

**NOVEL ALGORITHMS  
FOR *IN VITRO* GENE SYNTHESIS**

by

Chris Thachuk

B.C.S. Hons [Co-op], University of Windsor, 2005

A THESIS SUBMITTED IN PARTIAL FULFILLMENT  
OF THE REQUIREMENTS FOR THE DEGREE OF  
MASTER OF SCIENCE  
in the School  
of  
Computing Science

© Chris Thachuk 2007  
SIMON FRASER UNIVERSITY  
Summer 2007

All rights reserved. This work may not be  
reproduced in whole or in part, by photocopy  
or other means, without the permission of the author.

## APPROVAL

**Name:** Chris Thachuk  
**Degree:** Master of Science  
**Title of thesis:** Novel Algorithms for *in vitro* Gene Synthesis

**Examining Committee:** Dr. Eugenia Ternovska  
Chair

---

Dr. Arvind Gupta  
Professor, Computing Science  
Simon Fraser University  
Senior Supervisor

---

Dr. Anne Condon,  
Professor, Computer Science  
University of British Columbia  
Supervisor

---

Dr. Ramesh Krishnamurti  
Professor, Computing Science  
Simon Fraser University  
Internal Examiner

**Date Approved:** \_\_\_\_\_

# Abstract

Methods for reliable synthesis of long genes offer great promise for novel protein synthesis via expression of synthetic genes. Current technologies use computational methods for design of short oligos, which can then be reliably synthesized and assembled into the desired target gene. A precursor to this process is optimization of the gene sequence for improved protein expression. In this thesis, we provide the first results on the computational complexity of oligo design for gene synthesis. For collision-oblivious oligo design – when mishybridizations between oligos are ignored – we give an efficient dynamic programming algorithm. We show that an abstraction of the collision-aware oligo design problem is NP-hard. We extend our collision-oblivious algorithm to achieve collision-aware oligo design, when the target gene can be partitioned into independently-assembled segments. We propose additional algorithms for gene optimization and demonstrate their utility for collision-aware design. All methods are evaluated on a large biological gene set.

**Keywords:** gene synthesis; oligo design; gene engineering; codon optimization; complexity

*For my parents, Barry and Jeannette Thachuk*

*“The genes are the master programmers, and they are programming for their lives.”*

— *Richard Dawkins*, THE SELFISH GENE

# Acknowledgments

I have many people to acknowledge and for that I am lucky. First and foremost, I would like to thank my supervisors, Anne Condon and Arvind Gupta. Thank you Anne for your constant support, understanding, generosity with your time and for helping me explore very interesting research areas and problems. It has been a truly wonderful experience working with you. Thank you Arvind for all your support during the past two years, whether research related or the excellent advice you have always given to me, with my best interests in mind. For that I am truly thankful. Thanks to you both for being great mentors, both professionally and personally.

Thank you to Holger Hoos for being my unofficial third supervisor, for the great conversations and ongoing advice. Thanks to Ramesh Krishnamurti for examining this thesis and his feedback. I would like to thank all members of the SFU COMBI lab, in particular, Ján Maňuch and Alireza Khodabakhshi for their feedback and collaboration related to this thesis. I would also like to thank the members of the  $\beta$ eta lab at UBC, especially Mirela Andronescu and Hosna Jabbari for their never ending help and feedback. Thank you to the CIHR/MSFHR Bioinformatics Training Program and MITACS for training and support throughout and a special thanks to Sharon Ruschkowski for all that you have done.

I would also like to thank those associated with the University of Wisconsin-Madison Center for NanoTechnology, Kathryn Richmond, Michael Shortreed, James Kaysen, Lloyd M. Smith and Franco Cerrina for introducing us to such an interesting problem, for their time, for their invaluable input and for shaping the practical goals of this research.

A special thanks to my family who have always encouraged me in everything I have done. Finally, thank you Meagan for your unwavering support and the happiness you give me.

# Contents

<b>Approval</b>	<b>ii</b>
<b>Abstract</b>	<b>iii</b>
<b>Dedication</b>	<b>iv</b>
<b>Quotation</b>	<b>v</b>
<b>Acknowledgments</b>	<b>vi</b>
<b>Contents</b>	<b>vii</b>
<b>List of Tables</b>	<b>x</b>
<b>List of Figures</b>	<b>xi</b>
<b>List of Algorithms</b>	<b>xvi</b>
<b>1 Background</b>	<b>1</b>
1.1 Existing Gene Synthesis Algorithms . . . . .	2
1.2 Thesis Objectives . . . . .	6
1.3 Contributions . . . . .	6
1.4 Thesis Outline . . . . .	7
<b>2 The Oligo Design for Gene Synthesis Problem</b>	<b>8</b>
<b>3 Collision Oblivious Oligo Design Algorithms</b>	<b>13</b>
3.1 Ungapped Oligo Design Algorithm . . . . .	13

3.1.1	Time and Space Complexity . . . . .	15
3.2	Gapped Oligo Design Algorithm . . . . .	15
3.2.1	Time and Space Complexity . . . . .	16
3.3	Experimental Analysis . . . . .	16
3.3.1	Experimental Environment . . . . .	18
3.3.2	Performance of the Collision Oblivious Algorithm . . . . .	18
<b>4</b>	<b>Complexity of Collision-Aware Oligo Design</b>	<b>23</b>
4.1	The Collision-Aware String Partition Problem . . . . .	24
4.2	Reducing 3SAT(3) to CA-SP . . . . .	26
4.3	A Restricted Version of CA-SP . . . . .	30
<b>5</b>	<b>Collision Aware Oligo Design Algorithms</b>	<b>31</b>
5.1	Greedy Collision Aware Algorithm . . . . .	31
5.2	Collision Aware Synthon Design . . . . .	32
5.2.1	Synthon Partition Algorithm . . . . .	33
5.2.2	Time and Space Complexity . . . . .	33
5.2.3	A Speedup for determining the collision graph . . . . .	34
5.3	Experimental Analysis . . . . .	34
5.3.1	Performance of the Greedy Algorithm . . . . .	34
5.3.2	Performance of the Synthon Partition algorithm . . . . .	35
<b>6</b>	<b>Codon Optimization</b>	<b>37</b>
6.1	Problem Definition and Overview . . . . .	37
6.1.1	Notation and Definitions . . . . .	38
6.1.2	Constraint Satisfaction . . . . .	39
6.1.3	Criterion Optimization . . . . .	39
6.1.4	The Codon Optimization Problem . . . . .	42
6.2	Codon Optimization Algorithms . . . . .	44
6.2.1	A Dynamic Programming Algorithm for CAI Optimization . . . . .	44
6.2.2	CDI Optimization Algorithm . . . . .	48
6.2.3	Designing Homologous Sequences . . . . .	59
6.3	Experimental Analysis . . . . .	59
6.3.1	Performance of the CAI optimization algorithm . . . . .	61

6.3.2	Performance of the CDI optimization algorithm . . . . .	61
6.3.3	Utility of homologous sequences for synthon design . . . . .	64
<b>7</b>	<b>Conclusions and Future Work</b>	<b>68</b>
<b>A</b>	<b>Comparison: Existing Gene Synthesis Algorithms</b>	<b>70</b>
A.1	Oligo Design Algorithms . . . . .	73
A.2	Codon Optimization Algorithms . . . . .	73
	<b>Bibliography</b>	<b>76</b>

## List of Tables

1.1	An overview of features for existing gene synthesis programs. . . . .	5
3.1	Results of the collision oblivious algorithm, using various design parameters, for 500 randomly selected sequences of the filtered GENECODE data set. . .	20
5.1	The greedy oligo design algorithm performs poorly in practice, finding few collision free designs and only succeeds for short sequences. . . . .	35
6.1	Performance for all 3,157 sequences, relative to REMC, is shown for each Monte Carlo search fixed at different temperatures. . . . .	64
6.2	A comparison of the minimum number of synthons required for designs of the set of homologs compared with the original sequence, for all 3,157 sequences. An improvement occurs when at least one of the homologs has an optimal collision-oblivious design requiring less synthons than the original sequence. .	67
A.1	Test sequences for algorithm comparison. . . . .	70
A.2	CDI score results for <i>UpGene</i> and our algorithm after five independent runs on each of the five test sequences. Our algorithm outperforms <i>UpGene</i> for all problem instances. . . . .	75

# List of Figures

2.1 An ungapped oligo design (top) and a gapped oligo design (bottom) are shown for the same input duplex  $S$ . An oligo design must have at least one oligo covering each position in  $S$  and the oligos alternately overlap between strands. For ungapped designs, there are no gaps between successive oligos on the same strand. The oligo  $O_b$  has *designed hybridizations* with oligos  $O_a$  and  $O_c$  as they share a *complementary overlap region*. . . . . 9

2.2 An oligo design containing an oligo likely to *self-hybridize* (bottom right) and a pair of oligos which potentially form an *oligo collision* (bottom left). The melting temperature of the MFE structure of  $O_k$  and the MFE duplex formed by  $O_i$  and  $O_j$  is denoted as  $Tm(O_k)$  and  $Tm(O_i, O_j)$ , respectively. . . . . 10

2.3 The *complementary overlap regions* are shown for each *designed hybridization* in the oligo design  $\mathcal{O}$ . In this design  $Tm(O_i \cap O_j) = t_4$ ,  $T_{min}(O_i) = \min(t_3, t_4)$ , and  $T_{min}(\mathcal{O}) = \min(t_1, t_2, t_3, t_4, t_5)$ . . . . . 11

3.1 The ungapped recurrence (top) attempts to find the best starting position  $j$  for an oligo ending at  $j'$ , given that there is an oligo ending at  $i'$  on the opposite strand. Position  $(j - 1)'$  immediately precedes  $j$  in an ungapped design. The gapped recurrence (bottom) also considers the best position for the oligo ending at  $(j - 1)'$ , given that it overlaps the oligo ending at position  $i'$ . . . . . 14

3.2 The percentage of valid designs (of the 500 sample sequences) at different values of  $t_{max}$  is reported for both the gapped and ungapped algorithm for different values of  $l_{max}$ . The gapped algorithm is much more successful at finding valid designs. . . . . 19

3.3 On the left, the runtime distribution of the 500 sampled sequences is shown for both the ungapped and gapped algorithm for two different maximum oligo lengths and  $t_{max} = 60$ . Performance between gapped and ungapped variants is virtually indistinguishable for the same value of  $l_{max}$ . A constant difference is observed between different values of  $l_{max}$ . On the right, the CPU runtime for each of the 500 sampled sequences is plotted against sequence length for the gapped algorithm when  $t_{max} = 60$  and  $l_{max} = 52$ . A lowess regression curve of best fit has been added. . . . . 21

3.4 The number of collisions per 100 bases is reported for the optimal collision oblivious designs of the sampled 500 sequences, at different values of  $t_{max}$ , using the gapped algorithm with  $l_{max} = 42$  on the left, and  $l_{max} = 52$  on the right. . . . . 22

4.1 Two partitions are shown for the string *theimportantthingisnevertostopquestioning*. The substrings in both partitions have maximum length 3. The partition shown above the string is valid, in that no two substrings are identical; however, the partition shown below the string is invalid as there are two cases of identical substrings indicated with arrows. . . . . 23

4.2 An invalid design (middle) containing two pairs of identical oligos. The identical oligos  $c$  and  $g$  could hybridize out of order resulting in two distinct fragments (top). Identical oligos from opposite strands must also be avoided as entire regions could be assembled out of order (bottom). . . . . 25

4.3 The construction of string  $A = A'A''A'''$  for an instance  $\phi$  of 3SAT(3), with  $X = \{x_1, x_2, x_3\}$  and  $C = \{(\neg x_2, x_1, \neg x_3), (\neg x_1), (x_1, x_2)\}$ . Selections outlined with a box are required to avoid a collision. All potential partitions of clause strings, in  $A'$ , and forbidden strings, in  $A'''$ , are shown. . . . . 27

4.4 For a variable  $x_v$  having three literals  $\alpha_i^a, \alpha_j^b, \alpha_k^c$ , with  $(a_k^c = \neg a_i^a) \wedge (a_k^c = \neg a_j^b)$ , four subsets of the literals can possibly be selected in  $A'$ . The enforcer string guarantees two opposite literals cannot simultaneously be selected in  $A'$ . For instance, their are four valid partitions of the enforcer string when  $\alpha_j^b$ , is selected in  $A'$ . In all four of these partitions, the opposite literal,  $\alpha_k^c$  is selected and therefore cannot be selected in  $A'$  without introducing a collision. Similar arguments follow for the other three combinations of selecting literals of a variable in  $A'$ . . . . . 28

5.1 An oligo design with collisions denoted by edges (top) is transformed into an oligo collision graph (middle) with designed hybridizations shown with solid edges and collisions shown with dashed edges. A minimum-size partition of the graph, representing synthons, is shown (bottom). . . . . 32

5.2 For moderate values of  $t_{max}$ , very few synthons are required to make a design collision free. Results for  $l_{max} = 42$  are shown on the left and results for  $l_{max} = 52$  are shown on the right. . . . . 36

6.1 An example of an Aho-Corasick automaton. The automaton is for the set of patterns  $\mathcal{F} = \{ \text{eye, he, her, iris, is, their} \}$ . Output nodes are shown as boxes and failure links as dashed edges (failure links leading to the root are not shown). The example has been recreated based on an example found in Dori and Landau [7]. . . . . 40

6.2 An example of an unreachable state when  $\delta < k + 1$ . Shown above is an instance consisting of four amino acids, a forbidden set  $\mathcal{F} = \{ CGC, CGA, CCG, ACC, TAG \}$  and therefore  $k = 1$  and  $\delta = 1$  (by our assumption). Arginine has six corresponding codons, however, three of them appear in  $\mathcal{F}$  and are shown with red boxes. Adjacent codons, those following logically next in the sequence, are shown connected by an edge. Two adjacent codons are shown connected by a bold edge if and only if their concatenation does not introduce a forbidden motif. There are only three valid codon assignments for this problem instance. If the initial codon assignment is the assignment shown in the first row, then there is no valid transition to the other two states without introducing a forbidden motif since  $\delta < 2$ . . . . . 52

- 6.3 A counter example to the claim that a  $\delta$ -exchange search neighbourhood is guaranteed to be complete for a problem instance of size  $m$ , with  $1 \leq \delta < m$  and  $k \geq 1$ . Shown above is an instance consisting of four Arginine amino acids, a forbidden set  $\mathcal{F} = \{CGC, CGA, CGG, AGA, GCG, GTA\}$  and therefore  $k = 1$ . Arginine has six corresponding codons, however, four of them appear in  $\mathcal{F}$  and are shown with red boxes. Adjacent codons are shown connected by an edge. Two adjacent codons are shown connected by a bold edge if and only if their concatenation does not introduce a forbidden motif. Clearly, there are only two valid codon assignments for this problem instance, and when  $\delta < 4$ , there is no possible transition from one to the other. . . . . 60
- 6.4 Evaluation of the CAI dynamic programming algorithm for each of the 3,157 sequences of the filtered GENECODE data set using a forbidden motif set  $\mathcal{F}$  described in the text. On the left, the optimal CAI value for each sequence is plotted against sequence length (shown in log scale). On the right, CPU runtime performance is plotted for each sequence, against its length and a lowess regression curve of best fit is added. . . . . 62
- 6.5 Evaluation of the CDI SLS optimization algorithm for each of the 3,157 sequences of the filtered GENECODE data set using a forbidden motif set  $\mathcal{F}$  described in the text. All data points represent the mean of ten independent runs having a maximum runtime of 15 seconds. On the left, the difference between the observed CDI value and the theoretical lower bound is shown plotted against sequence length (shown in log scale). On the right, the runtime distribution of all 3,157 sequences is shown. . . . . 63
- 6.6 The relative difference in mean CDI values for all 3,157 sequences of the filtered GENECODE data set is reported for the REMC algorithm and a Monte Carlo algorithm fixed at a temperature of 1.0 (left) and a Monte Carlo algorithm fixed at a temperature of 100.0 (right). The Monte Carlo algorithm outperforms the REMC algorithm in only one instance (when temperature is fixed at 1.0) and is shown with a filled light circle. As sequence length increases, performance difference between REMC and MC algorithms diminishes. 65

6.7 A comparison between the number of synthons required for optimal collision-oblivious designs of original sequences and the best design found amongst a corresponding set of homologs. On the left, results for designs having maximal oligo length of 42 and homologs having a sequence difference of at least 10% is shown. On the right, results for designs having maximal oligo length of 52 and homologs having a sequence difference of at least 25% is shown. . . . . 66

A.1 A comparison of distributions for overlap melting temperature regions and for self-hybridization melting temperatures is shown for oligo designs of four different oligo design algorithms on a 1.6kb sequence. . . . . 74

# List of Algorithms

3.1	Collision oblivious dynamic programming algorithm. Pseudo-code is given for determining the optimal score of both gapped and ungapped oligo designs, for two different design goals. . . . .	17
6.1	Calculating the CDI lower bound for a sequence. . . . .	43
6.2	Dynamic programming algorithm for optimizing CAI. . . . .	46
6.3	A stochastic local search based algorithm for optimizing codon bias with respect to the CDI measure. . . . .	53
6.4	A replica exchange Monte Carlo algorithm to optimize codon bias with respect to the CDI measure. . . . .	54
6.5	A Monte Carlo algorithm to optimize codon bias with respect to the CDI measure. . . . .	55

## Chapter 1

# Background

Gene synthesis – efficient construction of long protein-coding or RNA-coding DNA strands – is emerging as an important technology for genomics and synthetic biology. Synthetic genes can be used to express target proteins of interest in a host cell, making it possible to produce protein fragments of a size manageable for structural analysis, to understand how variations in protein sequence affects binding properties of the protein, and to design novel proteins [5]. Use of a designed synthetic gene which codes for the target protein, rather than a naturally occurring gene, can also enhance the gene’s expression level in the host, for example by matching the codon bias with that of the host in which the gene is expressed [15] or by removing introns from the gene [17]. Synthetic genes can also be designed to be resistant to RNA interference in the cell [21]. While most existing studies have focused on the design of a single gene, need has been demonstrated for the ability to cheaply synthesize multiple genes for use in the study of regulatory pathways [39].

Recently, Lartigue *et. al* have taken a step past synthetic gene design, towards synthesis of entire genomes [23]. In their study, the authors introduced an entire circular genome of the bacterium *Mycoplasma mycoides* large colony (LC) into a cell of the bacterium *Mycoplasma capricolum*. Daughter cells effectively became identical to the donor bacterium, both genetically and phenotypically. In an article featuring the achievement, Pennisi [27] states the group is among several who are attempting to assemble a minimum genome to support life, with the end goal of adding desirable genes, such as ones for making biofuels. Although there are many factors which will need to be considered to achieve this ambitious goal, one required component is a reliable means for gene synthesis, which can scale to synthesis of entire genomes.

Several methods for synthesis of small genes or gene fragments (ranging from less than a hundred up to a thousand or so bases in length) are currently in use. For example, in assembly PCR [35], short oligos are selected which cover the desired gene duplex, with overlaps between successive oligos on the complementary (so-called sense and anti-sense) strands of the duplex. The oligos are synthesized separately, and are pooled in solution. Assembly of the oligos is achieved via hybridization of overlapping oligos on the sense and anti-sense strands. PCR extension is used to fill in any gaps in the assembly, and to amplify the product. Emerging technologies aim to significantly improve the scale and reliability of gene synthesis, enabling synthesis of genes of length 10kb (kilobases) and ultimately up to 100kb, as well as multiplexed synthesis of sets of genes [39]. These technologies typically avoid the traditional high cost of individual oligo synthesis, by the use of parallel synthesis of primer-tagged oligos on photolithographic microarrays, followed by amplification and cleavage of primers. Further array-based hybridization techniques are used to identify and remove oligos with incorrect base composition due to synthesis errors.

The success of all of these methods relies in part on properties of the selected short oligos. This leads to a computational *oligo design problem*: given an input DNA duplex, select oligos from both the sense and antisense strands, so as to satisfy the following conditions. First, the oligos should *cover* the duplex – if oligos are ordered by distance from one end of the duplex, they should alternate between sense and antisense strands, with some overlap between successive oligos. Second, the oligos should yield *no synthesis and assembly errors*: an oligo should not fold on itself or stably hybridize to any oligo, other than those which it overlaps in the covering of the duplex. As the scale of gene synthesis grows, the computational design of short oligos becomes more critical to the success of gene synthesis technologies. Typically, there is some flexibility in the oligo lengths, and so there are exponentially many possible designs.

## 1.1 Existing Gene Synthesis Algorithms

There are already many algorithms and software packages available for oligo design [5, 17, 19, 29, 30, 40, 41]. These methods vary in the types of design criteria they support, and in the underlying design optimization techniques (as well as in other aspects not considered further here, such as the user interface). However, the literature provides little or no insight on the computational complexity of the many interesting variations of the problem of designing

oligos for gene synthesis. Moreover, the techniques reported in the literature are all heuristic in nature, but there are no empirically-obtained insights on the relative efficiency of the design techniques used. Because of this, we focus next on the most common design criteria that have been proposed in the literature.

To ensure reliable synthesis and assembly, designed oligos which cover a DNA duplex should satisfy one or more of the following criteria:

*Limited length range* [5, 39, 19, 29, 30, 40, 43]: The length of each oligo should fall within a short specified range (typically between  $\pm 4$  and  $\pm 10$  nucleotides from a given length, which can be from 40 to 70 nucleotides); or a maximum length is specified.

*Structure-free* [17, 39, 19]: Each selected oligo should not self-hybridize to form secondary structure that would interfere with the oligo synthesis. If tags, such as primers or restriction sites, are concatenated to oligos during the synthesis phase (*e.g.*, for amplification purposes), the oligo-tag concatamers should be structure-free. Various tests of structure-freeness are used, including detection of palindromic sequences or use of secondary structure prediction software.

*Uniform melting temperature ( $T_m$ )* [17, 19, 30]: The thermodynamic melting temperature ( $T_m$ ) of duplexes formed from successive overlapping oligo fragments should fall within a narrow range.

*Collision-free* [39, 30]: Oligos on one strand of the duplex should bind specifically to overlapping oligos on the complementary strand, and avoid *collisions*, whereby a pair of oligos from different parts of the duplex hybridize stably.

*Codon-optimized* [39, 19, 40]: Any codon in the input DNA sequence may be replaced by a codon that codes for the same amino acid, so that the overall frequency of codons is approximately as specified by the user. Typically, codon optimization is done in order to create a gene whose codon frequencies (bias) matches those of the host in which the gene will be expressed. However, other reasons for codon optimization include the design of oligos that satisfy other design criteria, such as being collision-free [19] and the removal of restriction sites [40].

*Synthon-informed* [17, 19]: Jayaraj *et al.* [19] propose that assembly of a long gene be done in two phases. First, short oligos are assembled into segments, called synthons. Then, larger sequences are assembled from synthons. With this approach, selected oligos for the whole

*CHAPTER 1. BACKGROUND*

4

gene need not be collision free; rather it is sufficient that with each synthon, selected oligos are collision free. Similarly, DNAWorks [17] uses simulated annealing to optimize design of “sections”. Synthon or section boundaries may also shift during optimization phase, allowing flexibility in the design process.

In Table 1.1, an overview is given, with respect to the above features, of existing programs found in the literature. We note that certain programs provide additional features, such as robot automation scripts, graphical user interfaces and overall lab process design, however, we limit our discussion to those criteria previously listed. Also, there are a number of software packages available solely for codon optimization. We detail these algorithms and their features in Chapter 6. Overall, no current software package provides robust algorithms for the majority of design tasks related to gene synthesis.

	Synthon Support	Oligo Lengths	Codon Optimization	Oligo Structure Detection	Cross-hybridization Detection	Uniform Tm
DNAWorks [17]	Yes	user input	Yes	limited	No	Yes
Gene2Oligo [29]	Yes	user input	No	limited (checks for palindromes)	limited (checks for repeats)	Yes
GeMS [19]	Yes	40 bases	Yes	limited (no automatic elimination)	No	limited
Protein Fabrication Automation [5]	No	50-60 bases	Yes	No	No	No
Assembly PCR oligo maker [30]	No	40-70 bases	No	No	No	limited (not automated)
Gene Designer [40]	No	user designed (no automated oligo design)	Yes	limited (detects palindromes)	limited (detects repeat regions)	limited (not automated)

Table 1.1: An overview of features for existing gene synthesis programs.

## 1.2 Thesis Objectives

The objectives of this thesis are threefold. First, we aim to rigorously define the associated problems involved in gene synthesis. While there have been many proposed algorithms for these problems, the lack of formal definitions, problem constraints and optimization criteria make the comparison of methods difficult. Second, given formal definitions of the problem, we seek to develop algorithms, whenever possible, which are complete. That is, the algorithms have a guarantee to find an optimal solution (by our definition), if one exists. Finally, we must ensure any algorithm we develop is efficient. As the need for synthesizing entire genomes rises, reliable, yet scalable algorithms will be necessary to aid in the design process.

## 1.3 Contributions

We now describe the contributions of this thesis, in the order they are discussed:

1. Despite numerous algorithms proposed for the oligo design for gene synthesis problem, no rigorous definition could be found in the literature. We provide the first formal problem definitions by distinguishing between two variants, collision-oblivious and collision-aware oligo design. This problem dichotomy is the first step towards understanding which design tasks are computationally tractable.
2. A linear time and space dynamic programming algorithm is presented, which is guaranteed to find an optimal solution to the collision-oblivious design problem, if one exists. This is the first algorithm for this problem to have this guarantee. All existing algorithms rely heavily on heuristics.
3. We present evidence that the collision-aware oligo design problem is  $\mathcal{NP}$ -hard by showing the collision-aware string partition problem, one with similar design goals, is  $\mathcal{NP}$ -hard. This is the first result on the complexity of this problem.
4. Prompted by evidence that oligo collisions occur infrequently in practice, we present a simple heuristic algorithm for partitioning an optimal collision-oblivious oligo design into collision free regions.
5. To date, the most efficient algorithm for optimizing the codon adaptation index value of a gene [33], under the restriction that forbidden motifs must be removed, is  $\Theta(n^2)$

where  $n$  is the length of the sequence [31]. We present a linear time and space algorithm for this task and provide a proof that the algorithm is correct.

6. Existing algorithms for optimizing the codon bias of a gene, relative to naturally occurring codon frequencies, and under the restriction that forbidden motifs must be removed, are all heuristic in nature. We propose a stochastic local search algorithm for this purpose and provide evidence, both theoretically and empirically, that our algorithm outperforms the current state-of-the-art.
7. We discuss the utility of designing multiple, homologous gene sequences for the purpose of collision-aware oligo design. We demonstrate that in practice, it is common that a collision-oblivious design of a homolog can be partitioned into a less number of synthons than the design of the original sequence.

## 1.4 Thesis Outline

A thorough understanding of common constraints related to each variant of the oligo design problem – collision-oblivious and collision-aware – is necessary to understand the distinctions made in our proposed algorithms. Therefore, we have devoted the whole of Chapter 2 for formal definitions of these problems and to make note of their differences. We detail our linear time design algorithm for the collision-oblivious problem in Chapter 3. An experimental analysis of the algorithm is conducted on a large biological data set to determine its effectiveness and runtime efficiency. We end the chapter with an analysis of the frequency of oligo collision occurring in these collision-oblivious designs. Chapter 4 presents evidence the collision-aware oligo design problem is intractable in practice, if  $\mathcal{P} \neq \mathcal{NP}$ , by showing a related problem to be  $\mathcal{NP}$ -hard. In Chapter 5, we present an algorithm to partition a collision-oblivious design into a minimal number of collision-free regions. In Chapter 6, we formally define the problem of codon optimization and note common problem constraints and optimization criteria. We propose two algorithms to solve different versions of the problem, while extending one of them to find a set of solutions under an additional maximum sequence similarity constraint. We end the chapter with a rigorous experimental analysis and demonstrate the utility of codon optimization for the collision-aware oligo design problem. We discuss the conclusions of this thesis and the potential for future work in Chapter 7.

## Chapter 2

# The Oligo Design for Gene Synthesis Problem

In this chapter, we formally define the problem of oligo design for gene synthesis, along with some useful notation. A *DNA strand*, or *oligo*, is a string over alphabet  $\{A, C, G, T\}$ . We consider the left and right ends of the string to represent the 5' and 3' ends, respectively, of the corresponding physical DNA strand. A *DNA duplex*  $S$  consists of two complementary DNA strands distinguishable by a value  $\sigma$ ;  $S^\sigma$  refers to the sense strand when  $\sigma = 0$  and the anti-sense strand when  $\sigma = 1$ . The complement  $S^{1-\sigma}$  of DNA strand  $S^\sigma$  is obtained from  $S^\sigma$  by replacing each A with a T and vice versa, each C with a G and vice versa, and reversing the resulting string. Thus, for example, the *complement* of a strand having a sequence AATGGG is CCCATT. In this manner, for some sense strand  $S^0 = s_1s_2 \dots s_{|S|}$ , we let the oligo  $O_{a,b}^0$  denote the substring  $s_a \dots s_b$  and  $O_{a,b}^1$  denote the substring  $\overline{s_a \dots s_b}$ , the complement of  $s_a \dots s_b$ .

Informally, an oligo design is simply a covering of a DNA duplex by a set of oligos which alternately overlap between the sense and anti-sense strands. For an example, consider two potential oligo designs for the same duplex in Fig. 2.1; the top oligo design is ungapped whereas the bottom design is gapped. Next, we formally define what is an oligo design.

**Definition 2.0.1.** For a fixed DNA duplex  $S$ , a gapped oligo design  $\mathcal{O}$  is specified by two strictly monotonically increasing sequences of indices:  $i_1, i'_1, i_2, i'_2, \dots, i_x, i'_x$  and  $j_1, j'_1, j_2, j'_2, \dots, j_y, j'_y$ , such that:

- $|x - y| \leq 1$ ,

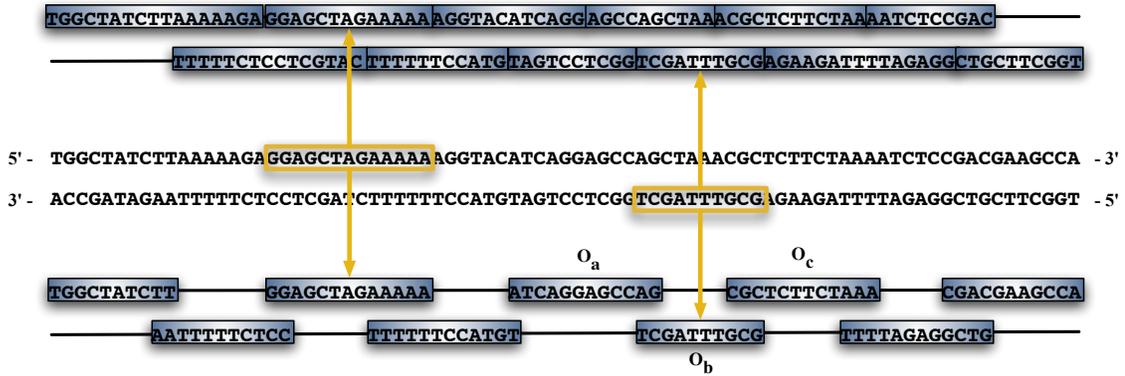


Figure 2.1: An ungapped oligo design (top) and a gapped oligo design (bottom) are shown for the same input duplex  $S$ . An oligo design must have at least one oligo covering each position in  $S$  and the oligos alternately overlap between strands. For ungapped designs, there are no gaps between successive oligos on the same strand. The oligo  $O_b$  has *designed hybridizations* with oligos  $O_a$  and  $O_c$  as they share a *complementary overlap region*.

- $\min(i_1, j_1) = 1$  and  $\max(i_x, j_y) = |S|$ ,
- the indices from the sequences alternately overlap in the following manner:
  - if  $i_1 = 1$  then  $x \geq y$  and
    - $i_k < j_k \leq i'_k < j'_k$ ,  $1 \leq k \leq y$
    - $j_y < i_x \leq j'_y < i'_x$ , if  $x > y$
  - if  $j_1 = 1$  then  $y \geq x$  and
    - $j_k < i_k \leq j'_k < i'_k$ ,  $1 \leq k \leq x$
    - $i_x < j_y \leq i'_x < j'_y$ , if  $y > x$

The set of oligos corresponding to  $\mathcal{O}$  is

$$\begin{aligned} \text{set}(\mathcal{O}) = & \{O_{i_k, i'_k}^0 = s_{i_k} \dots s_{i'_k} \mid 1 \leq k \leq x\} \\ & \cup \{O_{j_k, j'_k}^1 = \overline{s_{j_k} \dots s_{j'_k}} \mid 1 \leq k \leq y\}. \end{aligned}$$

An ungapped oligo design is a restricted version of a gapped oligo design where  $i_k = i'_{k-1} + 1$ ,  $1 < k \leq x$ , and  $j_k = j'_{k-1} + 1$ ,  $1 < k \leq y$ .

The remaining definitions are with respect to a fixed DNA duplex  $S$  and oligo design  $\mathcal{O}$  for  $S$ .

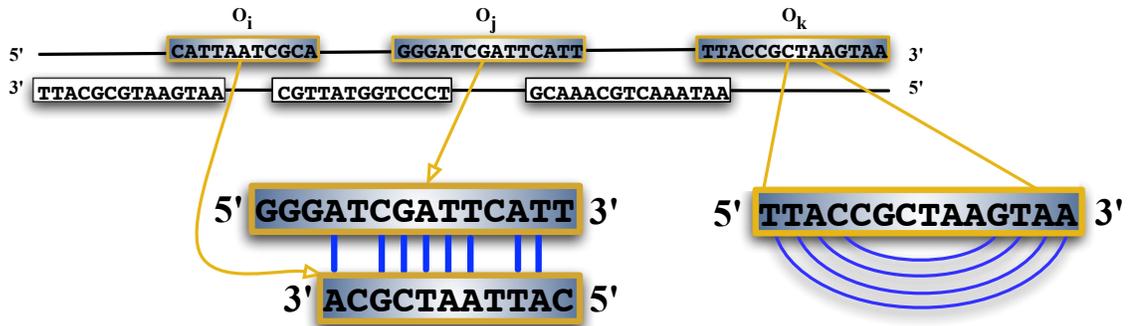


Figure 2.2: An oligo design containing an oligo likely to *self-hybridize* (bottom right) and a pair of oligos which potentially form an *oligo collision* (bottom left). The melting temperature of the MFE structure of  $O_k$  and the MFE duplex formed by  $O_i$  and  $O_j$  is denoted as  $Tm(O_k)$  and  $Tm(O_i, O_j)$ , respectively.

**Definition 2.0.2.** A designed hybridization is a pair of oligos  $O_{a,b}^\sigma, O_{c,d}^{1-\sigma} \in \text{set}(\mathcal{O})$  where either  $a < c \leq b < d$  or  $c < a \leq d < b$ . These oligos share a complementary overlap region (see Fig. 2.1).

Let  $Tm(O, O')$  and  $Tm(O)$  be the melting temperatures of the MFE (minimum free energy) duplex secondary structure formed by oligos  $O$  and  $O'$ , and of the secondary structure formed by  $O$  alone, respectively (see Fig. 2.2). Let  $Tm(O_{a,b}^\sigma \cap O_{c,d}^{1-\sigma})$  be the melting temperature of the complementary overlap region associated with a designed hybridization between  $O_{a,b}^\sigma$  and  $O_{c,d}^{1-\sigma}$  (see Fig. 2.3).

Finally, for any oligo  $O \in \text{set}(\mathcal{O})$ , let  $T_{min}(O)$  and  $T_{min}(\mathcal{O})$  be the minimum of all the melting temperatures of designed hybridizations involving  $O$ , and overall in  $\mathcal{O}$ , respectively. An example is given in Fig. 2.3.

**Definition 2.0.3.** An oligo design  $\mathcal{O}$  is:

- length range limited if all oligos in  $\text{set}(\mathcal{O})$  have length in the range  $[l_{min}, l_{max}]$
- structure free if for all oligos  $O \in \text{set}(\mathcal{O})$ ,  $Tm(O) \leq t_{sh}$ , the threshold for self hybridization
- designed Tm satisfied if for all oligo pairs  $(O_{a,b}^\sigma, O_{c,d}^{1-\sigma}) \in \mathcal{O}$  having a designed hybridization,  $t_{min} \leq Tm(O_{a,b}^\sigma \cap O_{c,d}^{1-\sigma}) \leq t_{max}$ , where  $t_{min}$  and  $t_{max}$  are the threshold of the minimum and maximum overlap region melting temperature, respectively

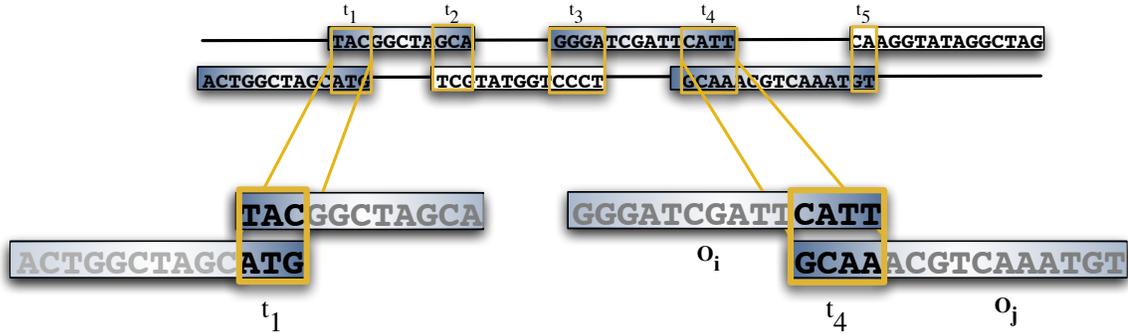


Figure 2.3: The *complementary overlap regions* are shown for each *designed hybridization* in the oligo design  $\mathcal{O}$ . In this design  $T_m(O_i \cap O_j) = t_4$ ,  $T_{min}(O_i) = \min(t_3, t_4)$ , and  $T_{min}(\mathcal{O}) = \min(t_1, t_2, t_3, t_4, t_5)$ .

**Definition 2.0.4.** Given length range  $[l_{min}, l_{max}]$  and temperatures  $t_{sh}$ ,  $t_{min}$  and  $t_{max}$ ,  $\mathcal{O}$  is valid if and only if it is length range limited, structure free and designed  $T_m$  satisfied. Let  $\mathcal{D}(S)$  be the set of all valid oligo designs for  $S$ .

**Definition 2.0.5.** Given threshold  $t_{col}$ ,  $\mathcal{O}$  is collision free if, for any pair of oligos  $(O, O')$  that do not have a designed hybridization,  $T_m(O, O') + t_{col} \leq \min(T_{min}(O), T_{min}(O'))$ . If  $T_m(O, O') + t_{col} > \min(T_{min}(O), T_{min}(O'))$ , we say that  $(O, O')$  is an oligo collision.

**Note 2.0.6.** An alternative definition is that for any pair of oligos  $(O, O')$  that do not have a designed hybridization,  $T_m(O, O') + t_{col} \leq T_{min}(\mathcal{O})$ . The latter definition is a stronger condition than the former and we say it evaluates for oligo collisions in the global sense.

Informally, a valid design is a requirement of any potential oligo design solution. Therefore, given that the constraints must be satisfied, we may choose to define some objective score function,  $g$ , of oligo designs, to optimize one or more design criteria. For example, in the remainder of this study, we have chosen to define  $g(\mathcal{O}) = t_{max} - T_{min}(\mathcal{O})$  in order to minimize the range of melting temperatures of designed overlap regions. We could have similarly defined  $g(\mathcal{O}) = T_{max}(\mathcal{O}) - t_{min}$  to achieve a similar effect. However, the definition of  $g$  is not restricted to design criteria alone. Consider the case where there is a known cost per nucleotide in the gene synthesis process. Then, of all possible valid designs, it may be beneficial to define  $g$  such that it would choose the design which requires the least number of nucleotides to be synthesized. This could be achieved by setting  $g(\mathcal{O}) = \sum_{O \in \mathcal{O}} |O|$ , where  $|O|$  is the length of the oligo  $O$ .

We now formally define the collision oblivious and collision aware oligo design problems.

### Collision Oblivious Oligo Design for Gene Synthesis (CO-ODGS)

*Instance:* DNA duplex  $S$ , oligo length range  $[l_{min}, l_{max}]$ , maximum melting temperature of self-hybridization,  $t_{sh}$ , and minimum and maximum overlap region melting temperatures,  $t_{min}$  and  $t_{max}$ .

*Problem:* Find a valid oligo design  $\mathcal{O}^*$  for  $S$ , such that  $g(\mathcal{O}^*) = \min\{g(\mathcal{O}') | \mathcal{O}' \in \mathcal{D}(S)\}$  where  $g$  is some objective score function of oligo designs.  $\mathcal{O}^*$  is an *optimal design with respect to  $g$* .

**Note 2.0.7.** *Although  $g(\mathcal{O})$  is defined above, and in the remainder of this study, as an objective function to be minimized, it could also be defined to maximize some appropriate measure, if required.*

### Collision Aware Oligo Design for Gene Synthesis (CA-ODGS)

The collision aware oligo design for gene synthesis problem (CA-ODGS) is defined as the CO-ODGS problem with the added constraint that the design must also be collision free.

## Chapter 3

# Collision Oblivious Oligo Design Algorithms

In this chapter, we first describe two dynamic programming algorithms that are guaranteed to find an optimal oligo design, if one exists, with respect to some objective evaluation function  $g$  for any instance of the CO-ODGS problem detailed in Chapter 2. Note that the algorithms proposed make no attempt to ensure collision-free oligo designs, a necessary condition of the CA-ODGS problem. We start by describing an algorithm for ungapped oligo designs, report on its space and time complexity, and show how the algorithm can be generalized to handle the case of gapped designs. We conclude the chapter with an extensive experimental analysis where we evaluate the efficacy and efficiency of both algorithms for various design parameters and investigate to what degree oligo collisions arise in collision oblivious designs.

### 3.1 Ungapped Oligo Design Algorithm

In Eqn. (3.1),  $D_{i',j'}^\sigma$  determines the score of the optimal design with respect to function  $g'$ , having an oligo ending at position  $j'$ , on strand  $\sigma$ , and another ending at position  $i'$  on the opposite strand (see Fig. 3.1). Intuitively, the recurrence evaluates all possible values of  $j$ , denoting the oligo  $O_{j,j'}^\sigma$ , for 1) constraint satisfaction, and 2) optimization with respect to the scoring function  $g'$ . Oligo  $O_{j,j'}^\sigma$  is first checked to ensure it is structure free with respect to the self hybridization melting temperature threshold  $t_{sh}$  (line 1). The base case is reached

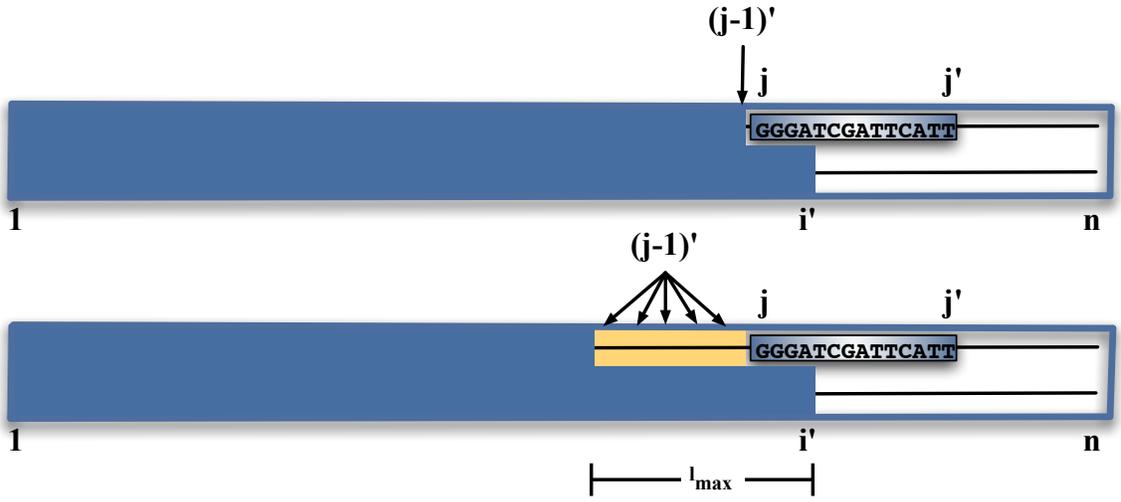


Figure 3.1: The ungapped recurrence (top) attempts to find the best starting position  $j$  for an oligo ending at  $j'$ , given that there is an oligo ending at  $i'$  on the opposite strand. Position  $(j - 1)'$  immediately precedes  $j$  in an ungapped design. The gapped recurrence (bottom) also considers the best position for the oligo ending at  $(j - 1)'$ , given that it overlaps the oligo ending at position  $i'$ .

when  $j = 1$  and the oligo is structure free (line 2). In the recursive case,  $j \neq 1$ , the new complementary overlap region introduced ( $s_j \dots s_{i'}$ ) is evaluated to ensure it is designed Tm satisfied (lines 3,4). If all constraints have been satisfied, then the score is evaluated and defined to be the larger of  $D_{j-1,i'}^{1-\sigma}$ , the score of the optimal design with an oligo ending at position  $j - 1$  on strand  $\sigma$  and one ending at position  $i'$  on strand  $1 - \sigma$ , and of the new score associated with oligo  $O_{j,j'}^\sigma$ , evaluated by  $g'$ .

$$D_{i',j'}^\sigma = \min_{\max(j'-l_{max}+1,1) \leq j \leq \min(j'-l_{min}+1,i')}$$

$$\left. \begin{array}{l} \infty, \text{ if } \text{Tm}(O_{j,j'}^\sigma) > t_{sh} \\ 0, \text{ if } j = 1 \wedge \text{Tm}(O_{j,j'}^\sigma) \leq t_{sh} \\ \infty, \text{ if } j \neq 1 \wedge \text{Tm}(O_{j,i'}^\sigma \cap O_{j,i'}^{1-\sigma}) > t_{max} \\ \infty, \text{ if } j \neq 1 \wedge \text{Tm}(O_{j,i'}^\sigma \cap O_{j,i'}^{1-\sigma}) < t_{min} \\ \max(D_{j-1,i'}^{1-\sigma}, g'(j, i')), \text{ otherwise} \end{array} \right\} \quad (3.1)$$

$$g'(j, i') = t_{max} - \text{Tm}(O_{j,i'}^0 \cap O_{j,i'}^1) \quad (3.2)$$

$$D_{j'}^* = \min_{j'-l_{max} < i' < j'} \left\{ \min_{0 \leq \sigma \leq 1} \{D_{i',j'}^\sigma\} \right\} \quad (3.3)$$

In this particular formulation, we chose to optimize the range of designed overlap melting temperatures by defining our objective optimization function  $g(\mathcal{O}) = t_{max} - T_{min}(\mathcal{O})$ . In order to determine the effect to the score when choosing a particular oligo,  $O_{j,j'}^\sigma$ , we have defined a function  $g'$  with respect  $g$  (see Eqn. (3.2)). We reiterate here that  $g$  could be defined to optimize for another design criteria if desired. Likewise, additional constraints could be imposed and existing ones removed, dependent on the application. Furthermore, while we have defined each hybridization related constraint in terms of melting temperature these could easily be expressed in terms of Gibbs free energy change.

In Eqn. (3.3),  $D_{j'}^*$  defines the optimal score of a design of the prefix of  $S$  of length  $j'$ . With respect to  $j'$ , it evaluates all possible placements of an oligo ending at some position  $i'$  on the opposite strand. Therefore, for some sequence  $S$ , having length  $|S|$ , the optimal design score is  $D_{|S|}^*$ .

### 3.1.1 Time and Space Complexity

We assume that satisfaction of all design constraints can be calculated in constant time (which depends on  $l_{min}$  and  $l_{max}$ ). Let  $S$  be the DNA duplex of the problem instance and  $n = |S|$ . In the case of the ungapped algorithm, for each possible pair  $(i', j') \in \{(i', j') \mid 1 < i', j' \leq |S| \wedge 1 \leq |j' - i'| < l_{max}\}$ , every possible  $j$  must be evaluated to determine the score contributed by oligo  $O_{j,j'}^\sigma$ . There are at most  $l_{max} - l_{min} + 1$  possible placements of  $j$ , given any  $(i', j')$ . Therefore, the ungapped algorithm runs in time  $O((l_{max} - l_{min}) \cdot l_{max} \cdot n) = O(n)$ , as  $l_{min}$  and  $l_{max}$  are design constants. An entry must be stored in the dynamic programming table for every  $(i', j')$ ,  $1 \leq |i' - j'| < l_{max}$ , for both strands, therefore  $O(l_{max} \cdot n) = O(n)$  space is needed. If only a score is required, the space can be reduced to  $O(1)$ .

## 3.2 Gapped Oligo Design Algorithm

Unlike the special case of ungapped designs where  $(j - 1)' = j - 1$ , in a gapped design the previous oligo on strand  $\sigma$  could end at a number of valid positions, denoted as position  $(j - 1)'$ , given that an oligo covers position  $(j - 1)'$  on strand  $1 - \sigma$ . This difference between gapped and ungapped designs is illustrated in Fig. 3.1. Therefore, to generalize Eqn. (3.1)

for gapped designs, replace  $D_{j-1,i'}^{1-\sigma}$  in the recursive case, with:

$$\min_{i'-l_{max} < (j-1)' < j} \left\{ D_{(j-1)',i'}^{1-\sigma} \right\} \quad (3.4)$$

For completeness, we list the full recurrence for finding an optimal gapped oligo design with respect to an optimization function  $g'$ .

$$\begin{aligned} \overline{D}_{i',j'}^\sigma = \min_{\max(j'-l_{max}+1,1) \leq j \leq \min(j'-l_{min}+1,i')} \\ \left( \begin{array}{ll} \infty & , \text{ if } \text{Tm}(O_{j,j'}^\sigma) > t_{sh} \\ 0 & , \text{ if } j = 1 \wedge \text{Tm}(O_{j,j'}^\sigma) \leq t_{sh} \\ \infty & , \text{ if } j \neq 1 \wedge \text{Tm}(O_{j,i'}^\sigma \cap O_{j,i'}^{1-\sigma}) > t_{max} \\ \infty & , \text{ if } j \neq 1 \wedge \text{Tm}(O_{j,i'}^\sigma \cap O_{j,i'}^{1-\sigma}) < t_{min} \\ \max(\min_{i'-l_{max} < (j-1)' < j} \left\{ \overline{D}_{(j-1)',i'}^{1-\sigma} \right\}, g'(j, i')) & , \text{ otherwise} \end{array} \right) \end{aligned} \quad (3.5)$$

### 3.2.1 Time and Space Complexity

Again we assume that satisfaction of all design constraints can be calculated in constant time (which depends on  $l_{min}$  and  $l_{max}$ ) and let  $S$  be the DNA duplex of the problem instance and  $n = |S|$ . For the gapped design algorithm the space remains the same as the ungapped algorithm,  $O(n)$ . As there are at most  $l_{max}$  possible placements of  $(j-1)'$ , in contrast to one valid position in the ungapped case, the time complexity is  $O(l_{max} \cdot (l_{max} - l_{min}) \cdot l_{max} \cdot n) = O(n)$ .

## 3.3 Experimental Analysis

In Sect. 3.3.2 we ask the following questions. First, for various values of  $t_{max}$ , how effective is the gapped and ungapped collision oblivious algorithm in finding valid designs, not necessarily collision free, on real data? Second, how efficient is the runtime performance of the algorithms? Finally, of the valid designs found by the gapped algorithm, to what degree do collisions arise? Following a short description of our data set and implementation details, we report the results of our analyses in the rest of this section.

```

Input: A DNA duplex  $S$ , a design goal  $g$ , and design parameters  $gapped, l_{min}, l_{max}, t_{sh},$ 
 $t_{min}$  and  $t_{max}$ 
Output:  $designScore$  – the score of an optimal design, or  $\infty$  if one does not exist

// first, fill in the  $D$  matrix
for  $j' \leftarrow 1$  to  $|S|$  do
  for  $i' \leftarrow \text{Max}(j' - l_{max} + 1, 1)$  to  $j' - 1$  do
    for  $\sigma \leftarrow 0$  to  $1$  do
       $D_{\sigma,j',i'} \leftarrow \infty;$ 
       $score \leftarrow \infty;$ 
      for  $j \leftarrow \text{Max}(j' - l_{max} + 1, 1)$  to  $\text{Min}(j' - l_{min} + 1, i')$  do
        if  $\text{TmSelfHybridization}(S, \sigma, j, j') > t_{sh}$  then continue;
        if  $j = 1$  then /* base case */
          |  $score \leftarrow 0;$ 
        else /* recursive case */
           $tm \leftarrow \text{TmOverlapRegion}(S, j, i');$ 
          if  $tm > t_{max}$  or  $tm < t_{min}$  then continue;
           $prevScore \leftarrow \infty;$ 
           $jPrevStart \leftarrow j - 1;$ 
          if  $gapped = true$  then  $jPrevStart \leftarrow i - l_{max} + 1;$ 
          for  $j'_{prev} \leftarrow jPrevStart$  to  $j - 1$  do
            |  $prevScore \leftarrow \text{Min}(prevScore, D_{1-\sigma,i',j'_{prev}});$ 
          end
          if  $g = \text{Minimize Tm Range}$  then
            |  $score \leftarrow \text{Max}(t_{max} - tm, prevScore);$ 
          else if  $g = \text{Minimize Base Count}$  then
            |  $score \leftarrow prevScore + j' - j + 1;$ 
          end
        end
      end
      if  $score < D_{\sigma,j',i'}$  then  $D_{\sigma,j',i'} \leftarrow score;$ 
    end
  end
end
end

// next, find the best score for designs having an oligo end at position  $|S|$ 
 $designScore \leftarrow \infty;$ 
 $j' \leftarrow |S|;$ 
for  $i' \leftarrow j' - l_{max} + 1$  to  $j' - 1$  do
  for  $\sigma \leftarrow 0$  to  $1$  do
    | if  $D_{\sigma,j',i'} < designScore$  then  $designScore \leftarrow D_{\sigma,j',i'};$ 
  end
end
end
return  $designScore;$ 

```

**Algorithm 3.1:** Collision oblivious dynamic programming algorithm. Pseudo-code is given for determining the optimal score of both gapped and ungapped oligo designs, for two different design goals.

### 3.3.1 Experimental Environment

#### Data Set

Throughout this thesis, we use a filtered set of the 3,891 CDS (coding DNA sequence) regions of the GENECODE subset of the ENCODE dataset [38] (version hg17 NCBI build 35). This curated dataset comprises approximately 1% of the human genome and is representative of several its characteristics such as distribution of gene lengths and GC composition (54.31%). After filtering any sequences less than 75 bases in length, the remaining 3,157 CDS regions range in length from 75 to 8186 bases, averaging 173 bases with 267 bases standard deviation.

#### Implementation and Hardware

In all experiments, unless otherwise noted, we fix the oligo length range  $[l_{min}, l_{max}] = [37, 52]$  and set  $t_{sh} = 37^{\circ}\text{C}$  and  $t_{min} = 37^{\circ}\text{C}$ . The value of  $t_{max}$  varies and details are given for each experiment set. We set the threshold for oligo collisions to be  $t_{col} = 10^{\circ}\text{C}$ . Calculation of melting temperature values, denoted by the function  $T_m$  in Sect. 2, were performed by the PairFold (for duplexes) and SimFold (for single strands) structure prediction software of Andronescu *et. al* [4]. All algorithms were implemented in C++ and compiled with g++ (GCC 4.1.0). Experiments were run on our reference Pentium IV 2.4 GHz processor machines, with 1GB main memory and 256 Kb of CPU cache, running SUSE Linux version 10.1.

### 3.3.2 Performance of the Collision Oblivious Algorithm

To better understand the practical effectiveness of the collision oblivious algorithm in finding valid solutions and the number of collisions which occur in these designs, we conducted a set of experiments on a limited dataset of 500 sequences chosen uniformly at random from our reference dataset. For each of the sequences, both the ungapped and gapped collision oblivious algorithms were run for each combination of four different values of  $t_{max} = \{60, 70, 80, 90\}$  and two different values of  $l_{max} = \{42, 52\}$ .

#### Efficacy of the Gapped and Ungapped Versions

The gapped version of the algorithm is much more successful in finding valid designs (see Fig. 3.2). This version finds a valid design for all 500 sequences, regardless of the value of

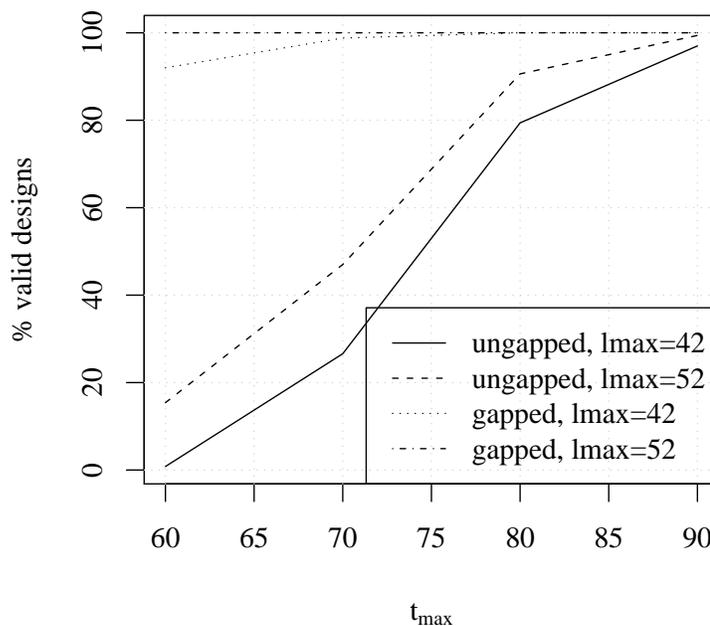


Figure 3.2: The percentage of valid designs (of the 500 sample sequences) at different values of  $t_{max}$  is reported for both the gapped and ungapped algorithm for different values of  $l_{max}$ . The gapped algorithm is much more successful at finding valid designs.

$t_{max}$ , when  $l_{max} = 52$ . When  $l_{max} = 42$ , a valid design is found in nearly every instance, however, when  $t_{max} = 60$  valid designs were not possible for 41 sequences. In contrast, the ungapped version is relatively ineffective in practice, unless  $t_{max} \geq 80$ . Even for the highest value of  $t_{max}$  we tested,  $90^{\circ}\text{C}$ , there was still one sequence where an ungapped design was impossible, given the design constraints. Overall, of the 4000 design attempts of the ungapped algorithm, only 2273 valid designs were found. Clearly, the added design flexibility of the gapped version is crucial for designs requiring a low or moderate value for  $t_{max}$ .

### Runtime Performance

In Fig. 3.3 left side, the runtime distribution of the 500 sampled sequences is shown for both the ungapped and gapped algorithms for both values of  $l_{max}$  when  $t_{max} = 60$ . Runtime performance between the gapped and ungapped algorithm is virtually indistinguishable for the same value of  $l_{max}$ . However, the distributions for different values of  $l_{max}$  are separated by a constant factor. This characteristic behaviour is also observed for the other values

$t_{max}$	$l_{max}$	Gapped	% Valid Designs	CPU runtime (stdev)	Score (stdev)
60.0	42	No	0.6	3.87 (6.84)	1.57 (1.34)
60.0	42	Yes	92.0	3.92 (6.81)	1.59 (1.49)
60.0	52	No	15.2	15.95 (29.36)	1.19 (1.90)
60.0	52	Yes	100	16.75 (29.77)	0.47 (0.39)
70.0	42	No	26.4	3.88 (6.80)	4.70 (3.33)
70.0	42	Yes	98.8	3.99 (6.96)	2.42 (2.49)
70.0	52	No	46.9	16.27 (29.95)	1.06 (1.63)
70.0	52	Yes	100	16.26 (29.35)	0.34 (0.36)
80.0	42	No	79.4	3.85 (6.83)	10.71 (5.55)
80.0	42	Yes	100	4.03 (8.43)	8.77 (5.16)
80.0	52	No	90.6	16.63 (36.59)	3.07 (2.95)
80.0	52	Yes	100	16.42 (30.36)	2.47 (2.98)
90.0	42	No	97.00	3.84 (6.73)	19.84 (6.30)
90.0	42	Yes	100	3.96 (6.92)	18.52 (5.55)
90.0	52	No	99.40	16.09 (29.10)	10.29 (5.25)
90.0	52	Yes	100	16.30 (29.91)	10.12 (5.28)

Table 3.1: Results of the collision oblivious algorithm, using various design parameters, for 500 randomly selected sequences of the filtered GENECODE data set.

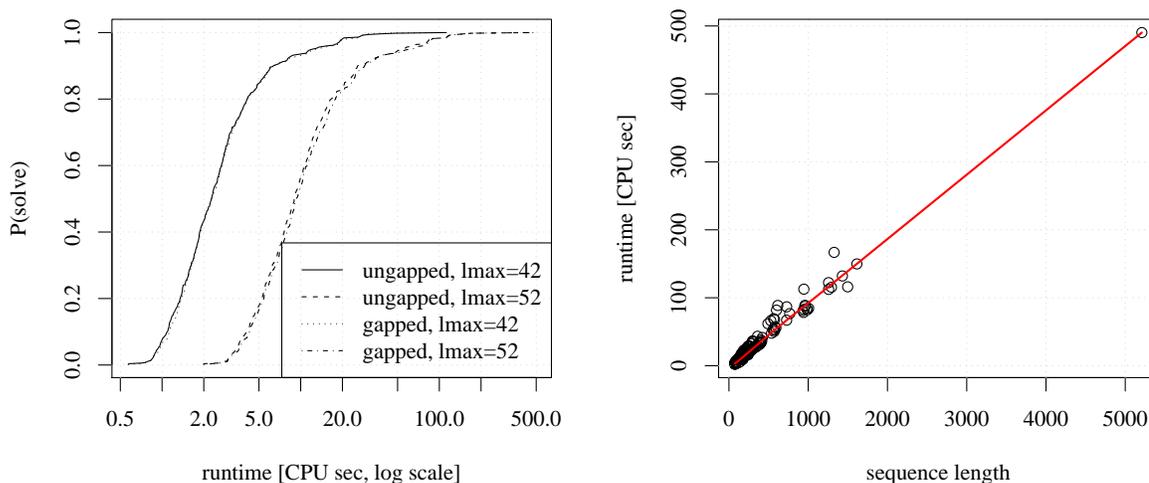


Figure 3.3: On the left, the runtime distribution of the 500 sampled sequences is shown for both the ungapped and gapped algorithm for two different maximum oligo lengths and  $t_{max} = 60$ . Performance between gapped and ungapped variants is virtually indistinguishable for the same value of  $l_{max}$ . A constant difference is observed between different values of  $l_{max}$ . On the right, the CPU runtime for each of the 500 sampled sequences is plotted against sequence length for the gapped algorithm when  $t_{max} = 60$  and  $l_{max} = 52$ . A lowest regression curve of best fit has been added.

of  $t_{max}$  (data not shown). On the right side of Fig. 3.3, the runtime is plotted for all 500 sequences against their sequence length for the gapped algorithm with  $l_{max} = 52$  and  $t_{max} = 60$ . Clearly, the algorithm scales linearly in practice. This characteristic runtime performance is also observed for all other combinations of design parameters (data not shown).

### Frequency of Oligo Collisions

Due to the relatively poor effectiveness of the ungapped design algorithm in finding valid designs compared with the gapped design algorithm, we omit it from further analysis. For each valid design found, PairFold was used to determine the number of oligo collisions based on the protocol previously described. In Fig. 3.4, we report the number of collisions per 100 bases for each valid design found, for various values of  $t_{max}$ , in an attempt to normalize for sequence length. The left side of the figure reports results for designs with  $l_{max} = 42$ , while the right side shows results for  $l_{max} = 52$ . The mean number of collisions per 100 bases, regardless of design parameters, is approximately 0. However, there do exist outliers having

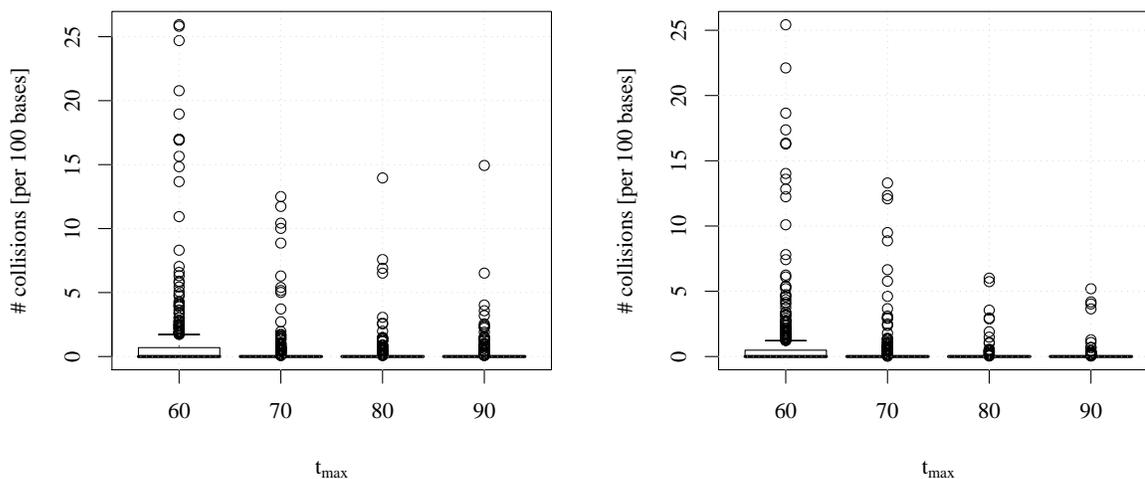


Figure 3.4: The number of collisions per 100 bases is reported for the optimal collision oblivious designs of the sampled 500 sequences, at different values of  $t_{max}$ , using the gapped algorithm with  $l_{max} = 42$  on the left, and  $l_{max} = 52$  on the right.

a significant number of collisions. When  $l_{max} = 52$ , the worst observed results generally decrease as  $t_{max}$  is increased. Overall, compared with the number of collisions possible, very few are observed in practice. This suggests it might be possible to adapt the collision oblivious algorithm to eliminate collisions.

## Chapter 4

# Complexity of Collision-Aware Oligo Design

We conjecture that the collision-aware oligo design for gene synthesis (CA-ODGS) problem is  $\mathcal{NP}$ -complete. To provide some evidence for this, we show  $\mathcal{NP}$ -completeness of a variation of the problem, which abstracts away thermodynamic details, while retaining the key challenge of collision-aware partitioning. Informally, the variation asks whether a single string (as opposed to a duplex) can be partitioned into short substrings of bounded length, no two of which are identical. For example, consider partitioning the string *theimportantthingisnevertostopquestioning* into substrings having a maximum length of 3. One possible solution is shown in Fig. 4.1 (top). Notice, however, that some partitions may produce substrings which are identical as is the case for the other partition shown in Fig. 4.1 (bottom) where both the substrings *th* and *ing* appear twice.



Figure 4.1: Two partitions are shown for the string *theimportantthingisnevertostopquestioning*. The substrings in both partitions have maximum length 3. The partition shown above the string is valid, in that no two substrings are identical; however, the partition shown below the string is invalid as there are two cases of identical substrings indicated with arrows.

In this chapter, we formally introduce the collision-aware string partition (CA-SP) problem. We motivate its similarity to the CA-ODGS problem and point out differences. Next, a known  $\mathcal{NP}$ -complete problem, 3SAT(3), is presented. A polynomial time reduction from 3SAT(3) to CA-SP is given and is followed by a proof of  $\mathcal{NP}$ -completeness for CA-SP. Finally, we conclude with some remarks to highlight the implications of a recent hardness proof of a restricted version of the CA-SP problem [24].

## 4.1 The Collision-Aware String Partition Problem

Let  $\Sigma$  be a finite alphabet. A  $k$ -partition of a string  $A \in \Sigma^*$  is sequence  $P = p_1, p_2, \dots, p_l$ , for some  $l$ , where each  $p_i$  is a string over  $\Sigma$  of length at most  $k$  and  $A = p_1 p_2 \dots p_l$ . A sequence of strings  $p_1, p_2, \dots, p_l$  is *collision-free* if for all  $i, j$ ,  $1 \leq i \neq j \leq l$ ,  $p_i \neq p_j$ . Given a string  $A$  and a partition  $P$  of  $A$ , we say that a substring  $a$  of  $A$  is *selected* if and only if  $a$  is an element of the partition  $P$ .

### Collision-Aware String Partition (CA-SP)

*Instance:* Finite alphabet  $\Sigma$ , a positive integer  $k$ , and a string  $A$  from  $\Sigma^*$ .

*Question:* Is there a collision-free,  $k$ -partition  $P$  of  $A$ ?

CA-SP differs from CA-ODGS in several ways: in CA-SP, the goal is to partition a string, rather than a duplex; pairs of substrings that form a stable duplex are not considered; and constraints on individual substrings are not modeled at all. Also, CA-SP pertains to an arbitrary alphabet, whereas CA-ODGS pertains to an alphabet of size 4. However, the task of designing a collision-free  $k$ -partition is quite similar to that of developing a collision-free oligo design. The design goal of CA-SP, that of avoiding identical substrings, is one of the design goals of CA-ODGS. To illustrate the importance of avoiding identical substrings in the CA-ODGS problem, consider the oligo design in Fig. 4.2. The oligos labeled  $c$  and  $g$  are identical and on the same strand. If oligo  $g$  hybridized to oligo  $b$ , the gene construction could result in two distinct fragments (Fig. 4.2 top). Those labeled  $e$  and  $h$  are also identical (read 5' to 3'), but on opposite strands. If  $h$  hybridized to  $d$ , one full fragment could result, however, it would contain a complementary inversion error (Fig. 4.2 bottom). This type of error can be particularly troublesome, as the construct may need to be fully sequenced before the error is discovered.

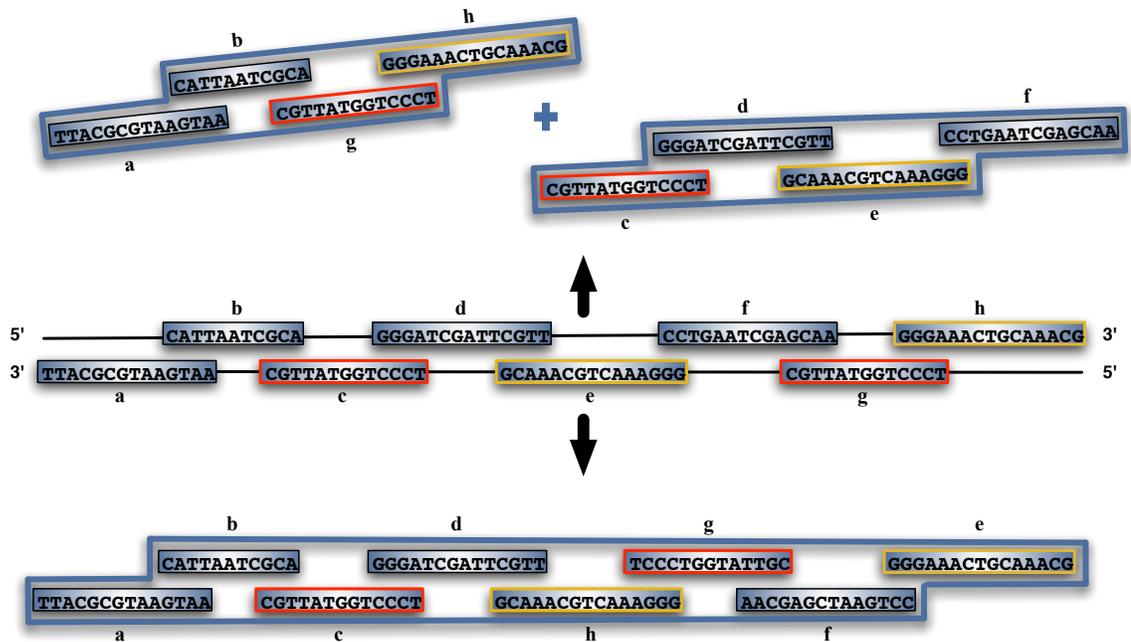


Figure 4.2: An invalid design (middle) containing two pairs of identical oligos. The identical oligos *c* and *g* could hybridize out of order resulting in two distinct fragments (top). Identical oligos from opposite strands must also be avoided as entire regions could be assembled out of order (bottom).

Next we introduce 3SAT(3) as presented and shown to be  $\mathcal{NP}$ -complete by Papadimitriou [26].

### 3SAT(3)

*Instance:* A formula  $\phi$  with a set  $C$  of clauses over a set  $X$  of variables in conjunctive normal form such that:

1. every clause contains at most three literals,
2. each variable is restricted to occur at most three times, and
3. each literal is restricted to occur at most twice.

*Question:* Is  $\phi$  satisfiable?

## 4.2 Reducing 3SAT(3) to CA-SP

We now describe a reduction from 3SAT(3) to CA-SP. Let  $\phi$  be an instance of 3SAT(3), with set  $C$  of clauses, and set  $X$  of variables. We shall define an alphabet  $\Sigma$  and construct a string  $A \in \Sigma^*$ , such that  $A$  has a collision-free 2-partition if and only if  $\phi$  is satisfiable.

We construct  $A$  to be the concatenation of three strings,  $A = A'A''A'''$ , with the following properties. First, the string  $A'$  encodes the clauses of  $\phi$ , so that a collision-free 2-partition of  $A$  unambiguously selects a literal from each clause. Intuitively, the selected literals are intended to be a satisfying truth assignment for the variables of  $\phi$ . Second, if  $A$  has a collision-free 2-partition, string  $A''$  ensures that the selected literals are *consistent*, that is, no selected literal can be the negation of another. The constructions of  $A'$  and  $A''$  rely on the fact that special short delimiting strings are *forbidden*—that is, cannot be selected in the 2-partition of  $A'A''$ . The string  $A'''$  is constructed to ensure that, in a collision-free 2-partition of  $A$ , the forbidden strings must be selected from  $A'''$ , and thus cannot be selected from  $A'A''$ .

We use alphabet  $\Sigma$  and set of forbidden strings  $\mathcal{F}$ , defined as follows:

$$\Sigma = \{x_i : x_i \in X\} \cup \{\alpha_i^j : c_i \in C \wedge j \leq |c_i|\} \cup \{\boxplus, \boxminus, \boxtimes, \boxtimes\} \quad (4.1)$$

$$\mathcal{F} = \{\boxplus, \boxplus\boxplus, \boxminus, \boxminus\boxminus, \boxtimes, \boxtimes\boxtimes, \boxtimes\boxtimes, \boxtimes\boxtimes\} \quad (4.2)$$

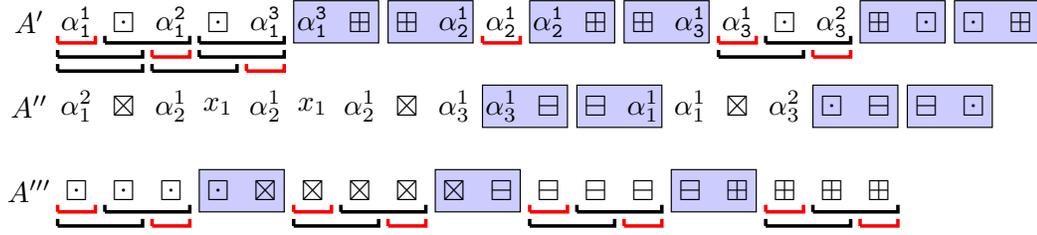


Figure 4.3: The construction of string  $A = A'A''A'''$  for an instance  $\phi$  of 3SAT(3), with  $X = \{x_1, x_2, x_3\}$  and  $C = \{(\neg x_2, x_1, \neg x_3), (\neg x_1), (x_1, x_2)\}$ . Selections outlined with a box are required to avoid a collision. All potential partitions of clause strings, in  $A'$ , and forbidden strings, in  $A''$ , are shown.

**Construction of  $A'$ :** For each clause  $c_i \in C$ , construct the  $i$ th clause string to be  $\alpha_i^1$  if  $|c_i| = 1$ ,  $\alpha_i^1 \boxtimes \alpha_i^2$  if  $|c_i| = 2$ , and  $\alpha_i^1 \boxtimes \alpha_i^2 \boxtimes \alpha_i^3$  if  $|c_i| = 3$ . Next, construct the  $i$ th clause connector string as  $\alpha_i^{|c_i|} \boxtimes \alpha_{i+1}^1$ .  $A'$  is the concatenation of the 1st clause string, the 1st clause connector, the 2nd clause string, the 2nd clause connector, and so on, up to the  $|C|$ th clause string, and then terminated by  $\boxtimes \boxtimes \boxtimes$ . See Fig. 4.3 for an example.

**Lemma 4.2.1.** *Given that no string from the forbidden set  $\mathcal{F}$  is selected, exactly one literal symbol can be selected for each clause string in any collision-free 2-partition  $P'$  of  $A'$ . Additionally, the termination string  $\boxtimes \boxtimes \boxtimes$  must be partitioned as  $\boxtimes \boxtimes$ ,  $\boxtimes \boxtimes$ .*

*Proof.* Consider a collision-free 2-partition of  $A'$ . Since the strings  $\boxtimes$  and  $\boxtimes \boxtimes$  are forbidden, it must be that the partition includes the substrings  $\alpha_i^{|c_i|} \boxtimes$  and  $\boxtimes \alpha_{i+1}^1$  of the  $i$ th clause connection string. Therefore, each clause string must be partitioned independently of the symbols in the adjoining clause connector strings.

Consider the clause string for clause  $c_i$ . If  $c_i$  has one literal, it must be selected. If  $c_i$  has two or three literals, the forbidden substring  $\boxtimes$  cannot be selected alone. Therefore, each  $\boxtimes$  must be selected with an adjacent literal symbol. This leaves exactly one other literal symbol which must be selected.

Finally, neither of the termination symbols  $\boxtimes$  or  $\boxtimes$  can be selected alone in the string, since they are in the forbidden set, and similarly the string  $\boxtimes \boxtimes$  cannot be selected since it too is in the forbidden set. Therefore, the termination string  $\boxtimes \boxtimes \boxtimes$  must be partitioned as  $\boxtimes \boxtimes$  and  $\boxtimes \boxtimes$ .  $\square$

**Construction of  $A''$ :** We must now ensure that no literal of  $\phi$  that is selected in  $A'$  is the negation of another selected literal. By definition of 3SAT(3), each variable is restricted

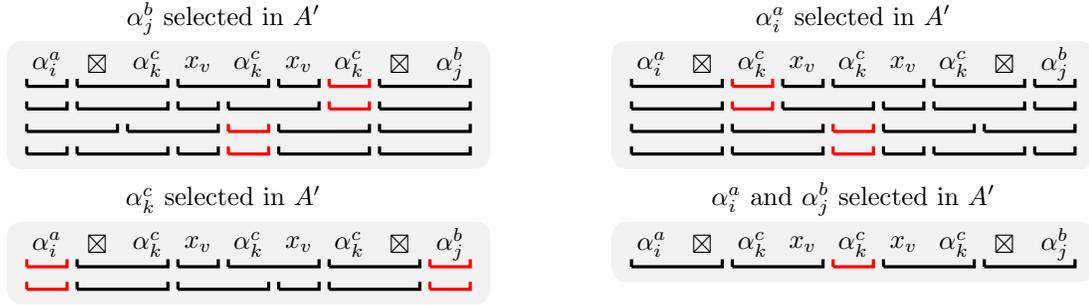


Figure 4.4: For a variable  $x_v$  having three literals  $\alpha_i^a, \alpha_j^b, \alpha_k^c$ , with  $(\alpha_k^c = \neg \alpha_i^a) \wedge (\alpha_k^c = \neg \alpha_j^b)$ , four subsets of the literals can possibly be selected in  $A'$ . The enforcer string guarantees two opposite literals cannot simultaneously be selected in  $A'$ . For instance, their are four valid partitions of the enforcer string when  $\alpha_j^b$  is selected in  $A'$ . In all four of these partitions, the opposite literal,  $\alpha_k^c$  is selected and therefore cannot be selected in  $A'$  without introducing a collision. Similar arguments follow for the other three combinations of selecting literals of a variable in  $A'$ .

to appear at most three times, and each literal is restricted to appear at most twice. It is sufficient to consider the following two cases for each variable:

1. *The variable occurs twice, with one occurrence being the negation of the other.* Without loss of generality, assume the two occurrences are  $\alpha_i^a$  and  $\alpha_j^b$ . Then construct the *enforcer string* for this variable to be  $\alpha_i^a \boxtimes \alpha_j^b$ .
2. *The variable occurs three times, with one literal being a negation of the other two.* Without loss of generality, let the three occurrences be  $\alpha_i^a, \alpha_j^b$  and  $\alpha_k^c$ , with  $\alpha_k^c$  being a negation of the other two. Then construct the *enforcer string* for this variable to be  $\alpha_i^a \boxtimes \alpha_k^c x_v \alpha_k^c x_v \alpha_k^c \boxtimes \alpha_j^b$ .

Now, construct  $A''$  by concatenating all enforcer strings (in any order), separated by *enforcer connector strings*. The  $i$ th enforcer connector string is  $\sigma \boxtimes \boxtimes \sigma'$ , where  $\sigma$  and  $\sigma'$  are the last symbol of the  $i$ th enforcer string in  $A''$  and the first symbol of the  $i + 1$ st enforcer string in  $A''$ , respectively. Terminate the resulting string with  $\boxtimes \boxtimes \boxtimes \boxtimes$ . See Fig. 4.3.

**Lemma 4.2.2.** *Given that no string from the forbidden set  $\mathcal{F}$  is selected, any collision-free 2-partition of  $A'A''$  must be consistent, and must partition the termination string of  $A''$  into  $\boxtimes \boxtimes$  and  $\boxtimes \boxtimes$ .*

*Proof.* Consider a collision-free 2-partition of  $A'A''$ . As was shown for clause strings in Lem. 4.2.1, the forbidden set forces that each enforcer connector string must be partitioned into two parts, each of size 2. Thus, each enforcer string must be partitioned independently.

Consider the enforcer string,  $\alpha_i^a \boxtimes \alpha_j^b$ , for two inconsistent literals  $\alpha_i^a$  and  $\alpha_j^b$ . Suppose that only one literal is selected in  $A'$ . Without loss of generality assume this to be  $\alpha_i^a$ . Then the enforcer string can be partitioned into  $\alpha_i^a \boxtimes$  and  $\alpha_j^b$ , without forcing a collision. However, if both  $\alpha_i^a$  and  $\alpha_j^b$  are selected in  $A'$ , there is no possible partition of the enforcer string to avoid a collision.

Next, consider the the enforcer string for three literals, with  $\alpha_k^c$  being the negation of  $\alpha_i^a$  and  $\alpha_j^b$ . We show the necessary and sufficient conditions by case analysis as presented in Fig. 4.4. Each of the four valid combinations of selecting the variables in  $A'$  is presented in the figure. For each, there exists at least one valid partitioning of the enforcer string. Furthermore, each of the possible valid partitions of the enforcer string prohibits an inconsistent selection. Specifically, suppose that  $\alpha_k^c$  is selected in  $A'$ . Then there are two possible partitions of the enforcer strings (see Fig. 4.4 bottom left). In both partitions,  $\alpha_i^a$  and  $\alpha_j^b$  are selected and therefore cannot be selected in  $A'$  without introducing a collision. In the other three cases where one or both of  $\alpha_i^a$  and  $\alpha_j^b$  are selected, every partition selects  $\alpha_k^c$  and therefore it cannot be selected in  $A'$  without a collision.

Finally, neither of the termination symbols  $\square$  or  $\boxminus$  can be selected alone in the string, since they are in the forbidden set, and similarly the string  $\boxminus\boxminus$  cannot be selected since it too is in the forbidden set. Therefore, the termination string  $\square\boxminus\boxminus\square$  must be partitioned as  $\square\boxminus$  and  $\boxminus\square$ .  $\square$

**Construction of  $A'''$ :** To ensure that the forbidden substrings cannot be selected in  $A'$  or  $A''$ , we construct  $A'''$  as shown in Fig. 4.3. A collision-free 2-partition of  $A'''$  selects every forbidden substring. Note that the connectors with a boxed outline in the figure must be selected, or a collision occurs. Then, for each sequence of forbidden symbols, there exists two possible partitions, both of which force selection of two forbidden substrings. Also, neither of the terminator strings at the end of  $A'$  or  $A''$  appear in  $A'''$ . This completes the reduction.

**Theorem 4.2.3.** *Collision-Aware String Partition (CA-SP) is  $\mathcal{NP}$ -complete.*

*Proof.* It is easy to see that  $\text{CA-SP} \in \mathcal{NP}$ : a nondeterministic algorithm need only guess a partition  $P$  where  $|p_i| \leq k$  for all  $p_i$  in  $P$  and check in polynomial time that no two substrings

in  $P$  are identical. Furthermore, it is clear that an arbitrary instance  $\phi$  of 3SAT(3) can be reduced to an instance CA-SP, specified by a string  $A$ , in polynomial time and space by the reduction detailed above.

Now suppose there is a satisfying truth assignment for  $\phi$ . Simply select one corresponding true literal per clause in  $A'$ . The construction of clause strings guarantees that a 2-partition of the rest of each clause string is possible, and this 2-partition can be extended to yield a 2-partition of  $A$ . Also, since a satisfying truth assignment for  $\phi$  cannot assign truth values to opposite literals, then Lem. 4.2.2 guarantees that a valid partition of the enforcer strings are possible. Therefore, there exists a collision-free string partition of  $A$ .

Likewise, consider a collision-free string partition of  $A$ . Lem. 4.2.1 ensures that exactly one literal per clause is selected. Furthermore, Lem. 4.2.2 guarantees that if there is no collision, then no two selected variables in the clauses are negations of each other. Therefore, this must correspond to a satisfying truth assignment for  $\phi$ .  $\square$

### 4.3 A Restricted Version of CA-SP

One of the major differences between the CA-SP and CA-ODGS problems is that CA-ODGS uses a restricted alphabet of size 4 whereas CA-SP uses an unbounded, finite alphabet. One could argue that CA-SP may be polynomial time solvable when  $|\Sigma| = 4$ , and therefore, there exists little evidence that CA-ODGS is  $\mathcal{NP}$ -hard. However, Mañuch *et al.* have recently shown CA-SP is  $\mathcal{NP}$ -complete even under this alphabet restriction [24]. Furthermore, the authors have also been able to model another design goal shared by CA-ODGS. The new version of CA-SP forbids any substring in a valid partition to be a complement of another, where a complementary pair of substrings is defined analogously as a complementary pair of oligos is in the CA-ODGS problem. This new result provides stronger evidence that CA-ODGS is hard. What remains is to extend the proof to consider a duplex of strings, rather than a single strand.

## Chapter 5

# Collision Aware Oligo Design Algorithms

In the previous chapter, we provided evidence that CA-ODGS is  $\mathcal{NP}$ -hard. Given that a polynomial time algorithm for this problem is unlikely, we now propose heuristic methods to either prevent oligo collisions from occurring in a design or to partition a design into collision free regions. An experimental analysis is conducted to determine the effectiveness of the algorithms in producing designs considered to be collision free.

### 5.1 Greedy Collision Aware Algorithm

We now briefly outline a simple extension to the collision oblivious dynamic programming oligo design algorithm detailed in Chapter 3, which uses a greedy approach. Recall that the oligo design recurrences of Eqn. (3.1) (ungapped) and Eqn. (3.5) (gapped) evaluate each prospective oligo and determine a score for adding it to an optimal sub-solution which precedes it (see Fig. 3.1). The greedy extension is applied as follows. For each new oligo being considered, it is first compared with all oligos in the sub-solution to determine if a collision is being introduced. If one is, the new oligo is considered invalid. In this greedy fashion, once an oligo is selected in a particular sub-solution, it always remains in that sub-solution. Note that the space complexity of the algorithm remains linear, however, the time complexity increases to  $O(n^2)$ .

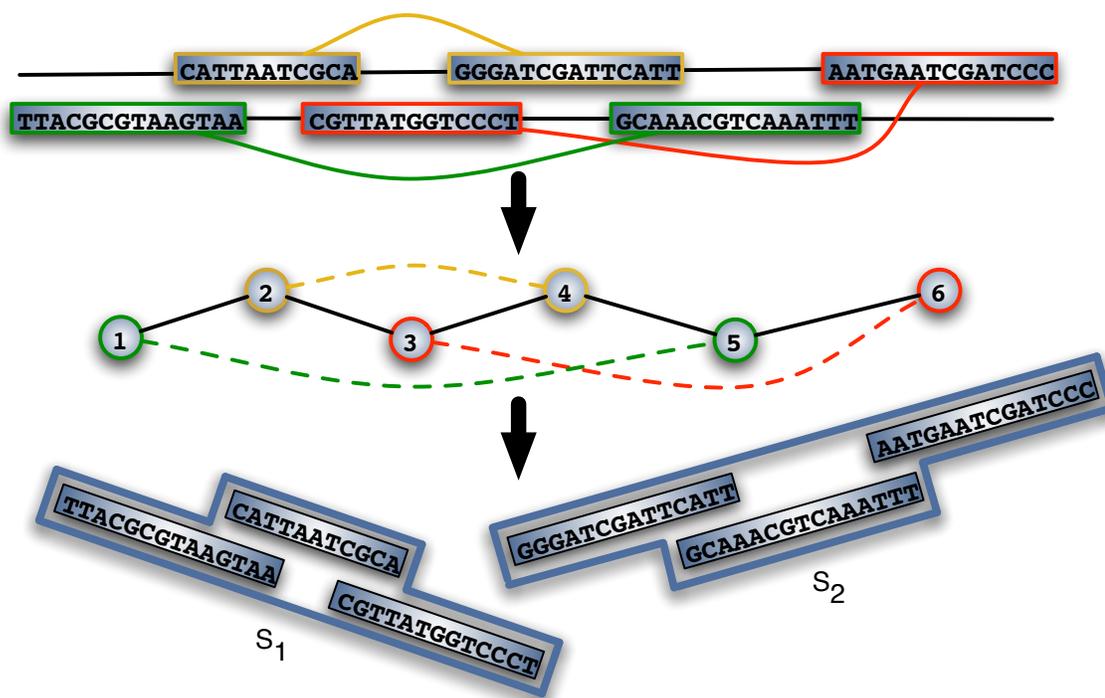


Figure 5.1: An oligo design with collisions denoted by edges (top) is transformed into an oligo collision graph (middle) with designed hybridizations shown with solid edges and collisions shown with dashed edges. A minimum-size partition of the graph, representing synthons, is shown (bottom).

## 5.2 Collision Aware Synthon Design

Informally speaking, the heuristic we now propose does not attempt to minimize the number of oligo collisions. Rather, it partitions an optimal collision oblivious design  $\mathcal{O}$  into a minimum number of synthon regions which are collision free. We note that there may exist another optimal collision oblivious design which produces fewer synthons, however, our algorithm makes no attempt to find such a design. As we demonstrate in Sect. 5.3.2 this simple approach is very effective in practice.

First, given the optimal valid design  $\mathcal{O}$  for an input duplex  $S$ , construct the *oligo collision graph*  $G = (V, E = E' \cap E'')$ , in which  $V = \text{set}(\mathcal{O})$  and each node is labeled according to the oligo order. That is, the oligo which covers the first position of the duplex is labeled ‘1’, then the first oligo on the opposite strand is labeled ‘2’, and the remaining labels are assigned by alternating between strands from left to right relative to the 5’ end of the sense strand.  $E'$

is the set of edges representing designed hybridizations, and  $E''$  the edges representing oligo collisions, based on our previous definitions. Refer to Fig. 5.1 for an example with designed hybridizations shown with solid edges and collisions shown with dashed edges.

Second, given  $G$ , determine the minimum-size partition  $P$  of  $V$  such that  $G[p]$  is connected and does not contain an edge from  $E''$ ,  $\forall p \in P$ . A minimum-size partition of the collision graph corresponds to the minimum number of collision free synthons required to cover  $\mathcal{O}$ . In Sect. 5.2.1, we present a dynamic programming algorithm for this task.

### 5.2.1 Synthon Partition Algorithm

Intuitively,  $D'_k$  from Eqn. (5.1) is the minimum number of collision free partitions (synthons) of an oligo collision graph  $G$  required to cover the collision oblivious design  $\mathcal{O}$ , up to oligo  $k$ . The recurrence determines the best start position  $i$  of a synthon ending at position  $k$ . For a potential synthon consisting of oligos  $i, \dots, k$ , if there is a collision between any pair of these oligos, then the synthon is considered invalid (line 1). The base case occurs when a synthon begins at the first oligo and no pair of oligos in the proposed synthon are in conflict (line 2). Otherwise, the recursive case adds an additional synthon to the score of the previous best solution at  $D'_{i-1}$  (line 3). Therefore, the minimum number of collision free synthons required to cover  $\text{set}(\mathcal{O})$  having  $x$  oligos is given by  $D'_x$ . We set  $D'_0 = 0$ , and for  $1 \leq k \leq x$  we have

$$D'_k = \min_{1 \leq i \leq k} \left\{ \begin{array}{ll} \infty & , \text{ if } \exists j, i \leq j \leq k, (j, k) \in E'' \\ 1 & , \text{ if } \forall j, i \leq j \leq k, (j, k) \notin E'' \wedge (i = 1) \\ D'_{i-1} + 1 & , \text{ otherwise} \end{array} \right\} \quad (5.1)$$

### 5.2.2 Time and Space Complexity

We assume that an oligo collision graph is given as input to the synthon partition algorithm. However, we note this graph can be constructed naively in  $O(x^2)$  time by comparing every pair of oligos under the assumption that the collision condition can be calculated in some constant time.

The synthon partition algorithm is quadratic in the number of oligos,  $x$ , in  $\mathcal{O}$ . Since for every ending position  $k$ ,  $1 \leq k \leq x$ , all possible starting positions of the synthon must be

evaluated,  $1 \leq i \leq k$ , then in the worst case,  $O(x^2)$  time is required. As we must store an entry in the dynamic programming table for each  $k$ ,  $1 \leq k \leq x$ , then  $O(x)$  space is required.

### 5.2.3 A Speedup for determining the collision graph

Although the collision graph can be naively constructed in  $O(x^2)$  time, the same complexity of the synthon partition algorithm, in practice the runtime performance for this task is much worse than the partition algorithm. The following observation can lead to a significant speedup in runtime performance.

**Observation 5.2.1.** *For any two oligos  $i, j$ ,  $i < j$ , if  $i$  and  $j$  are in conflict, then for any oligo  $k$ ,  $j < k$ ,  $k$  and  $i$  must be contained in different collision free partitions.*

This observation leads to the notation of a *minimal* collision graph. As conflict edges are being determined, it is not necessary to compare two oligos for a conflict if a previous conflict, occurring between the two, is already known. We note that this speedup does not decrease the worst case complexity, as every pair of oligos will still be compared for conflicts if no conflicts in the design exist. However, to quantify the improvement we conducted all experiments of Sect. 5.3.2 with and without the use of this speedup for determining the collision graph. The worst case for determining the collision graph without the speedup was 67 minutes, while only 19 minutes when the speedup was used. The average runtime of 151.1 seconds was reduced to 2.3 seconds when the speedup was applied. All results of experiments reported in Sect. 5.3.2 made use of this speedup.

## 5.3 Experimental Analysis

For all experiments conducted below, we use the same five hundred sequences chosen uniformly at random from our reference dataset as those in Chapter 3. Due to the poor design success rate of the ungapped algorithm, we omit it from further analysis.

### 5.3.1 Performance of the Greedy Algorithm

To evaluate the greedy algorithm, gapped oligo designs were attempted for each combination of the following design parameters. For oligo length,  $l_{min}$  was fixed at 37 bases, while  $l_{max}$  varied in  $\{42, 52\}$ . For oligo melting temperature,  $t_{sh}$  was fixed at 37.0,  $t_{col}$  at 10.0,  $t_{min}$  at 50.0 and  $t_{max}$  varied in  $\{60, 70, 80, 90\}$ . The results for valid designs are shown in Table 5.1.

$l_{max}$	$t_{max}$	valid designs found	longest valid design	average length of valid designs
42	60.0	0	0	0.0
42	70.0	1	90	90.0
42	80.0	9	90	83.0
42	90.0	33	137	91.6
52	60.0	70	93	83.1
52	70.0	70	93	83.1
52	80.0	71	109	83.5
52	90.0	82	145	87.4

Table 5.1: The greedy oligo design algorithm performs poorly in practice, finding few collision free designs and only succeeds for short sequences.

Overall, the greedy algorithm is ineffective in practice. In the best case, only 16% of the 500 sequences could be designed collision free. Furthermore, all valid collision free designs which were found were all for short sequences.

### 5.3.2 Performance of the Synthon Partition algorithm

For each of the five hundred sequences, the gapped collision oblivious design algorithm was run for each combination of  $t_{max} = \{60, 70, 80, 90\}$  and  $l_{max} = \{42, 52\}$ . All other design parameters remained fixed as described above. For each valid design which resulted, the collision conflict graph was constructed based on the protocol previously described. The synthon design algorithm was then run to determine the minimum number of synthons needed to have a collision free partition of the original valid design. Runtime statistics were tracked throughout and are discussed below.

In Fig. 5.2, the cumulative distributions of required synthons over the five hundred sequences is plotted for each value of  $t_{max}$  with results for  $l_{max} = 42$  shown on the left, and  $l_{max} = 52$  shown on the right. The worst case occurs for  $t_{max} = 60^\circ\text{C}$  and  $l_{max} = 42$  when 16 synthons are required to partition a 1.3kb sequence into collision free regions. However, even at this lowest temperature tested, approximately 85% of all sequences require two synthons or less to become collision free and roughly 70% requiring only one synthon. Each successive value of  $t_{max}$  further improves upon this result with  $t_{max} = 70^\circ\text{C}$  requiring at most two synthons for 95% of sequences.

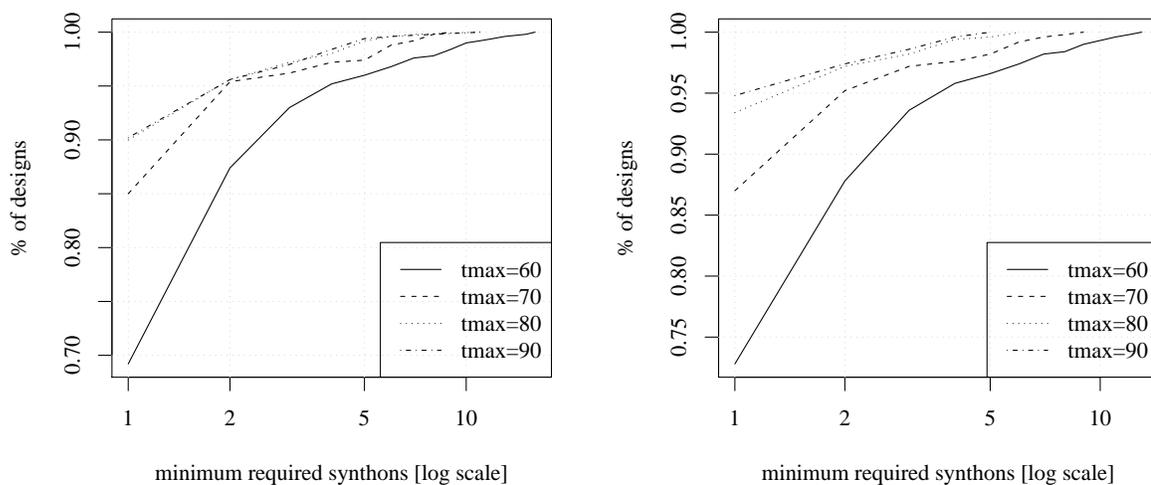


Figure 5.2: For moderate values of  $t_{max}$ , very few synthons are required to make a design collision free. Results for  $l_{max} = 42$  are shown on the left and results for  $l_{max} = 52$  are shown on the right.

## Chapter 6

# Codon Optimization

In this chapter, we begin by formally defining the problem of codon optimization. We detail the general objectives of the problem, some common constraints, and highlight two optimization criteria which are often used in practice. Algorithms are proposed for both, followed by a discussion of their correctness and time and space complexity. We adapt one of the algorithms to find a set of solutions, under the additional constraint that no two DNA sequences in the solution set can have a sequence similarity above a predetermined threshold. We conjecture that this adaptation can be particularly useful for finding collision-free designs using the previously proposed algorithms of Chapters 3 and 5. We conduct an extensive experimental analysis to determine the effectiveness and efficiency of the proposed algorithms on a large biological data set. Finally, we end with an experimental analysis on the utility of codon optimization for collision free oligo design.

### 6.1 Problem Definition and Overview

In general, codon optimization entails the encoding of an amino acid sequence into a corresponding DNA sequence, using the genetic code, such that some constraints are satisfied and some criterion is optimized. In this section we begin by defining some new terms and notation, review common constraints and optimization criteria for the problem and end with a formal problem definition.

### 6.1.1 Notation and Definitions

Recall from Chapter 2 that a DNA strand  $S$  is a string over the alphabet  $\Sigma_{DNA} = \{A, C, G, T\}$ . A *codon* is a triple over  $\Sigma_{DNA}$  and therefore  $4^3 = 64$  distinct codons exist. An *amino acid sequence* is a string over the alphabet  $\Sigma_{AA} = \{Ala, Arg, Asn, Asp, Cys, Glu, Gln, Gly, His, Ile, Leu, Lys, Met, Phe, Pro, Ser, Thr, Trp, Tyr, Val, stop\}$  with each symbol representing an amino acid and the special symbol ‘*stop*’ denoting a string terminal. The *genetic code* is a mapping between amino acids and codons. However, as there are 64 codons and only 20 amino acids (plus one stop symbol), the code is degenerate, resulting in a one-to-many mapping from each amino acid to a set of corresponding codons<sup>1</sup>. Suppose  $\alpha_i$  is an amino acid. We define  $|\lambda(\alpha_i)|$  to be the number of codons corresponding to  $\alpha_i$  and  $\lambda_j(\alpha_i)$  to be the  $j^{th}$  such codon,  $1 \leq j \leq |\lambda(\alpha_i)|$  – we assume there is a predetermined ordering of codons for each amino acid.

A *codon’s frequency* is the number of times that it appears in nature, divided by the total number of all codons corresponding to the same amino acid. By nature, we imply codon frequencies present in some reference sequence or set of sequences such as a genome or set of genomes. As an example, if for some amino acid  $\alpha_i$ ,  $|\lambda(\alpha_i)| = 2$ , and the codon  $\lambda_1(\alpha_i)$  is observed 37 times in nature, while  $\lambda_2(\alpha_i)$  is observed 63 times, we can define the relative frequency of  $\lambda_1(\alpha)$  to be  $\frac{37}{37+63} = 0.37$ . Let  $\rho_j(\alpha_i)$  denote the relative frequency of the  $j^{th}$  codon of  $\alpha_i$ ,  $1 \leq j \leq |\lambda(\alpha_i)|$ . Note that  $\sum_{i=1}^{|\lambda(\alpha_i)|} \rho_j(\alpha_i) = 1.0$ , for any  $\alpha_i$ . In the example above, we say that  $\lambda_2(\alpha_i)$  is the *most frequent codon*. Note that it is possible for more than one codon to have this property.

A *codon’s fitness* is the number of times that it appears in nature, divided by the number of occurrences of the corresponding most frequent codon. Returning to our previous example, if an amino acid  $\alpha_i$  has two codons with frequencies  $\rho_1(\alpha_i) = 0.37$  and  $\rho_2(\alpha_i) = 0.63$ , then their fitness values, denoted by  $\tau_1(\alpha_i)$  and  $\tau_2(\alpha_i)$  respectively, are  $0.37/0.67 \approx 0.55$  and  $0.67/0.67 = 1.0$ . Note that a most frequent codon will always have a fitness value of 1.0.

For convenience, we let  $\rho(b_1b_2b_3)$  and  $\tau(b_1b_2b_3)$  denote the relative frequency and fitness, respectively, of the codon represented by the DNA triple  $b_1b_2b_3$ . Furthermore, let  $|\lambda(i)|$  be the number of codons corresponding to the  $i$ th amino acid and  $\tau_{ij}$  denote the relative frequency of the  $j$ th codon of the  $i$ th amino acid, where we assume there is a predetermined

---

<sup>1</sup>The process of gene translation can be thought of more naturally as a mapping of codons to amino acids, however, we define our mapping as the inverse for convenience.

ordering of amino acids.

### 6.1.2 Constraint Satisfaction

For constraint satisfaction, we focus our attention on designing DNA sequences which do not contain any forbidden motif from a predetermined set,  $\mathcal{F}$ . This is a common practice and important for eliminating such things as restriction enzyme recognition sites of a host organism [15] and polyhomomeric repeat regions [40]. A DNA design is said to be *valid* if it satisfies this constraint. We limit our attention to this definition; however, we note that in certain applications, elimination of certain motifs can be seen as an optimization criterion and not a hard constraint. This is the case for the elimination of immuno-inhibitory CpG motifs in mammalian expression vectors [31].

In practice, forbidden motifs are short and we assume their length is bounded by a constant,  $g$ .

**Observation 6.1.1.** *If the largest forbidden motif is of length  $g$ , then any forbidden motif can span at most  $k + 1$  consecutive codons, where  $k = \lceil g/3 \rceil$ .*

One necessary feature of a codon optimization algorithm is an efficient means to detect if a forbidden motif from  $\mathcal{F}$  is present in a potential design. For both algorithms proposed below, we utilize an Aho-Corasick search for this purpose. Briefly, the Aho-Corasick algorithm builds a keyword tree (trie) for  $\mathcal{F}$  and transforms the structure into an automaton with the addition of failure links. Space and time complexity for building the initial structure is  $O(h)$ , where  $h$  is the total length of every motif in  $\mathcal{F}$ . Queries to determine if a sequence  $b$  contains any forbidden motif takes  $O(|b|)$  time [1]. An example of an Aho-Corasick automaton is given in Fig. 6.1. For a detailed description of the algorithm and existing applications of its use in computational biology, the reader is directed to Gusfield [14].

### 6.1.3 Criterion Optimization

We now review two different measures commonly employed for codon optimization, the codon adaptation index (CAI) and the codon deviation index (CDI). The former measures the overall fitness of a sequence based on the fitness of its constituent codons while the latter measures the deviation of observed frequencies of codons in the sequence, to the codon frequencies found in nature.

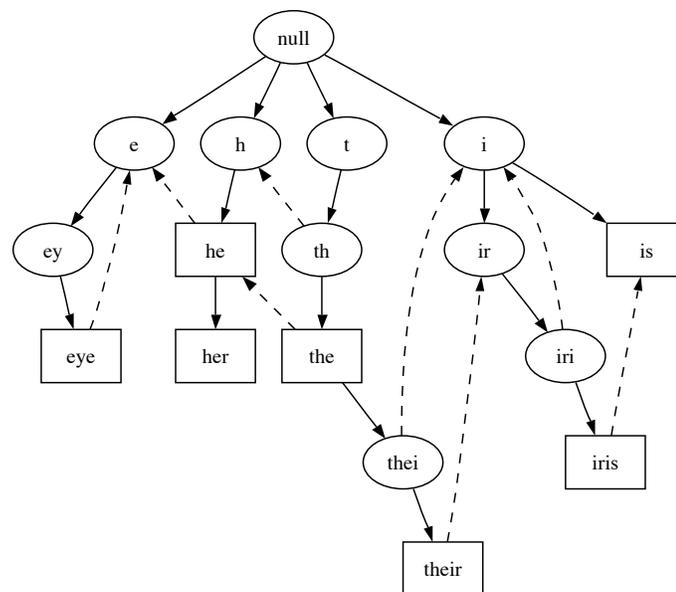


Figure 6.1: An example of an Aho-Corasick automaton. The automaton is for the set of patterns  $\mathcal{F} = \{ \text{eye, he, her, iris, is, their} \}$ . Output nodes are shown as boxes and failure links as dashed edges (failure links leading to the root are not shown). The example has been recreated based on an example found in Dori and Landau [7].

### The Codon Adaptation Index

The codon adaptation index, originally proposed by Sharp and Li [33], is based on the premise of each amino acid having a ‘best’ codon for a particular organism. This perspective evolved from the observation that protein expression is higher in genes using codons of high fitness and lower in genes using rare codons [15]. It is believed this is due to the relative availability of tRNAs within a cell.

The ‘best’ codon, referred to above, is the most frequent codon and therefore, by definition, has a fitness value of 1.0. For some DNA sequence  $S = b_1b_2 \dots b_{3m-1}b_{3m}$ , the CAI value for  $S$ ,  $\text{CAI}(S)$ , can be calculated as in Eqn. (6.1). Based on this definition, if  $S$  consists only of most frequent codons, it would have a CAI value of 1.0.

$$\text{CAI}(S) = \frac{1}{m} \prod_{i=0}^{m-1} \tau(b_{3i+1}b_{3i+2}b_{3i+3}) \quad (6.1)$$

### The Codon Deviation Index

The codon deviation index is based on a more recent theory. In a review of codon bias effects on protein expression, Gustafsson *et al.* [15] suggest three reasons that a ‘best’ codon approach to gene design may inhibit protein expression:

1. Skewed codon usage patterns, relative to those found in genes naturally occurring within the host organism, could result in high usage of only a subset of the tRNA pool, possibly exhausting their availability, and resulting in translational error [22].
2. No flexibility in codon usage could make it impossible to avoid repetitive elements and secondary structure within the gene, possibly inhibiting ribosome processing [13].
3. If codon usage is rigidly fixed, inclusion or exclusion of restriction sites relevant to gene synthesis may be impossible.

We add, however, that it is possible to account for the second and third items, if a less than perfect CAI value is permitted for a gene design.

Despite this characterization in the aforementioned study, to the best of our knowledge, no formal measure has been proposed to quantify the deviation of observed codon usage within a gene relative to the codon usage frequencies found to occur naturally within a host organism. We propose the following measure in Eqn. (6.2) as appropriate for this

purpose, where  $\text{OBS}(b_{3i+1}b_{3i+2}b_{3i+3})$  is the fraction of observed occurrences of the codon  $b_{3i+1}b_{3i+2}b_{3i+3}$ , relative to all codons for the same amino acid, in the DNA sequence  $S = b_1b_2 \dots b_{3m-1}b_{3m}$ . Note that the CDI measure has a maximum value of 2.0, when the observed and the naturally occurring codon distributions are disjoint, and a minimum value of 0.0 when both distributions are identical.

$$\text{CDI}(S) = \frac{1}{m} \sum_{i=0}^{m-1} |\rho(b_{3i+1}b_{3i+2}b_{3i+3}) - \text{OBS}(b_{3i+1}b_{3i+2}b_{3i+3})| \quad (6.2)$$

**Observation 6.1.2.** *It is not always possible to achieve a CDI value of 0.0, regardless of constraints. The lower bound of a CDI value is dependent on the amino acid composition and the naturally occurring codon distribution.*

A simple linear time algorithm for determining the lower bound of the CDI value of a given sequence is given in Algorithm 6.1.

#### 6.1.4 The Codon Optimization Problem

With the previously defined definitions, notation, constraints and optimization criteria, we now formally define the problem of codon optimization.

##### Codon Optimization

*Instance:* Amino acid sequence  $A = \alpha_1\alpha_2 \dots \alpha_{m-1}\alpha_m$ , codon mapping table  $\mathcal{C}$ , a set of forbidden motifs  $\mathcal{F}$  and an objective optimization function  $g$  to be maximized (minimized).

*Problem:* With respect to the codon mapping table  $\mathcal{C}$  find a DNA sequence  $S^*$ ,  $|S^*| = 3m$ , corresponding to  $A$  such that  $S^*$  contains no forbidden motif from  $\mathcal{F}$  as a substring and  $g(S^*) = \max\{g(S) | S \in \mathcal{D}(A)\}$ , where  $\mathcal{D}(A)$  is the set of all valid DNA sequences – sequences not containing a forbidden motif – corresponding to  $A$ . We similarly define  $g(S^*) = \min\{g(S) | S \in \mathcal{D}(A)\}$ , if  $g$  is meant to be minimized.  $S^*$  is an *optimal codon design with respect to  $g$* .

```

Input:  $A$  – an amino acid sequence
Output:  $CDI_{LB}$  – the lower bound CDI value for this instance

 $CDI_{LB} \leftarrow 0$ ;
//  $\text{Frequency}(i)$  counts the occurrences of the  $i$ th amino acid type in  $A$ 
for  $i \leftarrow 1$  to 20 do
  if  $\text{Frequency}(i) = 0$  then continue;
   $numallocated \leftarrow 0$ ;
   $remainingpqueue \leftarrow \emptyset$ ;
  for  $j \leftarrow 1$  to  $|\lambda(i)|$  do
     $expected \leftarrow \tau_{ij} \times \text{Frequency}(i)$ ;
     $numtoallocate \leftarrow \lfloor expected \rfloor$ ;
     $allocated_{ij} \leftarrow numtoallocate$ ;
     $numallocated \leftarrow numallocated + allocated_{ij}$ ;
     $remaining \leftarrow expected - numtoallocate$ ;
    if  $remaining > 0$  then
      |  $\text{AddToPriorityQueue}(remainingpqueue, \{remaining, j\})$ ;
    end
  end
  //  $remainingpqueue$  is sorted in descending order of  $remaining$  value
  while  $numallocated < |\lambda(i)|$  do
    |  $(remaining, j) \leftarrow \text{PopOffPriorityQueue}(remainingpqueue)$ ;
    |  $allocated_{ij} \leftarrow allocated_{ij} + 1$ ;
    |  $numallocated \leftarrow numallocated + 1$ ;
  end
   $CDI_i \leftarrow 0$ ;
  for  $j \leftarrow 1$  to  $|\lambda(i)|$  do
    |  $CDI_i \leftarrow CDI_i + |\tau_{ij} - \frac{allocated_{ij}}{\text{Frequency}(i)}|$ ;
  end
   $CDI_{LB} \leftarrow CDI_{LB} + \frac{\text{Frequency}(i)}{|A|} \times CDI_i$ ;
end
return  $CDI_{LB}$ ;

```

**Algorithm 6.1:** Calculating the CDI lower bound for a sequence.

## 6.2 Codon Optimization Algorithms

### 6.2.1 A Dynamic Programming Algorithm for CAI Optimization

We now propose a linear time and space dynamic programming algorithm guaranteed to maximize the CAI measure, such that the designed DNA sequence contains no forbidden motifs, if such a design exists. Previously, Satya *et al.* [31] proposed an  $\Theta(n^2)$  time and space algorithm, based on a graph theoretic approach of finding a critical path, to maximize the CAI measure, via codon optimization, while first minimizing occurrences of *undesirable* motifs and maximizing occurrences of *desirable* motifs, where  $n$  is the length of the input amino acid sequence and the maximum length of any motif is bounded by a constant. Their description of the problem is more general than ours, however, we conjecture that our algorithm can be extended to handle the general case, whilst maintaining linear time and space complexity.

Our algorithm stores a  $k$ -dimensional entry for each position  $i$ ,  $k \leq i \leq n$ , of the input amino acid sequence, where  $k = \lceil g/3 \rceil$  and  $g$  is the constant bounding the length of any forbidden motif found in  $\mathcal{F}$ . Each entry stores the results of a combination of the previous  $k$  codons, with  $P_{c_k, c_{k-1}, \dots, c_2, c_1}^i$  denoting the entry ending at position  $i$  with  $\alpha_i$  being mapped to the  $(c_1)^{\text{th}}$  codon,  $\alpha_{i-1}$  being mapped to the  $(c_2)^{\text{th}}$  codon, *etc.* The base case occurs when  $i = k$  and is computed as follows. Every combination of codons for the first  $k$  amino acids is evaluated to determine if it fully contains a forbidden motif (Eqn. 6.3, line 1) — note,  $M_{\mathcal{F}}(S)$  is true if and only if the sequence  $S$  contains a forbidden motif. If a  $k$ -sequence of codons does not contain a forbidden motif, a score for the entry is computed as the product of the fitness values for each of the codons in the sequence (Eqn. 6.3, line 2).

$$P_{c_k, c_{k-1}, \dots, c_2, c_1}^k = \left. \begin{array}{l} -\infty \\ \prod_{i=1}^k (\tau_{c_{k-i+1}}(\alpha_i)) \end{array} \right\} \begin{array}{l} \text{, if } M_{\mathcal{F}}(\lambda_{c_k}(\alpha_1)\lambda_{c_{k-1}}(\alpha_2)\dots\lambda_{c_2}(\alpha_{k-1})\lambda_{c_1}(\alpha_k)) = \textit{true} \\ \text{, otherwise} \end{array} \quad (6.3)$$

The recursive case occurs for  $i > k$ . As the base case ensures that any entry not equal to  $-\infty$  is free of forbidden motifs occurring within the corresponding codon sequence, we must simply ensure that no codon at successive positions introduces a forbidden motif that

end within it. By Observation 6.1.1, a forbidden motif could span  $k + 1$  codons. Therefore, it is necessary to evaluate the last  $k + 1$  codons of a potential design to ensure it is free of forbidden motifs and determine which leads to higher score. Due to the above reasons, every valid codon for  $\alpha_{i-k}$  must be tested to ensure a forbidden motif is not being introduced. If no valid design was previously found for the  $k$ -sequence ending at position  $i - 1$ , then appending an additional codon will still result in an invalid design (Eqn. 6.4, line 1). If adding the new codon introduces a forbidden motif, it must be marked as invalid (Eqn. 6.4, line 2). However, if the new codon sequence is free of forbidden motifs, the result at the previous position is multiplied with the fitness of the new codon (Eqn. 6.4, line 3).

$$P_{c_k, c_{k-1}, \dots, c_2, c_1}^i = \max_{1 \leq c_{k+1} \leq |\lambda(\alpha_{i-k})|} \left\{ \begin{array}{ll} -\infty & , \text{ if } P_{c_{k+1}, c_k, \dots, c_3, c_2}^{i-1} = -\infty \\ -\infty & , \text{ if } M_{\mathcal{F}}(\lambda_{c_{k+1}}(\alpha_{i-k})\lambda_{c_k}(\alpha_{i-k+1}) \dots \lambda_{c_2}(\alpha_{i-1})\lambda_{c_1}(\alpha_i)) = true \\ \tau_{c_1}(\alpha_i) \times P_{c_{k+1}, c_k, \dots, c_3, c_2}^{i-1} & , \text{ otherwise} \end{array} \right\} \quad (6.4)$$

Eqn. 6.5 determines the optimal score up to position  $i$  of the input amino acid sequence. Therefore, the optimal CAI value of some input sequence  $A$  of length  $n$  is given by  $\widetilde{P}_k^n$ .

$$\widetilde{P}_k^i = \max_{\substack{1 \leq c_1 \leq |\lambda(\alpha_i)| \\ 1 \leq c_2 \leq |\lambda(\alpha_{i-1})| \\ \vdots \\ 1 \leq c_{k-1} \leq |\lambda(\alpha_{i-k+2})| \\ 1 \leq c_k \leq |\lambda(\alpha_{i-k+1})|}} \left\{ P_{c_k, c_{k-1}, \dots, c_2, c_1}^i \right\} \quad (6.5)$$

Pseudo-code for the above recurrences is given in Algorithm 6.2. We now present a formal proof of correctness for the algorithm, followed by a discussion of the time and space complexity.

**Theorem 6.2.1.**  $\widetilde{P}_k^i$  of Eqn. (6.5) correctly determines the score of an optimal codon design up to the  $i^{\text{th}}$  codon, with respect to the CAI measure, that does not contain any forbidden motifs of maximum length  $3k$ , if such a design exists.

*Proof.* We will argue by induction. First, consider the base case when  $i = k$ .  $\widetilde{P}_k^k$  will determine the maximum score of all the codon designs ending at position  $k$ . The score for each of these designs is determined by Eqn. 6.3. Each design is tested to ensure it is free



of forbidden motifs. If it is, the score is the product of the fitness values for each of the  $k$  selected codons. If it does contain a forbidden motif, a score of  $-\infty$  is assigned. The maximum of these scores is clearly the best solution up to position  $k$ . If that score is  $-\infty$ , then no valid design is possible.

Let us assume that  $\widetilde{P}_k^{i-1}$  correctly determines the optimal score up to position  $i-1$ , if a valid design exists.  $\widetilde{P}_k^i$  must determine the maximum score possible by evaluating all codon designs ending at position  $i$ . These scores are calculated in the recursive case shown in Eqn. (6.4). Consider how a score is determined for an arbitrary assignment of the last  $k$  codons ending at position  $i$ , denoted as  $P_{c_k, c_{k-1}, \dots, c_2, c_1}^i$ . In determining the score of appending the  $c_1$ th codon of  $\alpha_i$ , we must first determine the maximum score of a design ending at position  $i-1$ , having the codon assignments  $c_k, c_{k-1}, \dots, c_3, c_2$  for amino acids  $\alpha_{i-k+1}, \alpha_{i-k+2}, \dots, \alpha_{i-2}, \alpha_{i-1}$ . This can be determined by looking at all  $|\lambda(\alpha_{i-k})|$  possible codon assignments to  $\alpha_{i-k}$  and is given by  $P_{c_{k+1}, c_k, \dots, c_3, c_2}^{i-1}$ . If no valid design exists amongst these possibilities, adding an additional codon at position  $i$  would also be invalid. Of the valid scores, we are guaranteed the associated codon designs do not contain a forbidden motif. However, by Observation 6.1.1 a forbidden motif could be contained within  $k+1$  successive codons. Therefore, we must evaluate the DNA string  $\lambda_{c_{k+1}}(\alpha_{i-k})\lambda_{c_k}(\alpha_{i-k+1})\dots\lambda_{c_2}(\alpha_{i-1})\lambda_{c_1}(\alpha_i)$  to determine if it contains a forbidden motif ending within the codon assignment for  $\alpha_i$ . If a forbidden motif is found, the invalid design is given a score  $-\infty$ , otherwise, the score is defined as the product of the fitness of the new codon assignment with the aforementioned best solution ending at position  $i-1$ . Therefore, by evaluating all assignments to the last  $k$  codon positions ending at position  $i$ ,  $\widetilde{P}_k^i$  is guaranteed to find the score of an optimal design ending at position  $i$ , unless no valid design exists.  $\square$

### Time and Space Complexity

Let  $A$  be an amino acid sequence of length  $n$  and  $\mathcal{F}$  be a set containing forbidden motifs of maximum length  $g$  with  $h$  being the sum of the lengths for all motifs in  $\mathcal{F}$ . Set  $k = \lceil g/3 \rceil$ . We assume  $g$ , and therefore  $k$ , is constant and in practice  $g \ll n$ . An Aho-Corasick automaton containing all forbidden motifs is built only once, in  $O(h)$  time. As the maximum number of codons for any amino acid is 6, the base case must evaluate  $6^k$  possible codon designs in the worst case. Determining if a design of length  $k$  contains a forbidden motif can be accomplished in  $O(3k)$  time and we assume the product of  $k$  numbers can be computed in  $O(k)$  time. Therefore,  $O(6^k \cdot 3k)$  time in total is required to compute the base case. For

every position  $i$ ,  $k < i \leq n$ ,  $O(6^k)$  possible designs must be evaluated. Each of these sub designs could potentially be prefixed by 1 of 6 different codons (at position  $i-k$ ). Evaluating each of these possibilities requires checking for forbidden motifs, in  $O(3(k+1))$  time, and performing one multiplication. Determining the best of the previous codons therefore takes  $O(6 \cdot 3(k+1))$  time. All  $O(6^k)$  possible designs at some position  $i$  can therefore be calculated in  $O(6^{k+1} \cdot 3(k+1))$  time. This effort must be repeated  $n-k$  times. The best answer can be determined by finding the maximum of the previously calculated scores at position  $n$  in  $O(6^k)$  time. Therefore, the total time complexity is  $O(h + 6^k \cdot 3k + (n-k) \cdot 6^{k+1} \cdot 3(k+1)) = O(h+n)$ , as  $k$  is constant.

Storing the Aho-Corasick tree takes  $O(h)$  space. Furthermore,  $O(6^k)$  entries must be maintained in the dynamic programming table for each of  $n-k$  codon positions. Therefore,  $O(h + 6^k \cdot (n-k)) = O(h+n)$  space is needed. If only a score is required, the space can be reduced to  $O(1)$ . Finally, we note that in practice  $h \ll n$ .

### 6.2.2 CDI Optimization Algorithm

It is not currently known whether a polynomial time algorithm exists to solve the CDI optimization problem under the constraint that forbidden motifs must be avoided. In the absence of a deterministic algorithm, we propose one based on stochastic local search (SLS) using a state-of-the-art extended ensemble Monte Carlo search method, replica exchange Monte Carlo (REMC). We first present an introduction to the concepts central to REMC search, followed by a discussion of our specific implementation. We then provide some theoretical insight into the search neighbourhood used by our search algorithm, motivating the need for future work on this problem.

#### Replica Exchange Monte Carlo Search

In the following, we provide a brief introduction to replica exchange Monte Carlo search<sup>2</sup>. For an in-depth description of the algorithm including its historical aspects, the reader is referred to the review of extended ensemble Monte Carlo algorithms by Iba [18], which also provides details related to simulated tempering [25] and replica Monte Carlo search [36].

Replica exchange Monte Carlo (REMC) search maintains  $\chi$  independent replicas of a potential solution. Each of the  $\chi$  replicas has an associated temperature value  $(T_1, T_2, \dots, T_\chi)$ .

---

<sup>2</sup>The description of REMC has been adapted from a previous work [37].

Each temperature value is unique and the replicas are numbered such that  $T_1 < T_2 < \dots < T_\chi$ . In our description of the algorithm, we will label the  $\chi$  potential codon designs maintained by the algorithm at any given time with the replica numbers  $(1, \dots, \chi)$  and always associate temperature  $T_j$  with replica  $j$  (for all  $j$  such that  $1 \leq j \leq \chi$ ). Thus, the exchange of replicas is equivalent to (and is commonly implemented as) the swap of replica labels.

Each of the  $\chi$  replicas independently performs a simple Monte Carlo search at the respective temperature setting. The transition probability from some current codon design  $c$  to an alternative codon design  $c'$  is determined using the so-called Metropolis criterion such that

$$Pr[c \rightarrow c'] := \begin{cases} 1 & , \text{ if } \Delta \text{CDI} \leq 0 \\ e^{-\frac{\Delta \text{CDI}}{T}} & , \text{ otherwise} \end{cases} \quad (6.6)$$

where  $\Delta \text{CDI} := \text{CDI}(c') - \text{CDI}(c)$  is the difference in the CDI values between codon designs  $c'$  and  $c$ , and  $T$  denotes the temperature of the replica.

We can represent the current state of the extended ensemble of all  $\chi$  replicas as a vector  $\mathbf{c} := (c_1, \dots, c_\chi)$  shown below, where  $c_j$  is the codon design of replica  $j$ , which (as previously stated) runs at temperature  $T_j$ .

During replica exchange, temperature values of neighbouring replicas are swapped with a probability proportional to their CDI value and temperature differences. An exchange of temperatures, and therefore a relabeling of replicas, affects the state of the extended ensemble  $\mathbf{c}$ . Therefore, we define an exchange between two replicas  $i$  and  $j$  more generally as a transition of the current ensemble state  $\mathbf{c}$  to an altered state  $\mathbf{c}'$ . We define  $l(c_i) = i$ , the current label or replica number, for all  $c_i$ . The probability of a transition from ensemble state  $\mathbf{c}$  to state  $\mathbf{c}'$  by exchanging replicas  $i$  and  $j$  is defined as:

$$\begin{aligned} Pr[\mathbf{c} \rightarrow \mathbf{c}'] &:= Pr[l(c_i) \leftrightarrow l(c_j)] \\ &:= \begin{cases} 1 & , \text{ if } \Delta \leq 0 \\ e^{-\Delta} & , \text{ otherwise} \end{cases} \end{aligned} \quad (6.7)$$

The value  $\Delta$  is the product of the CDI difference and inverse temperature difference:

$$\Delta := (\beta_j - \beta_i)(\text{CDI}(c_i) - \text{CDI}(c_j))$$

where  $\beta_i = \frac{1}{T_i}$  is the inverse of the temperature of replica  $i$ .

Potential replica exchanges are only performed between neighbouring temperatures, since the acceptance probability of the exchange drops exponentially as the temperature difference between replicas increases. Intuitively, replica exchanges are favourable when a promising solution at a high temperature is compared with a less favourable solution at a lower temperature. The more favourable solution should proceed to a lower temperature replica so it may converge on a local (possibly global) minima. The Monte Carlo search for the replica with a less favourable solution at a lower temperature may have stagnated, therefore an exchange to an increased temperature will make it more probable that barriers in the search landscape can be overcome to escape a local minima.

### **Our REMC Algorithm for Optimizing CDI**

We now describe specific implementation details of our proposed REMC algorithm. First, we must distinguish between a *valid search state* and an *invalid search state*. We define a search state to be valid if and only if the corresponding codon design is free of any forbidden motifs. Any other state is considered invalid. An important property of our algorithm is that the search will never proceed to an invalid state. In this way, we can assure an *incumbent solution*, the best solution so far, is guaranteed to be valid. However, to ensure this property is satisfied, the initial state of the search must be valid. As the CAI optimization algorithm is guaranteed to find a valid solution, if one exists, we employ a modified version of it for this purpose.

Recall that the CAI optimization algorithm keeps track of optimal paths to previous sub-solutions. We can easily modify this algorithm to keep track of all valid paths by increasing our space complexity by a constant factor of 6 — as there are always at most one of 6 different codons which could be prepended to a potential  $k$ -sequence of codons ending at some position  $i$ , in the worst case, all 6 could lead to valid paths and must be stored. If there is no valid solution found using this modified CAI optimization algorithm, then we are guaranteed by Thm. 6.2.1 that a solution does not exist and we may terminate our search immediately. However, if a solution is possible, we can randomly choose a path at each point of the traceback from all valid paths to a sub-solution. In this way, we are randomly sampling an initial start state from all valid codon designs. Due to a result we later show in Thm. 6.2.4, this property of our algorithm is necessary to ensure any valid state can be found.

A necessary design decision of any local search is the definition of its search neighbourhood. In our implementation, we have chosen to use the very common  $\delta$ -exchange neighbourhood, with  $\delta = k + 1$ , under the additional constraint that invalid states are not considered to be in any neighbourhood. To distinguish between a  $\delta$ -exchange neighbourhood that does not allow transitions to invalid states from one which does, we refer to the former as a  $\delta$ -valid-exchange neighbourhood. If our input sequence is of size  $m$ , then there exist  $m$  components comprising our search state – that is, the assignments of codons for each amino acid. Therefore, in a  $\delta$ -exchange (and  $\delta$ -valid-exchange), up to  $\delta$  of these codons can be reassigned to a different value. In addition to limiting our search neighbourhood exchange size to  $k + 1$ , we add the restriction that the  $k + 1$  possible codon reassignments must be in consecutive positions within the sequence.

The choice of setting  $\delta = k + 1$  and allowing only consecutive codon exchanges accomplishes two objectives. First, since by Observation 6.1.1 a forbidden motif can span at most  $k + 1$  codon positions, this search neighbourhood ensures that any valid assignment can be explored within a  $k + 1$  length region – this is the basic premise behind the CAI dynamic programming algorithm. For an example of why this may not be possible for an exchange neighbourhood with  $\delta < k + 1$ , refer to Fig. 6.2. However, we make no assertion with regards to a region larger than  $k + 1$ . Second, as only  $k + 1$  consecutive codons are possibly changed, only a constant size region (at most  $3k + 1$ ) must be searched for the introduction of forbidden motifs, which can be accomplished in constant time.

### Stochastic Local Search for the Codon Optimization Problem

We now pause to consider the applicability of using stochastic local search techniques to solve the codon optimization problem. While SLS algorithms are probabilistic, there are distinguishing notions of *completeness* analogous to deterministic algorithms. Hoos and Stützle [16] define what is meant for an SLS algorithm to be *complete*, *incomplete* and *probabilistically approximately complete* (PAC) for decision problems and corresponding definitions for optimization problems. Informally, an SLS search algorithm is probabilistically approximately complete for a given problem, if and only if, the probability of solving any instance of the problem is 1.0 in the limit  $t \rightarrow \infty$ , where  $t$  is the run time. Also, note that a search neighbourhood is considered complete, if and only if, the search can proceed from any state to any other state through a series of search state transitions, given enough time (in the limit  $t \rightarrow \infty$ ). Given this background, we now provide some theoretical insight into

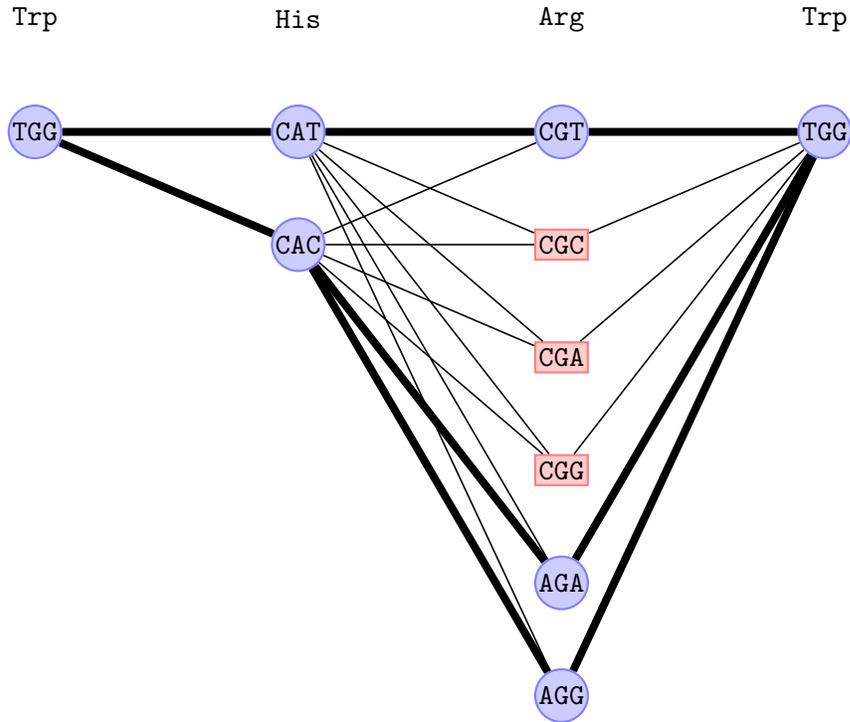


Figure 6.2: An example of an unreachable state when  $\delta < k + 1$ . Shown above is an instance consisting of four amino acids, a forbidden set  $\mathcal{F} = \{ CGC, CGA, CGG, ACC, TAG \}$  and therefore  $k = 1$  and  $\delta = 1$  (by our assumption). Arginine has six corresponding codons, however, three of them appear in  $\mathcal{F}$  and are shown with red boxes. Adjacent codons, those following logically next in the sequence, are shown connected by an edge. Two adjacent codons are shown connected by a bold edge if and only if their concatenation does not introduce a forbidden motif. There are only three valid codon assignments for this problem instance. If the initial codon assignment is the assignment shown in the first row, then there is no valid transition to the other two states without introducing a forbidden motif since  $\delta < 2$ .

```

Input:  $A = \alpha_1\alpha_2\dots\alpha_m$  – an amino acid sequence,  $\mathcal{F}$  – a set of forbidden motifs,  $k$  –
length of longest forbidden motif
Output:  $c^*$  – the sequence with lowest CDI value found

// we assume an optimal design can be returned from OptimizeCAI, not
the optimal score
 $c \leftarrow \text{OptimizeCAI}(A, m, \mathcal{F}, k)$ ;
if  $\text{CDI}(c) = \emptyset$  then                                     // if no design exists, return
| return  $\emptyset$ ;
end
 $\text{CDI}_{LB} \leftarrow \text{CDI}_{\text{lowerbound}}(A)$ ;
// initialize search variables to defaults
 $\chi \leftarrow 5$ ;
 $\phi \leftarrow 5000$ ;
 $T_{min} \leftarrow 1$ ;
 $T_{max} \leftarrow 100$ ;
 $\sigma \leftarrow 15$ ;
// search for optimal design
 $c^* \leftarrow \text{REMCsearch}(c, \text{CDI}_{LB}, \chi, \phi, T_{min}, T_{max}, k + 1, \sigma)$ ;
return  $c^*$ ;

```

**Algorithm 6.3:** A stochastic local search based algorithm for optimizing codon bias with respect to the CDI measure.

```

Input:  $c$  – a valid DNA sequence,  $CDI_{LB}$  – the lower bound for CDI values,  $\chi$  – the
        number of replicas,  $\phi$  – the number of local steps,  $T_{min}, T_{max}$  –
        minimum/maximum search temperatures,  $\delta$  – the maximum number of
        consecutive codons in exchanges,  $\sigma$  – maximum search time
Output:  $c^*$  – the sequence with lowest CDI value found
// initialize variables
 $CDI_{min} \leftarrow \infty$ ;
 $c^* \leftarrow c$ ;
 $offset \leftarrow 0$ ;
 $T_{step} \leftarrow \frac{T_{max} - T_{min}}{\chi - 1}$ ;

// initialize replica specific settings
foreach replica  $i$  in 1 to  $\chi$  do
    |  $c_i \leftarrow c$ ;
    |  $T_i \leftarrow T_{min} + (i - 1) \cdot T_{step}$ ;
end

// search until lower bound reached, or time runs out
while  $CDI_{min} > CDI_{LB}$  and  $ElapsedTime() < \sigma$  do
    | foreach replica  $i$  in 1 to  $\chi$  do
        | |  $c_i^* \leftarrow MCsearch(\phi, T_i, c_i, \delta)$ ;
        | | if  $CDI(c_i^*) < CDI_{min}$  then
        | | |  $CDI_{min} \leftarrow CDI(c_i^*)$ ;
        | | |  $c^* \leftarrow c_i^*$ ;
        | | end
    | end
    |  $i \leftarrow offset + 1$ ;
    | while  $i + 1 \leq \chi$  do
        | |  $j \leftarrow i + 1$ ;
        | |  $\Delta \leftarrow (\beta_j - \beta_i)(CDI(c_i) - CDI(c_j))$ ;
        | | if  $\Delta \leq 0$  then
        | | |  $swapLabels(c_i, c_j)$ ;
        | | else
        | | |  $q \leftarrow \mathcal{U}(0, 1)$ ;
        | | | if  $q \leq e^{-\Delta}$  then
        | | | |  $swapLabels(c_i, c_j)$ ;
        | | | end
        | | end
        | |  $i \leftarrow i + 2$ ;
    | end
    |  $offset \leftarrow 1 - offset$ ;
end
return  $c^*$ ;

```

**Algorithm 6.4:** A replica exchange Monte Carlo algorithm to optimize codon bias with respect to the CDI measure.

```

Input:  $\phi$  – the number of search steps to perform,  $T$  – the search temperature,  $c$  –
         the current codon design, and  $\delta$  – the maximum number of consecutive
         codons in exchanges
Output:  $c^*$  – the best codon design found
 $c^* \leftarrow c$ ;
 $CDI_{min} \leftarrow CDI(c)$ ;
for  $i \leftarrow 1$  to  $\phi$  do
   $c' \leftarrow c$ ;
  // choose an index uniformly at random for a
  // possible exchange of  $\delta$  consecutive codons
   $k \leftarrow \widehat{\mathcal{U}}(1, n - \delta + 1)$ ;
   $c' \leftarrow \text{Exchange}(c', k, \delta)$ ;
  // ensure this state is valid
  if ValidState( $c'$ ) then
     $\Delta CDI \leftarrow CDI(c') - CDI(c)$ ;
    if  $\Delta CDI \leq 0$  then // always accept an improvement
       $c \leftarrow c'$ ;
    else
      // choose a real number uniformly at random in [0,1]
       $q \leftarrow \mathcal{U}(0, 1)$ ;
      if  $q > e^{-\frac{\Delta CDI}{T}}$  then
         $c \leftarrow c'$ ;
      end
    end
    if  $CDI(c) < CDI_{min}$  then
       $CDI_{min} \leftarrow CDI(c)$ ;
       $c^* \leftarrow c$ ;
    end
  end
end
return  $c^*$ ;

```

**Algorithm 6.5:** A Monte Carlo algorithm to optimize codon bias with respect to the CDI measure.

the application of SLS to this problem to first point out possible shortcomings of existing SLS algorithms and finally to prove that our proposed algorithm has the PAC property.

**Lemma 6.2.2.** *The  $m$ -valid-exchange search neighbourhood is guaranteed to be complete for a codon optimization problem instance of size  $m$ , regardless of the value of  $k$ .*

*Proof.* By definition, a search neighbourhood is complete if and only if it guarantees the possibility that given enough time, any state can be reached from any other state through a series of search state transitions. Now suppose our sequence containing  $m$  codon positions begins in a valid state and further suppose there is only one other valid state of all possible assignments to the  $m$  codons. All possible assignments of codons could be enumerated and tested for validity in a finite, albeit exponential, amount of time. When the other valid assignment is found, a transition could occur in one step. Hence, the  $m$ -valid-exchange search neighbourhood is complete for problem instances of size  $m$ .  $\square$

**Lemma 6.2.3.** *A  $\delta$ -valid-exchange search neighbourhood is not guaranteed to be complete for a codon optimization problem instance of size  $m$ , for  $1 \leq \delta < m$  and  $k \geq 1$ .*

*Proof.* We will argue by contradiction. Assume a  $\delta$ -valid-exchange search neighbourhood is guaranteed to be complete for a problem instance of size  $m$ , with  $1 \leq \delta < m$  and  $k \geq 1$ . As any  $\delta$ -valid-exchange search neighbourhood fully contains all potential transitions of a  $(\delta - 1)$ -valid-exchange search neighbourhood, we assume  $\delta = m - 1$ , the maximum value under our assumption. Now consider the problem instance shown in Fig. 6.3 with  $m = 4$  and  $k = 1$ . The problem instance has two valid states and without loss of generality, we assume it is initially in the valid state shown in the top row (*AGGAGGAGGAGG*). Note that a transition must change at least one codon to change the state. As all other codons for Arginine are in the forbidden set, at least one of the *AGG* codons must change to a *CGT* codon. However, note that changing any one codon in this way creates a forbidden motif of *GCG* and/or *GTA* with neighbouring *AGG* codons. Therefore, all  $\delta$  *AGG* codons selected to be exchanged must simultaneously change to *CGT* to remain valid with respect to adjacent codons. However, as one of the codons in the sequence can not be exchanged, since  $\delta < m$ , at least one *AGG* codon is adjacent to a *CGT* codon, introducing a forbidden motif. Therefore, the transition is invalid and can not be accepted. As there does not exist a series of transitions from one state to another within this instance, we have contradicted our assumption. Hence, any  $\delta$ -valid-exchange search neighbourhood is not guaranteed to be complete for a problem instance of size  $m$ , with  $1 \leq \delta < m$  and  $k \geq 1$ .  $\square$

**Theorem 6.2.4.** *A  $\delta$ -valid-exchange search neighbourhood is guaranteed to be complete for a codon optimization instance of size  $m$  and  $k \geq 1$ , if and only if  $\delta = m$ .*

*Proof.* Note that by definition  $\delta \leq m$ . Therefore the proof follows immediately from Lemma 6.2.2 and Lemma 6.2.3.  $\square$

The consequence of Thm. 6.2.4 is that any SLS algorithm relying on a  $\delta$ -valid-exchange search neighbourhood must make other provisions to ensure the PAC property of the algorithm can be guaranteed. We end this section with a proof that our proposed algorithm has this property, however, we first comment on existing SLS algorithms for this problem found in the literature.

There have been a number of SLS algorithms proposed for the version of the problem not considering the elimination of forbidden motifs. These include *The synthetic gene designer* [41], *OPTIMIZER* [28], *DNAworks* [17], and *Codon Optimizer* [10]. We note that the latter identifies restriction sites within the optimized sequence, however, provides no mechanism for automated elimination of these sites. We argue this version of the problem is trivial. The simple algorithm we previously proposed for calculating the lower bound of the CDI value for an instance is guaranteed to find an optimal score (and can be adapted to find a corresponding sequence) and is therefore as effective or more effective than each of the aforementioned algorithms. As the simple algorithm runs in linear time and space, we see no advantage to the use of an SLS algorithm for this purpose.

Next, we consider SLS algorithms which attempt to eliminate forbidden motifs. The algorithm proposed for *UpGene* [11], probabilistically assigns codons for each codon position, tests for forbidden motifs and reiterates the process if any are found. As the algorithm randomly generates a new codon sequence with each iteration, then given enough time, it can be guaranteed that a valid solution will be found. However, the algorithm terminates immediately upon finding a valid solution and never attempts to converge towards an optimal answer. The *Gene Designer* SLS algorithm of Villalobos *et al.* [40] is perhaps the most sophisticated algorithm currently in the literature. The authors use a simple Monte Carlo search algorithm similar to the one used in the subsidiary search of our REMC algorithm. They initialize their search with codons randomly assigned by probability of their frequency. The algorithm then determines if the initial sequence contains forbidden motifs. If it does, the user is warned or asked to eliminate. They do provide a mechanism for elimination of motifs, however, no details are given and no guarantee is made that a valid sequence can be

found. Furthermore, no attempt is made to optimize the CDI score, rather, the objective, by their definition, is to find a sequence which does not contain repeat regions. This is a feature our algorithm currently does not consider, but can be adapted to do so with the use of suffix trees. It should also be noted that their algorithm does not test if forbidden motifs have been introduced through search perturbation steps, however, one could easily extend the algorithm in a similar manner as ours. As their search uses a 1-exchange neighbourhood, therefore allowing transitions to invalid states, the algorithm can be considered to have the PAC property. However, no selection pressure is present to drive the search towards valid solutions and finally no mechanism is in place to determine if an incumbent solution is valid.

Overall, no algorithm was found in the literature which attempts to optimize the CDI value of a design, although other SLS algorithms having the PAC property have been proposed. We conjecture that our algorithm provides a stronger guarantee of finding an optimal solution. We end our discussion by proving our SLS CDI optimization algorithm has the PAC property.

**Theorem 6.2.5.** *Our proposed algorithm for finding an optimal codon design solution, with respect to the CDI measure, under the constraint that forbidden motifs must be avoided is probabilistically approximately complete.*

*Proof.* First, consider that our algorithm is guaranteed to begin in a valid state (free of forbidden motifs), if such a codon design exists. Next consider that our algorithm, using the Metropolis criteria, attempts to converge upon the optimal solution reachable from the initial state, through a series of valid transitions. An invalid state is never accepted, therefore, the incumbent solution is always the best previously observed and is valid. Given sufficient time, the optimal solution which is reachable will be found. However, by Thm. 6.2.4 the  $(k + 1)$ -valid-exchange neighbourhood we employ is not guaranteed to be complete, and therefore it may not be possible to reach other valid states, and hence the global optimal solution. This is fully dependent upon the initial search state. This however can be offset by the introduction of a restart strategy. A static restart strategy reinitializes a search to a new start state before again proceeding. Restarts can be repeatedly applied, until a global optimum is found. Note that the initial state is sampled randomly and uniformly from all valid codon designs and therefore, we are guaranteed that given enough time, a valid search state will be chosen from which the search can converge on the global optimum solution. We end by stating that explicit restarts are not necessary as repeated runs of the

algorithm achieves this guarantee. Therefore, the proposed algorithm is probabilistically approximately complete.  $\square$

### 6.2.3 Designing Homologous Sequences

It was noted by Wu *et al.* [41] that codon optimization has previously been reported to improve protein expression levels in some studies [6, 8, 20, 34, 9], while inhibiting expression in others [3, 42] and overall, the factors affecting protein expression are not yet fully understood. It is therefore beneficial if multiple, each possibly optimal (by our definition), codon designs could be produced as some may be more highly expressible than others. We now briefly describe a process by which multiple homologous sequences can be designed for an input amino acid sequence, having the property that any two of these sequences differ by at least a predetermined threshold.

For this task we modify the SLS CDI optimization algorithm in the following way. First, a valid sequence (or search state) in the CDI algorithm is any one which does not contain a forbidden motif. We extend this definition by adding the constraint that a sequence must also differ from each sequence in a set  $\mathcal{P}$  by at least a percentage  $\gamma$ ,  $0 \leq \gamma \leq 1.0$ . A reasonable value for  $\gamma$  may be 0.25. Second, we repeat the design process, adding a newly designed homologous sequence to the set  $\mathcal{P}$  in each iteration, until some fixed time has elapsed or no additional valid solution is found. We evaluate different values of  $\gamma$  and highlight the utility of this adaptation for collision-aware oligo design in Sect. 6.3.

## 6.3 Experimental Analysis

The effectiveness and runtime efficiency is evaluated for the CAI dynamic programming algorithm in Sect. 6.3.1 and the CDI SLS algorithm in Sect. 6.3.2. The SLS algorithm is also compared against the performance of its subsidiary Monte Carlo search, to determine if there is any benefit gained from an extended ensemble algorithm. In Sect. 6.3.3 we evaluate the hypothesis that multiple homologous DNA sequences, designed through codon optimization, improves the chances for collision free oligo designs by necessitating fewer synthons.

For all experiments run, the same hardware and filtered GENECODE biological test set was used as detailed in Sect. 3.3. Experiment runs to evaluate the CAI and CDI algorithms

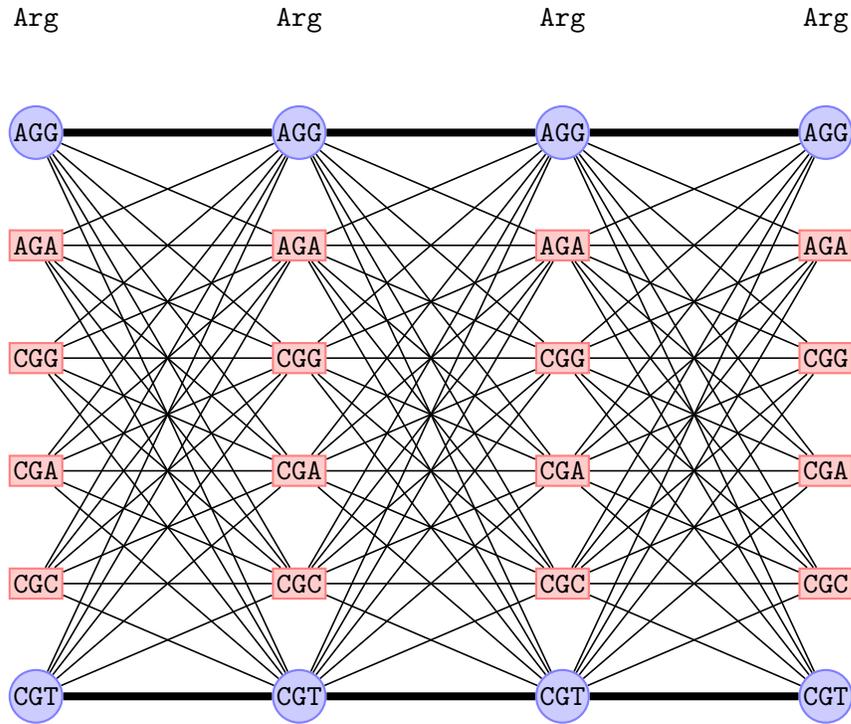


Figure 6.3: A counter example to the claim that a  $\delta$ -exchange search neighbourhood is guaranteed to be complete for a problem instance of size  $m$ , with  $1 \leq \delta < m$  and  $k \geq 1$ . Shown above is an instance consisting of four Arginine amino acids, a forbidden set  $\mathcal{F} = \{CGC, CGA, CGG, AGA, GCG, GTA\}$  and therefore  $k = 1$ . Arginine has six corresponding codons, however, four of them appear in  $\mathcal{F}$  and are shown with red boxes. Adjacent codons are shown connected by an edge. Two adjacent codons are shown connected by a bold edge if and only if their concatenation does not introduce a forbidden motif. Clearly, there are only two valid codon assignments for this problem instance, and when  $\delta < 4$ , there is no possible transition from one to the other.

performance use the entire 3,157 sequences of the set. Experiments conducted for Sect. 6.3.3 use the same 500 randomly chosen sequences evaluated in Chapter 3. In all cases, we use the codon bias of *Escherichia coli* as reported by the Codon Usage Database <sup>3</sup>.

### 6.3.1 Performance of the CAI optimization algorithm

To test the effectiveness of the CAI dynamic programming algorithm and its runtime performance, a forbidden motif set was first constructed which could be considered typical in practice. It is common for a gene synthesis experiment to use a single restriction enzyme. Furthermore, for reasons affecting gene expression, a common task is the removal of polyhomomeric regions (consecutive repeat region of identical nucleotides). Therefore, we have created the forbidden motif set  $\mathcal{F} = \{GAGTC, GACTC, AAAAA, TTTTT, GGGGG, CCCCC\}$  where *GAGTC* is the motif for the *MlyI* restriction enzyme, *GACTC* is its reverse complement and the other motifs ensure no polyhomomeric regions greater than length four are permitted. Results are shown for all 3,157 sequences in Fig. 6.4. On the left side of the figure, the optimal CAI value is plotted against sequence length. A worst CAI value of 0.87 is observed for a short sequence of 100 bases. Optimal CAI values of at least 0.95 were possible for all sequences having length greater than 200 bases. Shown on the right side of the figure is the CPU runtime performance based on sequence length. Clearly, the algorithm scales linearly in practice. The worst runtime of 0.7 seconds is observed for the longest sequence, having 8185 bases.

### 6.3.2 Performance of the CDI optimization algorithm

We adopt a similar experimental protocol to evaluate the CDI algorithm as that used for the CAI algorithm. However, as the CDI optimization algorithm is stochastic in nature, ten independent runs are conducted, having a maximum runtime of 15 seconds, for each sequence and the mean CDI value is reported. If a run reached the CDI theoretical lower bound value for the corresponding sequence, it was immediately terminated. Also, based on previous experience, we fixed the parameters of the REMC search algorithm as follows. Five replicas were used with corresponding temperatures found in the set  $\{1, 25.5, 50.75, 75.25, 100\}$ . Local steps, for Monte Carlo subsidiary search in each replica, was fixed at 5000 iterations.

---

<sup>3</sup><http://www.kazusa.or.jp/codon>

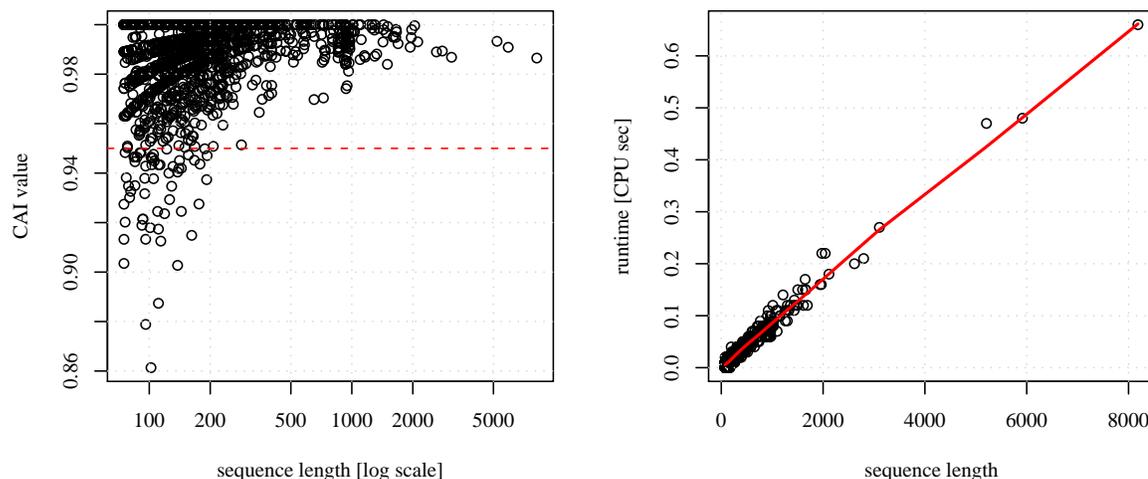


Figure 6.4: Evaluation of the CAI dynamic programming algorithm for each of the 3,157 sequences of the filtered GENECODE data set using a forbidden motif set  $\mathcal{F}$  described in the text. On the left, the optimal CAI value for each sequence is plotted against sequence length (shown in log scale). On the right, CPU runtime performance is plotted for each sequence, against its length and a lowess regression curve of best fit is added.

While, in theory, runtime performance could be improved through parameter tuning, we found these values to be highly effective and present evidence of this below.

Results are shown in Fig. 6.5. On the left of the figure, the difference of the mean CDI value of ten independent runs and the CDI theoretical lower bound is shown plotted against sequence length for all 3,157 sequences. Overall, a solution having the lower bound CDI value is found for 3,092 sequences. We note that due to mandatory elimination of forbidden motifs, the best CDI values found for the other 65 sequences may, in fact, be optimal. Of all sequences with length greater than 300 bases, a solution having the lower bound CDI value was found for all but four. The added design flexibility of longer sequences is apparent and can be directly attributed to the fact that, in general, longer sequences potentially have exponentially more solutions for this problem than shorter ones. In the right side of the figure, the collective runtime distribution is shown for all sequences. The runtime distribution shows that 99% of all runs terminate, due to finding the theoretical lower bound CDI value, in less than 0.2 seconds. All runs which find a theoretical lower bound CDI value terminate after only 7 seconds. The remaining runs correspond to the 65 sequences for which the lower bound CDI value was not reached.

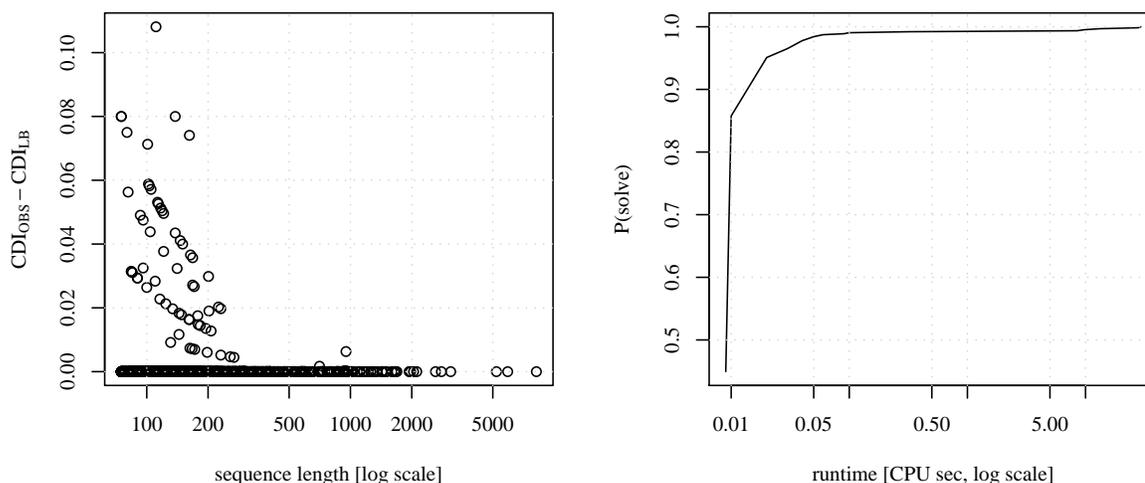


Figure 6.5: Evaluation of the CDI SLS optimization algorithm for each of the 3,157 sequences of the filtered GENECODE data set using a forbidden motif set  $\mathcal{F}$  described in the text. All data points represent the mean of ten independent runs having a maximum runtime of 15 seconds. On the left, the difference between the observed CDI value and the theoretical lower bound is shown plotted against sequence length (shown in log scale). On the right, the runtime distribution of all 3,157 sequences is shown.

### REMC vs. MC performance

While the REMC SLS algorithm has shown to be highly effective and efficient in practice, a natural question to ask is whether or not the same results could be achieved by a single instance of its subsidiary Monte Carlo search algorithm. To objectively evaluate the performance difference between a classical Monte Carlo algorithm and the extended ensemble REMC algorithm employed here, we conduct the following experiment. We first construct a new forbidden motif set such that it is unlikely any solution can be found having a CDI value equivalent to the theoretical lower bound. The purpose for this is to determine if one of the algorithms consistently finds improved solutions over the other. Using a forbidden set where the motifs are easy to eliminate would make finding a solution with a CDI value equivalent to the lower bound more likely. This would, in turn, make it difficult to distinguish performance between algorithms, other than runtime. Therefore, we add short motifs to our forbidden set with  $\mathcal{F} = \{CAGCT, TTT, AAA\}$ . As our REMC search uses five subsidiary Monte Carlo search instances at five different temperatures, comparing with only one Monte Carlo search fixed at one of these temperatures may be an unfair comparison.

Distribution of CDI values Relative to REMC			
Temperature	Improved	Worse	Same
1.00	1	1745	1411
25.25	1	1756	1400
50.75	0	1744	1413
75.25	0	1763	1394
100.00	0	1738	1419

Table 6.1: Performance for all 3,157 sequences, relative to REMC, is shown for each Monte Carlo search fixed at different temperatures.

Therefore, we compare with a Monte Carlo Search performed at each of the five temperatures in the temperature set of our REMC algorithm ( $\{1, 25.25, 50.75, 75.25, 100\}$ ). Each algorithm performed ten independent runs, having a maximum runtime of 15 seconds, for each sequence. The mean CDI value of the ten runs is reported.

In Fig. 6.6, the difference in mean CDI values, between the REMC algorithm and a Monte Carlo algorithm, is shown for each sequence. On the left, the comparison is made with the Monte Carlo algorithm fixed at a temperature of 1.0. In all but one instance, the REMC algorithm finds a solution as good or better than the MC algorithm. Similarly, the right side of the figure contrasts REMC with a Monte Carlo algorithm fixed at temperature 100.0. The MC algorithm does not outperform the REMC algorithm in any instance. However, it is also worth noting that as sequence length increases, the performance difference between REMC and MC algorithms diminishes. Overall the results, relative to the REMC algorithm, are broken down for each temperature the Monte Carlo algorithm is run at and is shown in Table 6.1. Regardless of temperature, the Monte Carlo algorithm is outperformed for at least 55% of the sequences, whereas, of all instances, the REMC algorithm is only outperformed twice.

### 6.3.3 Utility of homologous sequences for synthon design

Intuitively, having more potential oligo designs to choose from can increase the chances that a design can be found which contains few oligo collisions, if any. Using our adapted SLS CDI optimization algorithm, we conduct the following experiment to test this intuition. For each of the 500 sequences where optimal collision-oblivious designs were found in Chapter 3 and

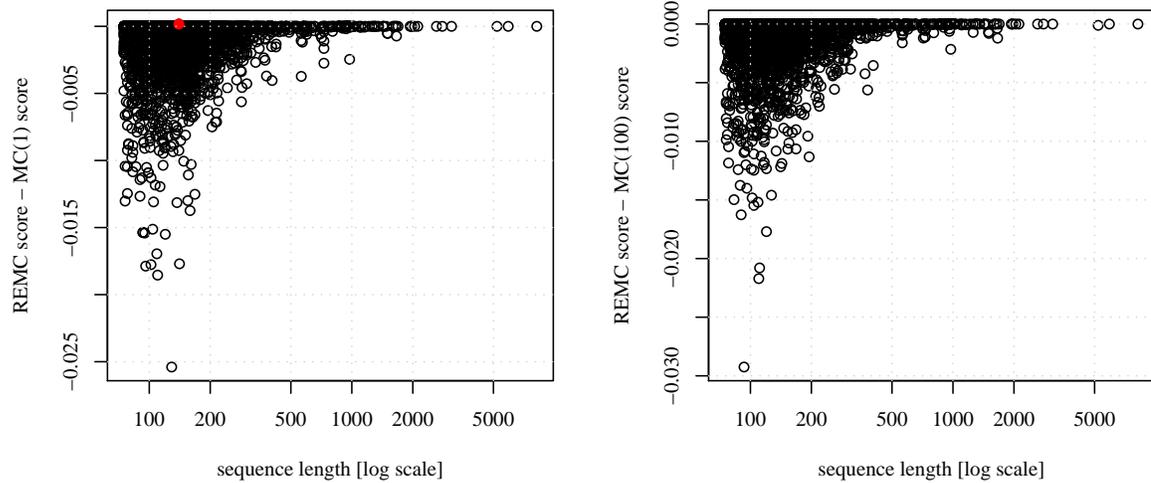


Figure 6.6: The relative difference in mean CDI values for all 3,157 sequences of the filtered GENECODE data set is reported for the REMC algorithm and a Monte Carlo algorithm fixed at a temperature of 1.0 (left) and a Monte Carlo algorithm fixed at a temperature of 100.0 (right). The Monte Carlo algorithm outperforms the REMC algorithm in only one instance (when temperature is fixed at 1.0) and is shown with a filled light circle. As sequence length increases, performance difference between REMC and MC algorithms diminishes.

for which the minimal number of synthons required for those designs to become collision-free were determined in Chapter 5, we attempt to create two different sets of 20 corresponding homologs. The first set is subject to the constraint that every homolog must have a sequence difference of at least 10% to every other sequence in the set. The second set requires a 25% sequence difference between every homolog. In creating each set, we run the SLS CDI algorithm until 20 homologs satisfying the sequence difference constraint are found, or a run of 15 seconds is unable to find a new solution. After the homolog sets are created, the optimal collision-oblivious design is determined for each. All of these designs are evaluated to determine the minimum number of synthons required to make them collision-free. Finally, the best homolog of the set, the one requiring the fewest synthons, is compared against the design of the original sequence.

Results for two comparisons are shown in Fig. 6.7. On the left, designs having a maximum length of 42 bases, with all other design constraints also fixed, are compared between the original sequences and the best homolog from the set having at least 10% sequence difference to any other. In all instances but one, the best homolog requires fewer synthons than

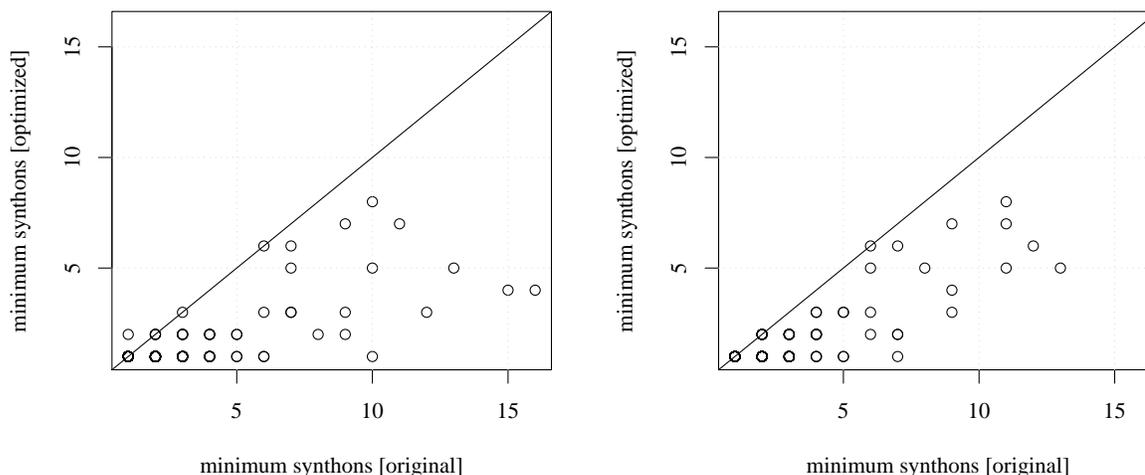


Figure 6.7: A comparison between the number of synthons required for optimal collision-oblivious designs of original sequences and the best design found amongst a corresponding set of homologs. On the left, results for designs having maximal oligo length of 42 and homologs having a sequence difference of at least 10% is shown. On the right, results for designs having maximal oligo length of 52 and homologs having a sequence difference of at least 25% is shown.

the corresponding original sequence. There are also some notable improvements. The two worst cases for number of required synthons in the designs of the original sequences, of 15 and 16 required synthons, is reduced to only requiring 4 synthons by the best corresponding homolog sequences. Similarly, the left side of the figure shows results for designs having maximum oligo length of 52 bases and at least 25% sequence difference between homologs. In this comparison, no instance of the original sequence has fewer synthons than the best corresponding homolog. Overall, the results are reported in Table 6.2 for both sequence difference requirements and both maximum oligo length design constraints. There is a clear advantage for this adaptation in producing designs requiring fewer synthons. However, we end by noting that certain applications in gene design require a specific DNA sequence, and therefore, other methods for eliminating the degree of oligo collisions must continue to be investigated.

$l_{max}$	$\gamma$	Improved	Worse	Same
42	0.10	148	1	351
42	0.25	147	0	353
52	0.10	132	1	367
52	0.25	128	0	372

Table 6.2: A comparison of the minimum number of synthons required for designs of the set of homologs compared with the original sequence, for all 3,157 sequences. An improvement occurs when at least one of the homologs has an optimal collision-oblivious design requiring less synthons than the original sequence.

## Chapter 7

# Conclusions and Future Work

In this thesis, we presented a suite of novel algorithms for *in vitro* gene synthesis. In Chapter 2, we formally defined two variants of the oligo design problem, collision oblivious oligo design, when oligo collisions are not considered, and collision aware oligo design. In Chapter 3 we presented an efficient dynamic programming algorithm for ungapped and gapped collision oblivious oligo design and showed the gapped variant to be highly effective in practice. Our algorithm is the first to guarantee an optimal solution will be found, if one exists. We provided empirical evidence, using a large gene set, that oligo collisions occur infrequently in the designs produced by our collision oblivious algorithm. In Chapter 4, we provided the first results on the computational complexity of oligo design for gene synthesis. We motivate that the general problem is  $\mathcal{NP}$ -hard by proving  $\mathcal{NP}$ -hardness for a related problem, Collision-Aware String Partition (CA-SP). Motivated by this fact and previous evidence suggesting collisions occur infrequently in practice, in Chapter 5 we described and evaluated an efficient synthon partition algorithm which determines the minimal number of regions required to make a design collision free. We have shown that for reasonable parameters, two synthons are usually sufficient to achieve this design goal. In Chapter 6, we formally introduced the codon optimization problem. For the variant of optimizing the codon adaptation index value of a gene, under the assumption forbidden motifs must be avoided, we improved upon the state-of-the-art by proposing a linear time and space dynamic programming algorithm. For the variant concerned with optimizing codon bias, we proposed a stochastic local search algorithm which was shown to be highly effective and efficient in practice. We extended this algorithm to design sets of optimized homologs, having a required sequence difference between all pairs. This adaptation was shown to be

useful for finding designs which require few synthons to become collision free.

Future work includes wet-lab validation that the designs produced by our algorithms result in correct gene assemblies, with high levels of expression. Also, despite motivating evidence suggesting collision aware oligo design is  $\mathcal{NP}$ -hard, the question still remains open. Mañuch *et. al* have extended this result by showing the CA-SP problem remains hard even under a restricted alphabet of size four [24]. What remains is to extend the proof to consider a duplex of strings. Potential improvements to our proposed algorithms should also be considered. While we have purposely separated the problem of codon optimization from oligo design, as certain applications require a specific DNA sequence, we could gain to integrate the two design tasks, when possible. For instance, when oligo collisions occur and the collision graph is produced, codons can be swapped to systematically eliminate these mis-hybridizations. This process, however, is not trivial, since altering the DNA sequence of any oligo could potentially introduce new collisions. Furthermore, it may be possible to solve the codon bias optimization problem with a deterministic algorithm. A promising start may be to capitalize on the fact that it is possible to determine all valid codon designs in linear time. Although there may be an exponential number of valid designs, algorithms such as  $A^*$  search may prove useful to determine an optimal answer, in reasonable time. Finally, as the need for entire genome synthesis increases so too will the requirement that gene synthesis algorithms scale efficiently. A consideration we have ignored in this study is the use of parallel algorithms for the various design tasks. While we have not explicitly outlined the details in the previous chapters, most of the algorithms we propose appear to be trivially parallelizable. However, careful study of this claim must be made to ensure algorithm correctness and relative efficiency can be maintained.

We conclude this thesis by stating that we have improved upon the current state-of-the-art for algorithms concerned with gene synthesis. It is important that we continue to refine these methods as new insight and understanding of gene expression is discovered.

## Appendix A

# Comparison: Existing Gene Synthesis Algorithms

Although theoretical results have been given throughout the thesis as to why our proposed algorithms improve upon the current state-of-the-art, we now provide empirical evidence. As existing software varies greatly in the platforms they support and the options they implement, we have focused on evaluating characteristic performance differences only and have not attempted to compare runtime performance. Certain implementations make it impossible to test a large number of problem instances, therefore, we have selected five sequences at random from the reference filtered GENECODE data set detailed in Chapter 3. The five sequences are listed in Table A.1.

Table A.1: Test sequences for algorithm comparison.

Sequence ID	Sequence
1117	TGTTTCATCCGCAGGGAGCAGGCCAACACATCCTGGCGAGGGTCACGAGGGCCAATTCCTTT CTTGAAGAGATGAAGAAAGG ACACCTCGAAAGAGAGTGCATGGAAGAGACCTGCTCATACGA AGAGGCCCGCGAGGTCTTTGAGGACAGCGACAAGA

Continued on next page

## APPENDIX A. COMPARISON: EXISTING GENE SYNTHESIS ALGORITHMS 71

---

Sequence ID	Sequence
1742	TTCTAATACCCGATTTACAATTACATTGAACTACAAGGATCCCCTCACTGGAGATGAAGAGA CCTTGGCTTCATATGGGATTGTTTCTGGGGACTTGATATGTTTGATTCTTCAAGATGACATT CCAGCGCCTAATATACCTTCATCCACAGATTCAGAGCATTCTTCACTCCAGAATAATGAGCA ACCCTCTTTGGCCACCAGCTCCAATCAGACTAGCATGCAGGATGAACAACCAAGTGATTCAT TCCAAGGACAGGCAGCCCAGTCTGGTGTGTTGGAATGACGACAGTATG
168	GAAGTCACCAAAGCAGTACAGCCTCTCTTACTGGGAAGAATCATAGCTTCCTATGACCCGGA TAACAAGGAGGAACGCTCTATCGCGATTTATCTAGGCATAGGCTTATGCCTTCTCTTTATTG TGAGGACACTGCTCCTACCCCAGCCATTTTGGCCTTCATCACATTGGAATGCAGATGAGA ATAGCTATGTTTAGTTTGATTATAAGAAG
1876	CTGTAGCTGCCCTTTTTGGTCAGCAGCTCCTTGAAGTGGCCAGGGTGACAGCGGGCCCCT CACCAGGGTGCGGTAGGGGATGGGTCCCCCGAGAAGTAGTACGCCACAACGATGCTGTAC ACGGCTGGGCACTCCCGCCGCCACCTTCCTCTGCGATCTTGTC

---

Continued on next page

## APPENDIX A. COMPARISON: EXISTING GENE SYNTHESIS ALGORITHMS 72

---

Sequence ID	Sequence
209	CTGATGCTGTTACCAGGAGGCTGTCTGAGGCACCTGGTCTACAGGATGAGGCACTAACGGGG TTTATGGAAGAGCAAAAGGCAATGGTGGTAGGAATGACAAGGAACTGTCTGAACATGCAGG TTGGTTCAGTCCGGGGTTGCAGCAGGCCAGGCACCCACCTGAGACGTGGCCAGGGCTGAAA CGGCACAGCCTGCAGGTGCCACAGTTAAATCTATGGATGGTTTTGGTGGCAGCTGAGGGGAG GATGACTTAGGAAGTTCTCCTCATGATCACCTGTTCCTTGTGGCAAACCTGGAAGGAGT CGAAGCCACAGTTTTGTTATCAACTTTGGCCCCTGTGTCGAGGCCCTGTCTGTCTATAA TAACCTTGAGTTGGGGGTGTGGTGGAGAAGGAGTTGGGAGAGCCCTGGCTTTTTTGGAGGG ATAGGAGGAGGATTTCTCTGTCAACTCGTGCTACACCATGAGTCTTTAAACTGGTCCGACC AACTGGAGGGTGGGTGCCAACATCCCCTGTTGGGGGCACTCCAGGCCTTGAGGGAGCACCTG CTTGAGGGCTTGAAATCTTGACAGTGCTTGGGTACAGTATTCGAGCTAGTGTGGCC ACTAGGTTGTCAGCACTTGTAGGCGAGACATCTCTTGACGGAGGACTTTGGGTAGTATTTCC ATTTTGGTCTGGGTCGTTAGCATTGCCCTGAAATCTAAATCTAGCTGCTTGGATTCTGGGGT TTAGACCTGGATGCAAAGAGGTGTGGCACATGGTGAATGTAACCTGTGCATAGGTGGAGCT TCGGAGTTCTGAGGAGCTATGCCTGGTGTGTTGAGCGGTGGGAGGGGCAGCGTTACTGGGAAG TGGGGGTGTGCTACTGGTTGGATCTGGTGTGAGCCAGTGCTTGGTCCATTTTCTCAATTT TGTTTGCACTTGAGGTGGGAAAGCGGGTACAGAAGCGCAATCAAGTCACCATAGGAAGCC TGCCTGTCAATACCTGGTCTGGCCATGGTGGCACTGGTGCACACGCTCCCTTTTGCATTTGC AGAACTAGGGGACTCCCTGTGGAAGTTTTACAGGCATGGTTAAAGGCAACTTTTTCATGT GGTGAGCATTTTCTGCACTAGGTCTGTCTGGCATGCAACAGACCTTGTGACAGTTCTTCT GTCCACAGATATGAAACCAAACGCCTATCTTTGTCTCCGTGGAAGAGAGAGGCTTGG TTTGCTGCACTTCCCTTTTTCAGTTGCTCAATCATTTCCTCATCTTGTCTATCTCCTCTT TGAGGTCAGTGGTGTGTGCTTCTTCCCGTTTCAGCTTGGCACGAAGCTGTTCCCGCTCAGTG TCAAACCTCAGAGAGTTGTTTTTCCATCTGAGCTTCCATTTCTGTGCTTCTTCGTTTCTCAGC GGAGAGTTCCTCTTCTAATTCATTCTGCTTTTTTCTTTTCTTCCAGTTTGGCCATTACGT CTTGAGCTTCTGGGCTCCTCTATGACTTTGCCTGAGAGCTGCTTGCACCTTTGACCAGC ATCAGGACCACCTGCTTGTCTTGGCACGCTCTTCTCAAGGCGGGCAGCCAGCTTCTTGTG CTCTTGCTCAAGGGCTGTAGCTGAAGCTTCTCCATTTCCAG

---

## A.1 Oligo Design Algorithms

In this section, we compare our gapped collision oblivious oligo design algorithm with existing algorithms available to us. Each algorithm treats design constraints slightly differently. For instance, in most existing oligo design algorithms, desired oligo length can be input, however, it is not treated as a hard constraint. Furthermore, all algorithms provide the option to optimize oligo overlap melting temperature, but each interpretation is slightly different. For instance, some attempt to minimize a range around a fixed value while others accept any melting temperature within a particular range. We do not seek to provide objective tests to account for all of these differences, since the goal of this evaluation is to demonstrate characteristic performance differences. Therefore, for each of the five sequences, we have run each of the algorithms with the general design goals of 1) providing oligos approximately 40 bases in length, 2) which are unlikely to self-hybridize and 3) have a narrow range for oligo melting temperature near 60°C. We were able to evaluate the Assembly PCR oligo maker [30], denoted as APCROM, DNAWorks [17] and Gene2Oligo [29]. All calculations of melting temperature were made using the programs SimFold and Pairfold [4].

In Fig. A.1 left side, a comparison of the range of overlap melting temperatures is presented for each algorithm for sequence 209 (1.6kb in length). We configured our collision oblivious algorithm to optimize for overlap melting temperature, with a maximum temperature of 60°C. Of all the algorithms, ours reported the narrowest range of overlap melting temperature. On the right side of the figure, the distributions of self-hybridization melting temperatures for each algorithm is reported for the same 1.6kb sequence. Again, our algorithm reports the best results, as the highest melting temperature for any oligo is less than 20°C. Results are similar for the other four sequences (data not shown).

## A.2 Codon Optimization Algorithms

In this section, we compare our REMC CDI optimization algorithm with *UpGene* [11]. We are unable to show results of a comparison with *Gene Designer* [40] due to terms of the software license. As previously mentioned, other algorithms do not attempt to remove forbidden motifs, therefore our comparison is limited to the one algorithm. Also we note that for deterministic CAI optimization algorithms which remove forbidden motifs, we are only aware of the  $\Theta(n^2)$  algorithm of Satya *et al.* [31]. As of this writing we have not

## APPENDIX A. COMPARISON: EXISTING GENE SYNTHESIS ALGORITHMS 74

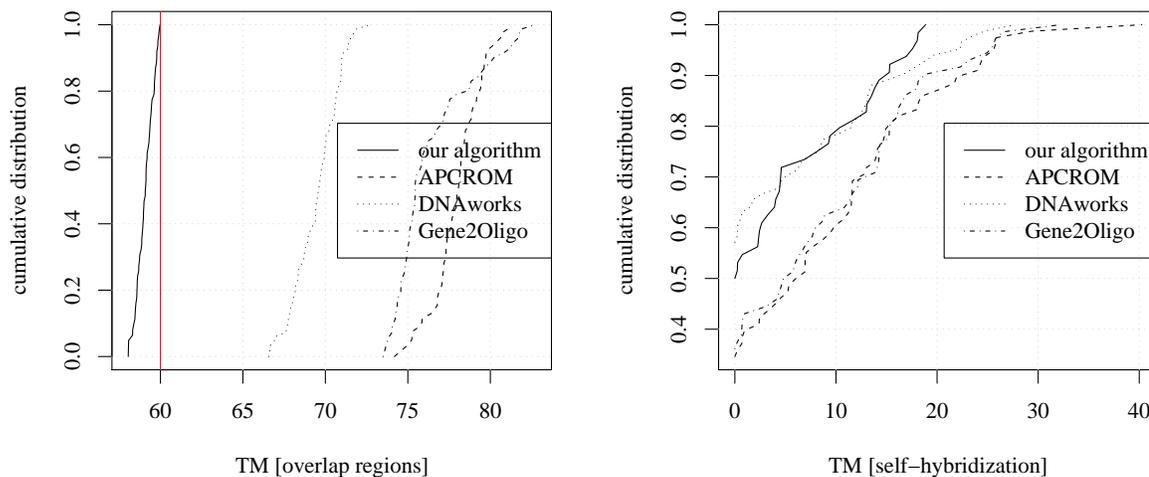


Figure A.1: A comparison of distributions for overlap melting temperature regions and for self-hybridization melting temperatures is shown for oligo designs of four different oligo design algorithms on a 1.6kb sequence.

attained their implementation. However, since our algorithm has a linear time complexity, an empirical comparison is unnecessary.

We limited our REMC algorithm to maximum runtime of 5 seconds for any run. The *UpGene* algorithm was run for five independent runs on each sequence. According to the algorithm description, the search terminates when a valid sequence is found or a set number of iterations of the algorithm has been run. We made no attempt to adjust the maximum number of iteration attempts. Results are shown for both *UpGene* and our algorithm for five independent runs on each sequence. We used the forbidden motif set  $\mathcal{F} = \{CAGCT, TTT, AAA\}$ . For two of the problem instances, including the largest (209), *UpGene* was unable to find a valid design. In addition, a valid design could not be reached for one run on sequence 1117. Our algorithm, however, finds a valid design for all runs. There is no variability in CDI scores between different runs for each sequence, suggesting that these may be optimum scores. Overall, the worst CDI scores attained by our REMC algorithm outperform the best CDI scores attained by *UpGene*, in every instance.

## APPENDIX A. COMPARISON: EXISTING GENE SYNTHESIS ALGORITHMS 75

Sequence ID	% Valid Designs	min CDI score	max CDI score	mean CDI score
<i>UpGene</i>				
1117	80%	0.751562	0.935116	0.862195
1742	0%			
168	100%	0.734483	0.968649	0.872846
1876	100%	0.828317	0.997601	0.891928
209	0%			
<i>REMC CDI optimization algorithm</i>				
1117	100%	0.413213	0.413213	0.413213
1742	100%	0.257367	0.257367	0.257367
168	100%	0.387502	0.387502	0.387502
1876	100%	0.342693	0.342693	0.342693
209	100%	0.117092	0.117092	0.117092

Table A.2: CDI score results for *UpGene* and our algorithm after five independent runs on each of the five test sequences. Our algorithm outperforms *UpGene* for all problem instances.

# Bibliography

- [1] Alfred V. Aho. Algorithms for finding patterns in strings. pages 255–300, 1990.
- [2] B Alberts, A Johnson, J Lewis, M Raff, K Roberts, and P Walter. *Molecular biology of the cell*. Garland, 2002.
- [3] M F Alexeyev and H H Winkler. Gene synthesis, bacterial expression and purification of the rickettsia prowazekii atp/adp translocase. *Biochim Biophys Acta*, 1419(2):299–306, Jul 1999.
- [4] M Andronescu, Z C Zhang, and A Condon. Secondary structure prediction of interacting RNA molecules. *J Mol Biol*, 345(5):987–1001, Feb 2005.
- [5] J C Cox, J Lape, M A Sayed, and H W Hellinga. Protein fabrication automation. *Protein Sci*, 16(3):379–390, Mar 2007.
- [6] T Deng. Bacterial expression and purification of biologically active mouse c-fos proteins by selective codon optimization. *FEBS Lett*, 409(2):269–272, Jun 1997.
- [7] Shiri Dori and Gad M. Landau. Construction of aho corasick automaton in linear time for integer alphabets. *Inf. Process. Lett.*, 98(2):66–72, 2006.
- [8] L Feng, W W Chan, S L Roderick, and D E Cohen. High-level expression and mutagenesis of recombinant human phosphatidylcholine transfer protein using a synthetic gene: evidence for a c-terminal membrane binding domain. *Biochemistry*, 39(50):15399–15409, Dec 2000.
- [9] D Fredman, M Jobs, L Strömqvist, and A J Brookes. Dfold: Pcr design that minimizes secondary structure and optimizes downstream genotyping applications. *Hum Mutat*, 24(1):1–8, Jul 2004.
- [10] A Fuglsang. Codon optimizer: a freeware tool for codon optimization. *Protein Expr Purif*, 31(2):247–249, Oct 2003.
- [11] W Gao, A Rzewski, H Sun, P D Robbins, and A Gambotto. Upgene: Application of a web-based dna codon optimization algorithm. *Biotechnol Prog*, 20(2):443–448, Mar-Apr 2004.

## BIBLIOGRAPHY

77

- [12] Xinxin Gao, Peggy Yo, Andrew Keith, Timothy J. Ragan, and Thomas K. Harris. Thermodynamically balanced inside-out (TBIO) PCR-based gene synthesis: a novel method of primer design for high-fidelity assembly of longer gene sequences. *Nucl. Acids Res.*, 31(22):e143–, 2003.
- [13] K E Griswold, N A Mahmood, B L Iverson, and G Georgiou. Effects of codon usage versus putative 5'-mrna structure on the expression of fusarium solani cutinase in the escherichia coli cytoplasm. *Protein Expr Purif*, 27(1):134–142, Jan 2003.
- [14] Dan Gusfield. *Algorithms on strings, trees, and sequences*. Cambridge Press, 1997.
- [15] C Gustafsson, S Govindarajan, and J Minshull. Codon bias and heterologous protein expression. *Trends Biotechnol*, 22(7):346–353, Jul 2004.
- [16] H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, 2005.
- [17] D M Hoover and J Lubkowski. DNAWorks: an automated method for designing oligonucleotides for PCR-based gene synthesis. *Nuc Acids Res*, 30(10), May 2002.
- [18] Yukito Iba. Extended ensemble monte carlo. *International Journal of Modern Physics C*, 12:623, 2001.
- [19] S Jayaraj, R Reid, and D V Santi. Gems: an advanced software package for designing synthetic genes. *Nuc Acids Res*, 33(9):3011–3016, 2005.
- [20] L Kotula and P J Curtis. Evaluation of foreign gene codon optimization in yeast: expression of a mouse ig kappa chain. *Biotechnology (N Y)*, 9(12):1386–1389, Dec 1991.
- [21] D Kumar, C Gustafsson, and D F Klessig. Validation of RNAi silencing specificity using synthetic genes: salicylic acid-binding protein 2 is required for innate immunity in plants. *Plant J*, 45(5):863–868, Mar 2006.
- [22] C Kurland and J Gallant. Errors of heterologous protein expression. *Curr Opin Biotechnol*, 7(5):489–493, Oct 1996.
- [23] Carole Lartigue, John I. Glass, Nina Alperovich, Rembert Pieper, Prashanth P. Parmar, Clyde A. Hutchison III, Hamilton O. Smith, and J. Craig Venter. Genome Transplantation in Bacteria: Changing One Species to Another. *Science*, page 1144622, 2007.
- [24] Jan Manuch, C Thachuk, and A Condon. A note on the complexity of the oligo design for gene synthesis problem. Manuscript in preparation.
- [25] E Marinari and G Parisi. Simulated tempering: a new monte carlo scheme. *Europhysics Letters*, 19:451–458, 1992.
- [26] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

## BIBLIOGRAPHY

78

- [27] Elizabeth Pennisi. GENETICS: Replacement Genome Gives Microbe New Identity. *Science*, 316(5833):1827a–, 2007.
- [28] P Puigbò, E Guzmán, A Romeu, and S Garcia-Vallvé. OPTIMIZER: a web server for optimizing the codon usage of dna sequences. *Nucleic Acids Res*, Apr 2007.
- [29] J M Rouillard, W Lee, G Truan, X Gao, X Zhou, and E Gulari. Gene2Oligo: oligonucleotide design for in vitro gene synthesis. *Nuc Acids Res*, 32(Web Server issue):176–180, Jul 2004.
- [30] R Rydzanicz, X S Zhao, and P E Johnson. Assembly PCR oligo maker: a tool for designing oligodeoxynucleotides for constructing long DNA molecules for RNA production. *Nucl Acids Res*, 33(Web Server issue):521–525, Jul 2005.
- [31] R V Satya, A Mukherjee, and U Ranga. A pattern matching algorithm for codon optimization and cpg motif-engineering in dna expression vectors. *Proc IEEE Comput Soc Bioinform Conf*, 2:294–305, 2003.
- [32] T Sekiya, T Takeya, E L Brown, R Belagaje, R Contreras, H J Fritz, M J Gait, R G Lees, M J Ryan, H G Khorana, and K E Norris. Total synthesis of a tyrosine suppressor transfer RNA gene. XVI. Enzymatic joinings to form the total 207-base pair-long DNA. *J Biol Chem*, 254(13):5787–5801, Jul 1979.
- [33] P M Sharp and W H Li. The codon adaptation index—a measure of directional synonymous codon usage bias, and its potential applications. *Nucleic Acids Res*, 15(3):1281–1295, Feb 1987.
- [34] G Sinclair and F Y Choy. Synonymous codon usage bias and the expression of human glucocerebrosidase in the methylotrophic yeast, *pichia pastoris*. *Protein Expr Purif*, 26(1):96–105, Oct 2002.
- [35] W P Stemmer, A Cramer, K D Ha, T M Brennan, and H L Heyneker. Single-step assembly of a gene and entire plasmid from large numbers of oligodeoxyribonucleotides. *Gene*, 164(1):49–53, Oct 1995.
- [36] R.H. Swendsen and J.S. Wang. Replica monte carlo simulation of spin glasses. *Physical Review Letters*, 57:2607–2609, November 1986.
- [37] C Thachuk, A Shmygelska, and H Hoos. A replica exchange Monte Carlo algorithm for protein folding in the HP model. *BMC Bioinformatics*, To Appear, 2007.
- [38] The ENCODE Consortium. The ENCODE (ENCyclopedia of DNA elements) project. *Science*, 306(5696):636–640, Oct 2004.
- [39] J Tian, H Gong, N Sheng, X Zhou, E Gulari, X Gao, and G Church. Accurate multiplex gene synthesis from programmable DNA microchips. *Nature*, 432(7020):1050–1054, Dec 2004.

*BIBLIOGRAPHY*

79

- [40] A Villalobos, J E Ness, C Gustafsson, J Minshull, and S Govindarajan. Gene Designer: a synthetic biology tool for constructing artificial DNA segments. *BMC Bioinformatics*, 7:285–285, 2006.
- [41] G Wu, N Bashir-Bello, and S J Freeland. The synthetic gene designer: a flexible web platform to explore sequence manipulation for heterologous expression. *Protein Expr Purif*, 47(2):441–445, Jun 2006.
- [42] X Wu, H Jörnvall, K D Berndt, and U Oppermann. Codon optimization reveals critical factors for high level expression of two rare codon genes in escherichia coli: Rna stability and secondary structure but not trna abundance. *Biochem Biophys Res Commun*, 313(1):89–96, Jan 2004.
- [43] D Zha, A Eipper, and M T Reetz. Assembly of designed oligonucleotides as an efficient method for gene recombination: a new tool in directed evolution. *Chembiochem*, 4(1):34–39, Jan 2003.