are satisfied. Here, S' can be the end set EN , the start set BE and set S .

Algorithm: *Overlap*($S', I, O, type$): Assign a word $S'[i] \in S'$ to set I such that $h(I, O) > d_2$ and $S'[i]$ overlaps with one of the word $I[j] \in I$ of certain type.

Input: S' - a subset of S ;

I - a subset of S ;

O - a subset of S ; and

$type$ - the overlapping type (*begin*, *end* and *either*).

Procedure:

if ($type = begin$)

then find a word $S'[i] \in S'$ such that $S'[i] \notin I$, $h(S'[i], O) > d_2$
and $Pref_{k-1}(I[j]) = Suff_{k-1}(S'[i])$.

if ($type = end$)

then find a word $S'[i] \in S'$ such that $S'[i] \notin I$, $h(S'[i], O) > d_2$
and $Suff_{k-1}(I[j]) = Pref_{k-1}(S'[i])$.

if ($type = either$)

then find a word $S'[i] \in S'$ such that $S'[i] \notin I$, $h(S'[i], O) > d_2$ and either
 $Pref_{k-1}(I[j]) = Suff_{k-1}(S'[i])$ or $Suff_{k-1}(I[j]) = Pref_{k-1}(S'[i])$.

if the word $S'[i]$ is found

then $I = I \cup S'[i]$

5.2.3 Bond-free Languages Closed under Catenation

As mentioned in Section 4.3, we need to produce two bond-free languages B_{S_1} and B_{S_2} that satisfy properties P1 - P3. In order to generate bond-free languages B_{S_1} and B_{S_2} that satisfy both properties P1 and P2, we modified the algorithm reported in [9] to construct bond-free languages which are closed under concatenation. Given a set S satisfying $\theta(S) \cap H_{d_1}(S) = \emptyset$ (4.2) and a set $S' \subset S$, the main idea to generate a bond-free language $B_{S'}$ which is closed under concatenation is to find subsets $EN \subseteq S'$ and $BE \subseteq S'$ such that $Sub_k(EN \cdot BE) \subseteq S$ and all the words in $B_{S'}$ can only have prefixes $p \in BE$, and suffixes $s \in EN$.

The selection of EN and BE from a set $S' \subset S$ is based on two Guidelines described in Section 4.2.4, originally from [10]. Given words $w = w_1w_2\dots w_k \in EN$ and $v = v_1v_2\dots v_k \in BE$ of length k , it is easy to know that both words w and v are in S . Therefore, when we check whether all the subwords of length k of the word $wv = w_1w_2\dots w_kv_1\dots v_{k-1}v_k$ are in S , we only need to check whether all the subwords of length k of the word $\text{Suff}_{k-1}(w)\text{Pref}_{k-1}(v) = w_2\dots w_kv_1\dots v_{k-1}$ are in S . This observation led to the first guideline, the words are assigned to EN based on suffixes of length $k - 1$, that is, whenever we assign a word $w \in S'$ to EN , we also assign the words $u \in S'$ to EN that have the same suffix of length $k - 1$ as w . Furthermore, a pair of EN and BE is evaluated based on the cardinality of the set $B_{S'}$ of words of certain length. The algorithm is as follows:

Algorithm: Generate a pair of subsets EN and BE of S' which can generate a large set $B_{S'}$ of words of certain length satisfying properties P1 and P2.

Input: S - the set satisfies $\theta(S) \cap H_{d_1}(S) = \emptyset$;

S' - a subset of S ; and

k - the length of the words.

Procedure:

Find a set SU which contains all distinct suffixes of length $k - 1$ in S'

for each $SU[i] \in SU$

do find the corresponding $BE_i \subset S'$ such that $\text{Sub}_k(SU[i] \cdot BE_i) \subseteq S$

Sort SU according to the cardinality of BE_i in descending order

$cur_SU = \emptyset$, $cur_BE = S'$, and $i = 0$

while $i < |SU|$

do $cur_SU = cur_SU \cup \{SU[i]\}$

$cur_BE = cur_BE \cap BE_i$

if cur_SU and cur_BE can generate larger $|B_{S'}|$ of words of certain length

then $best_SU = cur_SU$ and $best_BE = cur_BE$

$i++$

Find the set EN which contains all the words in S' such that $\text{Sub}_k(EN) = best_SU$

$BE = best_BE$

In this algorithm, the method to evaluate the cardinality of the set $B_{S'}$ of words of certain length is described in Section 4.2.4.

In order to make the algorithm more clear, we give example sets S , S_1 and S_2 and show how to construct the pairs of EN and BE of S_1 and S_2 , respectively, that satisfy properties P1 and P2. The example sets S , S_1 and S_2 are given in Table 5.3 with parameters $k = 3$, $d_1 = 1$ and $d_2 = 0$. With these parameters, any two words $w, v \in S$ satisfy $h(w, \theta(v)) > 1$ and any two words $w \in S_1, v \in S_2$ satisfy $h(w, v) > 0$.

Set	All the words in the set
S	{GGG, GGT, GGA, GTG, GTT, GTA, GAG, GAT, TGG, TGT, TTG, TTT, AGG}
S_1	{GGG, TGG, AGG, GTG, GAG, GGT, GAT, GGA}
S_2	{TTG, TGT, GTT, TTT, GTA}

Table 5.3: Sets S , S_1 and S_2 with parameters $k = 3$, $d_1 = 1$ and $d_2 = 0$.

We first consider the begin set EN and the end set EN of S_1 . Firstly, we obtain the set of all distinct suffixes of length $k - 1$ of S_1 : $SU = \{GG, TG, AG, GT, AT, GA\}$. Secondly, we sort the words in the set SU according to the number of words in S_1 that it can concatenate to, and moreover, the bond-free property is maintained. That is, for a word $w \in SU$, we calculate the number of words $v \in S_1$ such that $\text{Sub}_k(wv) \subseteq S$. Table 5.4 lists each word $w \in SU$ and the words $v \in S_1$ such that $\text{Sub}_k(wv) \subseteq S$. Finally, the pair of sets EN and BE are selected by evaluating the cardinality of $B_{S_1}(l)$. We list all cur_SU tried by the algorithm and the respective BE in Table 5.5. By comparing the values of $|B_{S_1}(5)|$, we may have $best_SU = \{GG, TG\}$ and $BE = \{GGG, TGG, GTG, GAG, GGT, GAT, GGA\}$. Thus, $EN = \{GGG, TGG, AGG, GTG\}$ is obtained by $best_SU$. By this process, we finally obtain the end set $EN = \{GGG, TGG, AGG, GTG\}$ and the begin set $BE = \{GGG, TGG, GTG, GAG, GGT, GAT, GGA\}$ of S_1 and the corresponding language B_{S_1} satisfies

w	v								Size
GG	GGG	TGG	AGG	GTG	GAG	GGT	GAT	GGA	8
TG	GGG	TGG		GTG	GAG	GGT	GAT	GGA	7
AG	GGG			GTG	GAG	GGT	GAT	GGA	6
GT	GGG	TGG		GTG		GGT		GGA	5
GA	GGG					GGT		GGA	3
AT									0

Table 5.4: List of all the suffixes of S_1 , i.e., $w \in SU$, and the corresponding words v in S_1 satisfying $\text{Sub}_k(wv) \subseteq S$.

properties P1 and P2.

cur_SU	cur_BE	$ B_{S_1}(5) $
{GG}	{GGG, TGG, AGG, GTG, GAG, GGT, GAT, GGA}	7
{GG, TG}	{GGG, TGG, GTG, GAG, GGT, GAT, GGA}	8
{GG, TG, AG}	{GGG, GTG, GAG, GGT, GAT, GGA}	7
{GG, TG, AG, GT}	{GGG, GTG, GGT, GGA}	8
{GG, TG, AG, GT, GA}	{GGG, GGT, GGA}	7

Table 5.5: Given a set S_1 , the table contains all the pairs of cur_SU and cur_BE we considered in the algorithm.

By the same token, we can generate the begin set BE and the end set EN of $S_2 = \{\text{TTG, TGT, GTT, TTT, GTA}\}$ as follows: Firstly, the set of all distinct suffixes of length $k - 1$ is $SU = \{\text{TG, GT, TT, TA}\}$. Secondly, the set SU is sorted according to the number of words in S_2 that it can concatenate to, maintaining the bond-free property. Table 5.6 sorts words $w \in SU$ by the number of words $v \in S_2$ such that $\text{Sub}_k(wv) \subset S$. Finally, set EN and BE are selected by evaluating the cardinality of $B_{S_2}(l)$ and all the cur_SU tested are listed in Table 5.7. By comparing

w	v	Size
TG	TTG TGT GTT TTT GTA	5
GT	TTG TGT GTT TTT GTA	5
TT	TTG TGT GTT TTT GTA	5
TA		0

Table 5.6: List of all the suffixes of S_2 , i.e., $w \in SU$, and the corresponding words v in S_2 satisfying $\text{Sub}_k(wv) \subseteq S$.

the values of $B_{S_2}(l)$ where $l = 5$, we obtain the $best_SU = \{TG, GT, TT\}$ and $BE = \{TTG, TGT, GTT, TTT, GTA\}$. Based on $best_SU$, EN is obtained as $\{TTG, TGT, GTT, TTT\}$.

cur_SU	cur_BE	$ B_{S_1}(5) $
{TG}	{TTG, TGT, GTT, TTT, GTA}	3
{TG, GT}	{TTG, TGT, GTT, TTT, GTA}	5
{TG, GT, TT}	{TTG, TGT, GTT, TTT, GTA}	9

Table 5.7: Given set S_2 , all the pairs of cur_SU and cur_BE we considered in the algorithm.

The example shown above is the way to construct a pair of sets EN and BE which satisfy properties P1 and P2. To meet property P3, we will modify the end set EN_1 of S_1 and the begin set BE_2 of S_2 . The algorithm is shown as follows:

Algorithm:

Input: S - the set satisfies $\theta(S) \cap H_{d_1}(S) = \emptyset$;

EN_1 - the best end set of S_1 ; and

BE_2 - the best begin set of S_2 .

Output: Sets $EN'_1 \subseteq EN_1$ and $BE'_2 \subseteq BE_2$ such that $\text{Sub}_k(EN'_1 \cdot BE'_2) \subseteq S$ and $|EN'_1|$ is maximal.

for each $BE_2[i] \in BE$

do find set $cur_EN_i \subseteq EN_1$ such that $\text{Sub}_k(cur_EN_i \cdot BE_2[i]) \subseteq S$

 Find i such that $|cur_EN_i|$ is the maximum

if the maximal number is 0, that is, $\forall w \in EN_1, v \in BE_2, \text{Sub}_k(wv) \not\subseteq S$

then return ERROR

$EN'_1 = \emptyset$ and $BE'_2 = \emptyset$

for $j \leftarrow 0$ to $|EN_1| - 1$

do if $\text{Sub}_k(EN_1[j] \cdot BE_2[i]) \subseteq S$

then $EN'_1 = EN'_1 \cup \{EN_1[j]\}$

for $j \leftarrow 0$ to $|BE_2| - 1$

do if $\text{Sub}_k(EN'_1 \cdot BE_2[j]) \subseteq S$

then $BE'_2 = BE'_2 \cup \{BE_2[j]\}$

Using this algorithm, we can ensure that the concatenation of EN'_1 and BE'_2 satisfies the bond-free property, and the cardinality of $|EN'_1|$ is maximal. Using the previous sets S , S_1 and S_2 as an example, the end set of S_1 is $EN_1 = \{\text{GGG}, \text{TGG}, \text{AGG}, \text{GTG}\}$ and the begin set of S_2 is $BE_2 = \{\text{TTG}, \text{TGT}, \text{GTT}, \text{TTT}, \text{GTA}\}$. The algorithm first finds a word $w \in BE_2$ and a corresponding subset $EN'_1 \subseteq EN_1$ such that $\text{Sub}_k(EN'_1 \cdot w) \subseteq S$ and $|EN'_1|$ is the maximum. Then, we obtain the word $w = \text{TTG}$ and $EN'_1 = \{\text{GGG}, \text{TGG}, \text{AGG}, \text{GTG}\}$. Secondly, according to EN'_1 , all the words v in BE_2 are selected to BE'_2 such that $\text{Sub}_k(EN'_1 \cdot v) \subseteq S$. In our the example, $BE'_2 = \{\text{TTG}, \text{TGT}, \text{GTT}, \text{TTT}, \text{GTA}\}$. Therefore, $EN'_1 = EN_1$ and $BE'_2 = BE_2$, that is, all the words $u \in EN_1$ and $v \in BE_2$ satisfy $\text{Sub}_k(uv) \subseteq S$.

5.3 Bond-free Languages: Experimental Results

In Section 5.2, we introduced the algorithm to select the set S . Using the same method mentioned in [10], we evaluate the set S by the cardinality of $B(l)$ which is a set of words of length l and satisfies $(B(l))^* \subset S^\otimes$.

Since the cardinality of $B(l)$ depends on the length k of words in the set S , the reverse-complement Hamming distance d_1 between words in the set S , and the initial word we chose to construct the set S , we first fix k and d_1 to evaluate the relationship between the initial word and $|B(l)|$. Table 5.8 gives the cardinality of B under various initial words and lengths l , when $k = 2$ and $d_1 = 0$. Table 5.9 gives the cardinality

$w \setminus l$	4	5	6	$w \setminus l$	4	5	6	$w \setminus l$	4	5	6	$w \setminus l$	4	5	6
AA	20	45	101	CA	20	45	101	GA	20	45	101	TA	20	45	101
AC	20	45	101	CC	20	45	101	GC	20	45	101	TC	20	45	101
AG	20	45	101	CG	20	45	101	GG	20	45	101	TG	2	4	7
AT	20	45	101	CT	20	45	101	GT	6	10	19	TT	6	10	19

Table 5.8: Given parameters $k = 2$, $d_1 = 0$ and mapping α , this table gives the size of set $B(l)$ under various initial words and various lengths l .

of B under various initial words and lengths l , when the parameters are $k = 3$ and $d_1 = 0$.

$w \backslash l$	4	5	6	$w \backslash l$	4	5	6	$w \backslash l$	4	5	6	$w \backslash l$	4	5	6
AAA	41	117	335	CAA	48	128	384	GAA	38	110	311	TAA	48	128	384
AAC	41	117	335	CAC	48	128	384	GAC	38	110	311	TAC	48	128	384
AAG	41	117	335	CAG	48	128	384	GAG	38	110	311	TAG	40	113	323
AAT	41	117	335	CAT	48	128	384	GAT	38	110	311	TAT	38	101	284
ACA	41	117	335	CCA	48	128	384	GCA	40	113	323	TCA	48	128	384
ACC	41	117	335	CCC	48	128	384	GCC	40	113	323	TCC	48	128	384
ACG	41	117	335	CCG	48	128	384	GCG	40	113	323	TCG	28	76	204
ACT	41	117	335	CCT	48	128	384	GCT	28	76	204	TCT	35	94	258
AGA	40	113	323	CGA	48	128	384	GGA	40	113	323	TGA	40	113	323
AGC	40	113	323	CGC	48	128	384	GGC	40	113	323	TGC	28	76	204
AGG	40	113	323	CGG	40	113	323	GGG	40	113	323	TGG	22	60	166
AGT	40	113	323	CGT	28	76	204	GGT	25	67	181	TGT	18	44	109
ATA	48	128	384	CTA	48	128	384	GTA	40	113	323	TTA	38	105	291
ATC	48	128	384	CTC	48	128	384	GTC	24	64	173	TTC	28	78	216
ATG	40	113	323	CTG	36	98	277	GTG	18	46	124	TTG	21	60	161
ATT	38	101	284	CTT	35	95	267	GTT	23	61	165	TTT	26	69	185

Table 5.9: Given $k = 3$, $d_1 = 0$ and mapping α , this table gives the size of set $B(l)$ under various initial words and lengths l .

Several observations are in order regarding the resulting set $B(l)$. With fixed parameters k and d_1 , distinct initial words may result in the same cardinality of $B(l)$. This may be because the set S is selected mainly according to the overlapping property, rather than the alphabetical order. Furthermore, it is not clear how to assign the initial word which will always generate the largest set $B(l)$. However, by checking the data, we find that if we fix the initial word with index 0 (If we use the mapping β , then the initial word with index 0 is GG...G), we will usually generate a fairly large set $B(l)$. Hence, we just fix the initial words with index 0.

Then we fix k and the length l of words in $B(l)$ to check the relationship between d_1 and $|B(l)|$. In Table 5.10, we fix parameters k to 3 and l to 6, then test $|B(l)|$ under various initial words and parameters d_1 . When d_1 increases, $|B(l)|$ will decrease and even become empty. This is easy to understand since the increase of d_1 will make the reverse-complement Hamming distance constraint more strict and the number of words satisfying the constraint will be reduced. Thus, the cardinality of $B(l)$ will decrease.

$w \setminus d_1$	0	1	2	$w \setminus d_1$	0	1	2	$w \setminus d_1$	0	1	2	$w \setminus d_1$	0	1	2
AAA	335	68	64	CAA	384	68	64	GAA	311	68	64	TAA	384	68	1
AAC	335	68	64	CAC	384	68	64	GAC	311	68	1	TAC	384	64	1
AAG	335	68	64	CAG	384	68	1	GAG	311	68	64	TAG	323	6	1
AAT	335	68	1	CAT	384	64	1	GAT	311	6	0	TAT	284	9	1
ACA	335	64	64	CCA	384	64	64	GCA	323	68	1	TCA	384	64	1
ACC	335	64	64	CCC	384	64	64	GCC	323	68	1	TCC	384	64	1
ACG	335	64	0	CCG	384	64	1	GCG	323	1	1	TCG	204	64	1
ACT	335	68	1	CCT	384	64	1	GCT	204	1	1	TCT	258	1	1
AGA	323	20	64	CGA	384	68	1	GGA	323	68	64	TGA	323	6	1
AGC	323	20	0	CGC	384	20	1	GGC	323	68	1	TGC	204	1	1
AGG	323	20	64	CGG	323	68	1	GGG	323	68	64	TGG	166	64	1
AGT	323	1	1	CGT	204	1	1	GGT	181	1	1	TGT	109	6	1
ATA	384	64	0	CTA	384	64	1	GTA	323	1	1	TTA	291	13	1
ATC	384	64	1	CTC	384	64	1	GTC	173	1	1	TTC	216	13	1
ATG	323	1	1	CTG	277	1	1	GTG	124	64	1	TTG	161	1	1
ATT	284	13	1	CTT	267	13	1	GTT	165	1	1	TTT	185	6	1

Table 5.10: Given $k = 3$, this table shows the cardinality of set $B(6)$ under various initial words and d_1 .

Since we are interested in finding proper parameters k and d_1 for constructing the multiple-hairpin structure, we check the cardinality of $B(l)$ as well as the words in the set S . Let $k = 3$ and the index of the initial word be 0 (by using the mapping β , the initial word is GGG). Varying the parameter d_1 , we may achieve distinct sets S , see Table 5.11.

d_1	S
0	{GGG, GGT, GGA, GGC, GTG, GTT, GTA, GTC, GAG, GAT, GAA, GCG, GCT, TGG, TGT, TGA, TGC, TTG, TTT, TTA, TAG, TAT, TCG, AGG, AGT, AGA, ATG, ATT, AAG, CGG, CGT, CTG}
1	{GGG, GGT, GGA, GTG, GTT, GTA, GAG, GAT, TGG, TGT, TTG, TTT, AGG}
2	{GGG, GGT, GTG, GTT, TGG, TGT, TTG, TTT}

Table 5.11: Given $k = 3$, mapping β and the initial word GGG, this table shows the words in S obtained under various d_1 .

Given words $w, v \in S$, we have that $h(w, \theta(v)) > d_1$. By observing Table 5.11, we see that if the parameter d_1 is too strict, that is, d_1 is close to k , the set S will only contain half of the DNA alphabet. When $k = 3$ and $d_1 = 2$, S only contains the DNA bases G and T. Also, when parameter $k = 4$ and $d_1 = 3$, S will only contains bases G and T. In order to maintain the DNA alphabet, when selecting d_1 , we will choose d_1 not larger than half of k . Therefore, if $k = 3$, d_1 equal to 0 and 1 are acceptable. If $k = 4$, d_1 equal to 0, 1 and 2 are acceptable.

5.4 Constructing a Multiple-hairpin Structure: Experiments

In this section, we provide and evaluate several example sets of sequences that are expected to fold into a four-consecutive-hairpin structure. Herein we always set the

length s of the stem and the external segment to 20, and the length l of loop to 7. In order to maintain a higher GC-ratio, when we select the set S , we always use the mapping β mentioned in Section 5.2.1. Furthermore, all the initial words considered here are GG...G.

Example Set 1: Given parameters $k = 3$, $d_1 = 1$, $d_2 = 0$, we obtain $S = \{GGG, TGG, AGG, GTG, TTT, GAG, GGT, TGT, GTT, TTT, GAT, GGA, GTA\}$, $S_1 = \{GGG, TGG, AGG, GTG, GAG, GGT, GAT, GGA\}$ and $S_2 = \{TTG, TGT, GTT, TTT, GTA\}$. For these sets $\theta(S) \cap H_1(S) = \emptyset$ and $H(S_1, S_2) > 0$.

For the sequences to be used to construct stems and the external segment, we apply the following additional combinatorial constraints: GC-content is ranging between $GC_{lower} = 50\%$ and $GC_{upper} = 80\%$, the continuity constraint $c = 5$ (there are not 5 consecutive same letter in any stems) and the Hamming distance constraint $d_3 = 10$ (the Hamming distance between any two stems is larger than 10). Thus, we produce B_{stem} of size 5. The words are shown in Table 5.12. For the sequences to be used to

Index i	v_i	$\theta(v_i)$
0	GAGGAGGAGGAGGAGGAGGG	CCCTCCTCCTCCTCCTCCTC
1	GAGGGAGGAGGAGGAGGAGG	CCTCCTCCTCCTCCTCCCTC
2	GGAGGAGGGAGGAGGAGGTG	CACCTCCTCCTCCCTCCTCC
3	GGGAGGAGGTGGGTGGGTGG	CCACCCACCCACCTCCTCCC
4	TGGGTGGTGGGTGGGTGGGTG	CACCCACCCACCCACCCCA

Table 5.12: The words in B_{stem} and $\theta(B_{stem})$ of Example Set 1 with $c = 5$ and $d_3 = 10$.

construct the loops, we apply the continuity constraint $c = 5$ and the end constraint. Thus, we obtain B_{loop} of size 9. The words are shown in Table 5.13. These sets of sequences can be used to construct the four-hairpin DNA structure and the expected four hairpin structure is shown in Figure 5.4.

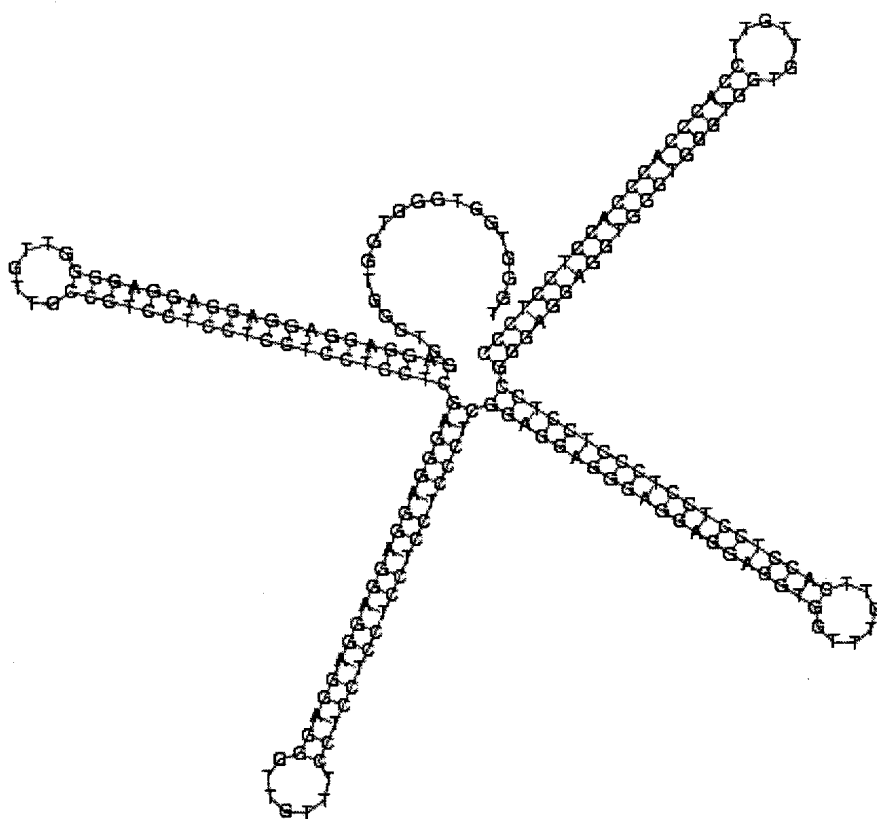


Figure 5.4: Correct DNA secondary structure with four consecutive hairpins.

Index i	y_i	Index i	y_i	Index i	y_i
0	GTTGTTG	3	TGTTGTT	6	TTGTTTG
1	GTTGTTT	4	TGTTTGT	7	TTTGTTG
2	GTTTGTT	5	TTGTTGT	8	TTTGTTT

Table 5.13: The words in B_{loop} of Example Set 1.

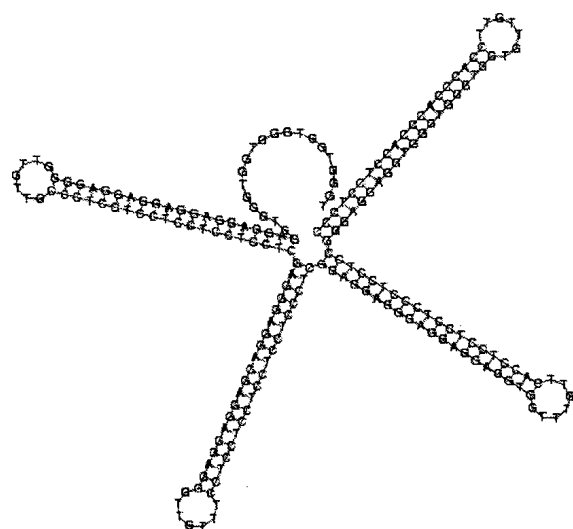
To conveniently denote the assignment of words to structure, we use the form $(index_1, index_2, \dots, index_{n+1})$ where $index_i$ indicates the words in B_{stem} and B_{loop} . To ease the process of the assignment, the last index in the form indicates the sequence assigned to the external segment. For example, by $(0, 1, 2, 3, 4)$, we denote the sequence $v_4v_0y_0\theta(v_0)v_1y_1\theta(v_1)v_2y_2\theta(v_2)v_3y_3\theta(v_3)$ where v_i and y_i are from Table 5.12 and Table 5.13, respectively.

By the method reported in Section 4.5, we can generate $\frac{5!}{(5-4-1)!} = 5! = 120$ sequences. The structures of all 120 sequences are predicted by Vienna RNA package [17] included in our software. The structures that the sequences fold into are close to or exactly the desired structure. If we evaluate the structures by the number of mismatches between the desirable structure and the structure of our sequences, we obtain the result in Table 5.14. Here, 16 sequences fold into the result structure correctly and 92 sequences fold into the desirable structure with only 2 mismatches.

# of mismatches	0	2	6	8
# of sequences	16	92	8	4

Table 5.14: The predictive results of Example Set 1.

In Figure 5.5, we give all distinct number of mismatches we obtained when we predict the structures. The worst cases of this example contain 8 wrong folding positions and they are assignments $(4, 0, 2, 1, 3)$, $(4, 0, 3, 1, 2)$, $(4, 1, 2, 0, 3)$ and $(4, 1, 3, 0, 2)$. By our observation, all of them have the same structures, therefore, we



(a) Correct structure.

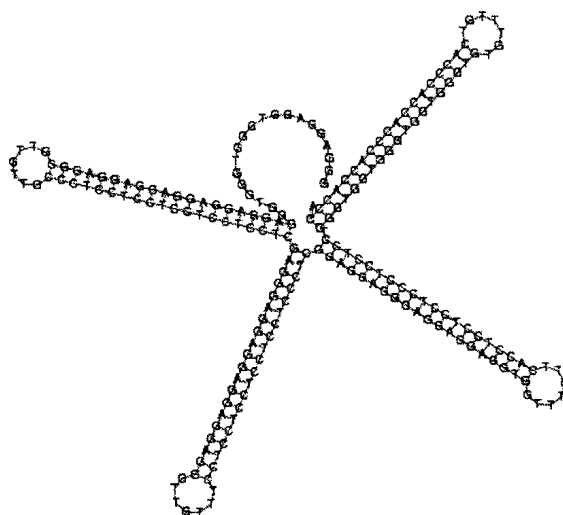
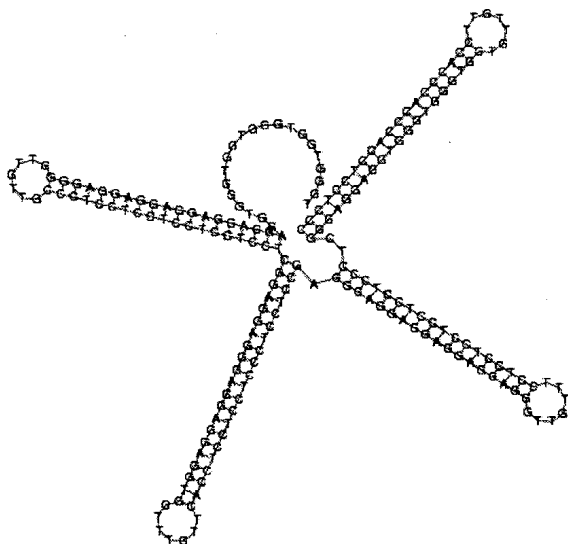
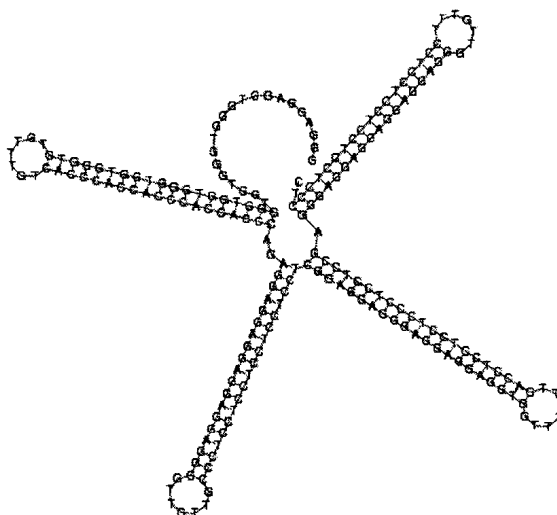
(b) $|\text{Mismatch}| = 2$ (c) $|\text{Mismatch}| = 6$ (d) $|\text{Mismatch}| = 8$

Figure 5.5: The structures of the result sequences predicted by the Vienna RNA package.

give the folding result of Assignment (4, 0, 2, 1, 3) as example, see in Figure 5.5(d). Even in the worst case, we still produce the four-hairpin structure and the undesirable bonds we emphasized are not shown in the structure. Therefore, we believe all 120 sequences can fold into the given four-hairpin structure.

If we change the Hamming distance constraint to $d_3 = 9$, then, we obtain a new set B_{stem} , see Table 5.15. We may obtain $|B_{stem}| = 7$ and we can generate 2520 sequences

Index i	v_i	$\theta(v_i)$
0	GAGGAGGAGGAGGAGGAGGG	CCCTCCTCCTCCTCCTCCTC
1	GAGGGAGGAGGAGGAGGAGG	CCTCCTCCTCCTCCTCCCTC
2	GAGGGGAGGAGGAGGAGGGG	CCCCTCCTCCTCCTCCCCTC
3	GGAGGAGGGGAGGGGTGGTG	CACCACCCCTCCCCTCCTCC
4	GGGAGGGTGGTGGGTGGTG	CCACCACCCACCACCCTCCC
5	TGGAGGTGGTGGGAGGGGTG	CACCCCTCCCACCACCTCCA
6	TGGGGTGGTGGGTGGGAGGG	CCCTCCCACCCACCACCCCA

Table 5.15: The words in B_{stem} and $\theta(B_{stem})$ of Example Set 1 with the Hamming constraint $d_3 = 9$.

and the corresponding result is shown in Table 5.16. By checking the result images, all the sequences are folding into the structures that avoid the undesirable bonds U1 - U4.

# of mismatches	0	2	4	6	8
# of sequences	144	1332	720	216	108

Table 5.16: The predictive results of Example Set 1 with $d_3 = 9$.

Example Set 2: Given parameters $k = 4$, $d_1 = 0$, $d_2 = 0$, the sets S , S_1 and S_2 satisfy $\theta(S) \cap H_0(S) = \emptyset$ and $H(S_1, S_2) > 0$. Here we apply other constraints to the

stems set as follows: the GC-content is ranging between 50% and 80%, the continuity constraint $c = 5$ and the Hamming distance constraint $d_3 = 11$. We list all the sequences of B_{stem} and part of the sequences of B_{loop} in Table 5.17. By evaluating

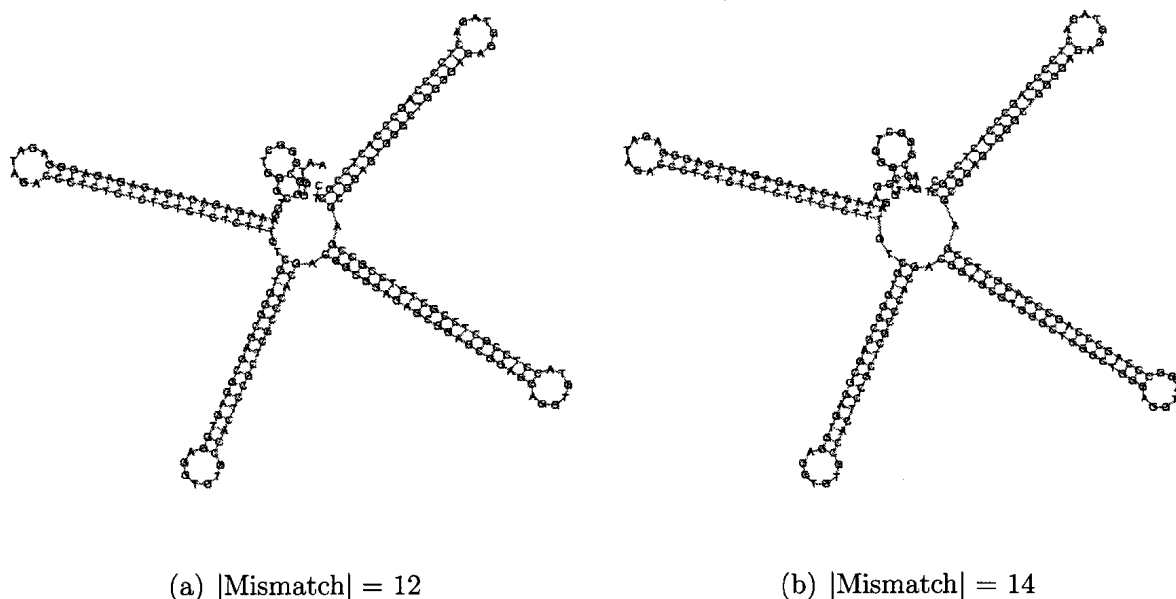
Index i	v_i	$\theta(v_i)$	y_i
0	AAAGAGAGAGAGAGAGAGGG	CCCTCTCTCTCTCTCTCTTT	AGATAGA
1	AAAGAGCGGAGAGAGAGAGG	CCTCTCTCTCTCCGCTCTTT	AGATAGG
2	AAGAGAGAGAGAGCGGAGAG	CTCTCCGCTCTCTCTCTCTT	AGATTGA
3	AAGAGCGGAGCGGAGCGTGG	CCACGCTCCGCTCCGCTCTT	AGATTGG
4	AATGGGGCGGGCTGGGCTGG	CCAGCCCAGCCCGCCCCATT	AGATTTG
5	AGAGCGGGCTGGGCTGGGAG	CTCCCAGCCCAGCCCCTCT	AGGATAG
6	ATGGGCGGGCTGGGAGCGTG	CACGCTCCCAGCCCGCCCAT	AGGATTG
7	GAGCGGAGTGGGCTGGGGAG	CTCCCCAGCCCCTCCGCTC	AGGTAGA
8	GGAGCGTGGGCTGGGCTGGG	CCCAGCCCAGCCCACGCTCC	AGGTAGG
9	GGCGGAGAGCGGAGCGGAGG	CCTCCGCTCCGCTCTCCGCC	AGGTGTA
10	GTCGTGGGCGAGCGGAGTGG	CCACTCCGCTCGCCCACGAC	AGGTGTG

Table 5.17: The words in B_{stem} , $\theta(B_{stem})$ and some of words in B_{loop} of Example Set 2.

the number of mismatches between the desirable structures and the structure of our sequences, we obtain the result in Table 5.18. When the number of mismatches is less than 6, all the result sequences will not contain any undesirable bonds. When the number of mismatches is larger than or equal to 6, the undesirable bond U1 may happen in external segments. The predicting structures when the number of mismatches equals to 12 and 14 are given in Figure 5.6.

# of mismatches	0	2	4	6	8	10	12	14
# of sequences	1200	13024	5819	11305	18254	5358	449	31

Table 5.18: The predictive results of Example Set 2.

Figure 5.6: Some structures of the result sequences, with $k = 4$, $d_1 = 0$ and $d_2 = 0$.

5.5 Discussions

In the previous section, we give successful examples of our proposed bottom-up approach. Here, we further discuss the selection of the parameters. In the bottom-up approach, the main parameters are the length of the block k in the set S , the reverse-complement Hamming distance d_1 between any two blocks in the structure, and the Hamming distance d_2 between sets S_1 and S_2 .

The parameter k should not be too large. Since our method is based on operations on word sets of length k , this is a time consuming process if the length of the block k is too large. As discussed in Section 5.3, if the reverse-complement Hamming distance d_1 is too large, the number of distinct bases will be too small. Therefore,

it may be reasonable to set the reverse-complement Hamming distance d_1 not larger than half of k . On the other hand, when d_1 is too small such as $k = 4$ and $d_1 = 0$ mentioned in the previous section, this may not prevent the undesirable bonds of blocks in the external segment x . For the parameter d_2 , we just make sure words in S_1 are different from the words in S_2 , since this may work well to prevent the undesirable bonds inside single hairpins.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, we have presented a bottom-up approach to construct a specific DNA secondary structure namely a consecutive-hairpin structure. As described in Section 3.5, by applying distinct keys, each hairpin in the multiple-hairpin structure can be opened one after another. Therefore, this multiple-hairpin structure can be used as the address segment of a memory wherein the data can be accessed by fully opening all the single hairpins. By our approach, we construct a set of sequences that fold in the initial state of the address segment of such a memory. For example, our software is able to generate 120 sequences that can each fold into a four-hairpin structure where the length of each hairpin stem is 20, the length of each hairpin loop is 7 and the length of each external segment is 20. All sequences have been verified by using the Vienna package.

Our method is different from other secondary structure design methods that construct a sequence mainly based on the evaluation of the free energy. By using the bottom-up approach, we use combinatorial constraints to construct the sequences such as the Hamming distance constraint (C1), the reverse-complement constraint (C2), the GC-content constraint (C4) and the continuity constraint (C5). Specifically, we consider the bond-free property defined [26]. By our approach, blocks of length k in the desirable structure are constructed first. These shorter blocks are then used

to construct the external segment x , the loop y_i and the stem v_i that avoid the undesirable bonds U1 and U2 mentioned in Section 4.1.

We construct the block set S by considering the reverse-complement constraint, i.e., $h(w, \theta(v)) > d_1, \forall w, v \in S$. To avoid the undesirable hybridizations of blocks within a single hairpin structure, we construct two subsets S_1 and S_2 from S by considering the Hamming distance constraint, i.e., $h(S_1, S_2) > d_2$. To make sure the concatenations of the external segment x , the stems v_i and the loops y_i such as xv_i and v_iy_i are still bond free, we construct special bond-free languages which are closed under concatenation. By using the software package Grail, we can generate the stem set $B_{S_1}(s)$ and the loop set $B_{S_2}(l)$ by constructing the corresponding automata of S_1 and S_2 . To avoid the undesirable bonds between the stems or between the loops, we apply the combinatorial constraints mentioned in Section 3.2.2 to filter out the undesirable sequences. Finally, the consecutive hairpin structure is obtained by the concatenations of the segments x, v_i, y_i and $\theta(v_i)$. By the implementation of our bottom-up approach, we successfully construct the set of sequences that fold into the multiple-hairpin structure and several examples are given in Section 5.4.

Compared with the existing algorithm [9] that generates bond-free languages, our algorithm has an improved method of selecting the set S (Section 5.2.1) which can generate a larger bond-free language $B(l)$ closed under concatenation. This is a crucial for our approach since set S is the fundamental set which is used to construct the sets S_1 and S_2 (Section 5.2.2) such that $H(S_1, S_2) > d_2$ and $S_1, S_2 \subset S$. In addition, recall that the optimal result [9] constructed a bond-free language that is closed under concatenation. Here, we expand that result to satisfy a more complex constraint: the construction of two separate bond-free language $B_{S_1}(s)$ and $B_{S_2}(l)$ such that all $(B_{S_1}(s))^*$, $(B_{S_2}(l))^*$ and $B_{S_1}(s) \cdot B_{S_2}(l)$ are still bond-free.

Finally, the original method proposed by Uejima and Hagiya [48] to construct the consecutive hairpin structure is primarily based on the evaluation of the free energy using a ‘‘trial and error’’ method. This method will result in a sequences that folds into

the consecutive hairpin structure and experiment result shows that this structure can indeed work as a multi-state machine. However, the method used by [48] to generate the multiple-hairpin structure seems problem specific, and it is also not clear whether it would scale-up. Our approach has the advantage that by changing the parameters, we can easily generate sequences suitable for a large variety of possible uses.

6.2 Future Work

In the following, we propose some possible improvements of our algorithms as well as a new application of our bottom-up approach.

In this thesis, our implementation of the bottom-up approach is focused on construct the sequence that will fold into the multiple-hairpin structure of the form

$$hps(n, s, l, \theta) = \{xv_1y_1\theta(v_1)\dots v_ny_n\theta(v_n) \mid x, v_i, \theta(v_i) \in \Sigma^s, y_i \in \Sigma^l \text{ and } 1 \leq i \leq n\}$$

In the structure, the loop is a short segment and the stem and the external segment are longer segments. Therefore, when we construct the words of the sets S_1 and S_2 which will be the blocks in the stems and the loop, respectively, we prefer to assign more words to S_1 to ensure the stability of base pairs in the stem. If the goal was another secondary structure, the way we implement the bottom-up approach may be thus very different.

By our bottom-up approach, we have successfully constructed the initial structure of the multi-state machine. Further effort is needed to make sure that the resulting multiple-hairpin structures can be opened one by one by applying distinct keys. A possible approach is to include the GC content constraint when we select the set S . Also, when we filter out the undesirable sequences in the set $B_{S_1}(s)$, it is good idea to maintain a word v_i if it has lower free energy when it forms double-stranded segment.

The software of the bottom-up approach we implemented does not include a graphical interface. Due to the large amount of data, it would be more convenient to evaluate both the sequences and the corresponding structures in a graphical interface.

In [44], the authors also propose another multiple-hairpin structure of the form

$$u' = x_1 v_1 y_1 \theta(v_1) x_2 v_2 y_2 \theta(v_2) \dots x_n v_n y_n \theta(v_n),$$

which is the concatenations of substructures $x_i v_i y_i \theta(v_i)$. Figure 6.1 shows an example

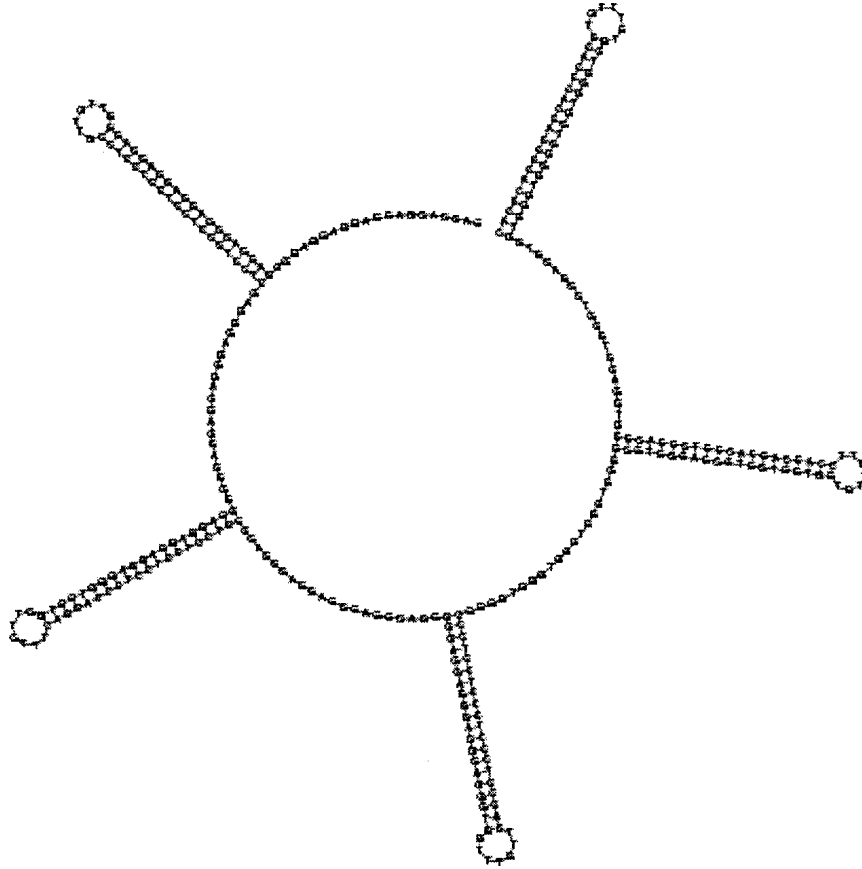


Figure 6.1: The new multiple-hairpin structure with five hairpins.

of this multiple-hairpin structure with 5 substructures $x_i v_i y_i \theta(v_i)$. This structure is very similar to the one we investigated, with the exception that segments x_i are inserted between each neighbor hairpin structures. By our bottom-up approach, these segments x_i can also be selected from B_{stem} . Using the algorithm we implemented, $(B_{stem})^*$ is $(\theta, H_{d_1, k})$ -bond-free, so the concatenation of the segment x_i and the stem v_i still satisfies the bond-free property. In this new application, it would be interesting

to investigate how large a structure we can construct, that is, what is the maximal number n of the substructures $x_i v_i y_i \theta(v_i)$ we can construct.

Bibliography

- [1] L. M. Adleman, *Molecular computation of solutions to combinatorial problems*, Science **266** (1994), no. 5187, 1021–1024.
- [2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data structures and algorithms*, Addison-Wesley, 1983.
- [3] M. Andronescu, A. P. Fejes, F. Hutter, H. H. Hoods, and A. Condon, *A new algorithm for RNA secondary structure design*, Journal of Molecular Biology **336** (2004), 607–624.
- [4] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, and E. Shapiro, *An autonomous molecular computer for logical control of gene expression*, Nature **429** (2004), 423–429.
- [5] S. Brenner, S. R. Williams, E. H. Vermaas, T. Storck, K. Moon, C. McCollum, J. Mao, S. Luo, J. J. Kirchner, S. Eletr, R. B. DuBridge, T. Burcham, and G. Albrecht, *In vitro cloning of complex mixtures of DNA on microbeads: Physical separation of differentially expressed cDNAs*, Proceedings of the National Academy of Sciences of the United States of America **97** (2000), no. 4, 1665–1670.
- [6] A. Busch and R. Backofen, *INFO-RNA - a fast approach to inverse RNA folding*, Bioinformatics **22** (2006), no. 15, 1823–1831.

- [7] A. Carbone and N. A. Pierce (eds.), *DNA Computing, 11th International Workshop on DNA Computing, DNA11, London, ON, Canada, June 6-9, 2005, Revised Selected Papers*, LNCS, vol. **3892**, Springer, 2006.
- [8] J. Chen and J. H. Reif (eds.), *DNA Computing, 9th International Workshop on DNA-Based Computers, DNA9, Madison, WI, USA, June 1-3, 2003*, LNCS, vol. **2943**, Springer, 2004.
- [9] B. Cui, *Encoding methods for DNA languages defined via the subword closure operation*, Master's thesis, Saint Mary's University, Halifax, Nova Scotia, Canada, May 2007.
- [10] B. Cui and S. Konstantinidis, *DNA coding using the subword closure operation*, in Garzon and Yan [15], pp. 284–289.
- [11] M. Domaratzki, *Hairpin structures defined by DNA trajectories*, in Mao and Yokomori [31], pp. 182–194.
- [12] U. Feldkamp, H. Rauhe, and W. Banzhaf, *Software tools for DNA sequence design*, *Genetic Programming and Evolvable Machines* **4** (2003), no. 2, 153–171.
- [13] U. Feldkamp, S. Saghafi, W. Banzhaf, and H. Rauhe, *DNASequenceGenerator: A program for the construction of DNA sequences*, in Jonoska and Seeman [21], pp. 23–32.
- [14] C. Ferretti, G. Mauri, and C. Zandron (eds.), *DNA Computing, 10th International Workshop on DNA Computing, DNA10, Milan, Italy, June 7-10, 2004, Revised Selected Papers*, LNCS, vol. **3384**, Springer, 2005.
- [15] M. H. Garzon and H. Yan (eds.), *DNA Computing, 13th International Meeting on DNA Computing, DNA13, Memphis, TN, USA, June 4-8, 2007, Revised Selected Papers*, LNCS, vol. **4848**, Springer, 2008.

- [16] M. Hagiya and A. Ohuchi (eds.), *DNA Computing, 8th International Workshop on DNA-Based Computers, DNA8, Sapporo, Japan, June 10-13, 2002, Revised Papers*, LNCS, vol. **2568**, Springer, 2003.
- [17] I. L. Hofacker, W. Fontana, P. F. Stadler, L. S. Bonhoeffer, M. Tacker, and P. Schuster, *Fast folding and comparison of RNA secondary structures*, Monatshefte für Chemie (Chemical Monthly) **125** (1994), 167–188.
- [18] S. Hussini, L. Kari, and S. Konstantinidis, *Coding properties of DNA languages*, in Jonoska and Seeman [21], pp. 57–69.
- [19] N. Iimura, M. Yamamoto, F. Tanaka, and A. Ohuchi, *Sequence design support system for 4×4 DNA tiles*, in Garzon and Yan [15], pp. 140–145.
- [20] N. Jonoska and K. Mahalingam, *Languages of DNA based code words*, in Chen and Reif [8], pp. 61–73.
- [21] N. Jonoska and N. C. Seeman (eds.), *DNA Computing, 7th International Workshop on DNA-Based Computers, DNA7, Tampa, FL, USA, June 10-13, 2001, Revised Papers*, LNCS, vol. **2340**, Springer, 2002.
- [22] A. Kameda, M. Yamamoto, H. Uejima, M. Hagiya, K. Sakamoto, and A. Ohuchi, *Conformational addressing using the hairpin structure of single-strand DNA*, in Chen and Reif [8], pp. 219–224.
- [23] L. Kari, *DNA computing: The arrival of biological mathematics*, The Mathematical Intelligencer **19** (1997), no. 2, 9–22.
- [24] L. Kari, R. Kitto, and G. Thierrin, *Codes, involutions, and DNA encodings*, Formal and Natural Computing (W. Brauer, H. Ehrig, J. Karhumäki, and A. Salomaa, eds.), LNCS, vol. **2300**, Springer, 2002, pp. 376–393.

- [25] L. Kari, S. Konstantinidis, E. Losseva, P. Sosík, and G. Thierrin, *Hairpin structures in DNA words*, in Carbone and Pierce [7], pp. 158–170.
- [26] L. Kari, S. Konstantinidis, and P. Sosík, *Bond-free languages: Formalizations, maximality and construction methods*, in Ferretti et al. [14], pp. 169–181.
- [27] L. Kari, S. Konstantinidis, P. Sosík, and G. Thierrin, *On hairpin-free words and languages*, *Developments in Language Theory* (C. De Felice and A. Restivo, eds.), LNCS, vol. **3572**, Springer, 2005, pp. 296–307.
- [28] D. E. Kephart and J. Lefevre, *CODEGEN: the generation and testing of DNA code words*, *Evolutionary Computation*, 2004. CEC2004. Congress on., vol. **2**, June 2004, pp. 1865–1873.
- [29] D. Kim, S. Shin, I. Lee, and B. Zhang, *NACST/Seq: A sequence design system with multiobjective optimization*, in Hagiya and Ohuchi [16], pp. 242–251.
- [30] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*, North-Holland, 1977.
- [31] C. Mao and T. Yokomori (eds.), *DNA Computing, 12th International Meeting on DNA Computing, DNA12, Seoul, Korea, June 5-9, 2006, Revised Selected Papers*, LNCS, vol. **4287**, Springer, 2006.
- [32] A. Marathe, A. Condon, and R. M. Corn, *On combinatorial DNA word design*, *Journal of Computational Biology* **8** (2001), no. 3, 201–219.
- [33] G. Mauri and C. Ferretti, *Word design for molecular computing: A survey*, in Chen and Reif [8], pp. 37–47.
- [34] K. U. Mir, *A restricted genetic alphabet for DNA computing*, DNA2, American Mathematical Society, 1996.

- [35] D. Raymond and D. Wood, *Grail: A C++ library for automata and expressions*, Journal of Symbolic Computation **17** (1994), 341–350.
- [36] J. H. Reif, T. H. LaBean, M. Pirrung, V. S. Rana, B. Guo, C. Kingsford, and G. S. Wickham, *Experimental construction of very large scale DNA databases with associative search capability*, in Jonoska and Seeman [21], pp. 231–247.
- [37] E. Rivas and S. R. Eddy, *A dynamic programming algorithm for RNA structure prediction including pseudoknots*, Journal of Molecular Biology **285** (1999), 2053–2068.
- [38] D. Sadava, H. C. Heller, G. H. Orians, W. K. Purves, and D. Hillis, *Life: The science of biology*, 8th ed., W. H. Freeman, 2008.
- [39] J. Sager and D. Stefanovic, *Designing nucleotide sequences for computation: A survey of constraints*, in Carbone and Pierce [7], pp. 275–289.
- [40] K. Sakamoto, D. Kiga, K. Komiya, H. Gouzu, S. Yokoyama, S. Ikeda, H. Sugiyama, and M. Hagiya, *State transitions by molecules*, BioSystems **52** (1999), 81–91.
- [41] J. SantaLucia, *A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics*, Proceedings of the National Academy of Sciences of the United States of America **95** (1998), no. 4, 1460–1465.
- [42] N. C. Seeman, *Design of sequences for nucleic acid structure engineering*, Journal of Biomolecular structure & Dynamics **8** (1990), no. 3, 573–581.
- [43] N. C. Seeman and N. R. Kallenbach, *Design of immobile nucleic acid junctions*, Biophysical Journal **44** (1983), 201–209.
- [44] N. Takahashi, A. Kameda, M. Yamamoto, and A. Ohuchi, *Aqueous computing with DNA hairpin-based RAM*, in Ferretti et al. [14], pp. 355–364.

- [45] F. Tanaka, M. Nakatsugawa, M. Yamamoto, T. Shiba, and A. Ohuchi, *Developing support system for sequence design in DNA computing*, in Jonoska and Seeman [21], pp. 129–137.
- [46] D. C. Tulpan, *Effective heuristic methods for DNA strand design*, Ph.D. thesis, The University of British Columbia, Vancouver, British Columbia, Canada, October 2006.
- [47] D. C. Tulpan, H. H. Hoos, and A. Condon, *Stochastic local search algorithms for DNA word design*, in Hagiya and Ohuchi [16], pp. 229–241.
- [48] H. Uejima and M. Hagiya, *Secondary structure design of multi-state DNA machines based on sequential structure transitions*, in Chen and Reif [8], pp. 74–85.
- [49] B. Wei, Z. Wang, and Y. Mi, *Uniquimer: A de novo DNA sequence generation computer software for DNA self-assembly*, in Mao and Yokomori [31], pp. 266–273.
- [50] M. Zuker, *Mfold web server for nucleic acid folding and hybridization prediction*, *Nucleic Acids Research* **31** (2003), no. 13, 3406–3415.