Non-Deterministic, Constraint-Based Parsing of Human Genes

A Dissertation

Presented to

The Faculty of the Graduate School of Arts and Sciences

Brandeis University

Department of Computer Science

Jacques Cohen, Advisor

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

by

Olivier Baby

May, 1998

This dissertation, directed and approved by Olivier Baby's Committee, has been accepted and approved by the Graduate Faculty of Brandeis University in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Dean of Arts and Sciences

Dissertation Committee

Prof. Jacques Cohen, Chair

Prof. Martin Cohn

Prof. James Pustejovsky

Prof. Bonnie Berger, MIT

Prof. Laurent Trilling, Université Joseph Fourier

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor, Jacques Cohen, for his guidance and support. Jacques taught me a lot about being a scientist and help me shape my perspectives about the world of research. His constructive criticisms contributed a lot to the quality of this Ph.D. thesis.

I am also indebted to Dave Wedaman, who helped me a lot in the writing of this thesis, and Steve Vacirca, with whom I had passionate and spicy conversations about all and everything including the kitchen sink, and who spent many hours of his valuable time in proofreading this dissertation.

I would also like to thank the friends from the French table who often gave me the pleasant illusion of being home. The French table would have never existed without the talent, determination and passion of Jacob Meunier with whom I had numerous zestful discussions. A special mention needs to be made of Joseph Panepinto – Joe is one of these rare persons who are always in a good humor and fun to be with both socially and intellectually.

I would also like to express my appreciation to Svetlana Shvarstman for her exceptional support and kindness during the first year, and to Kayoko Otani for being such a generous and warmhearted person – Kayo greatly improved the quality of my life over the past year. I am also grateful to my friends in France, in particular Jérôme Moulager and Philippe Servoz, who always welcome me with open arms. For creating a pleasant working atmosphere I also want to thank all the personnel in the computer science department of Brandeis.

Finally, my best thanks of all goes to my parents, Lucien and Denise, and my brother, Jean-Luc, who together formed the caring and supportive substratum of my accomplishments.

ABSTRACT

Non-Deterministic, Constraint-Based Parsing of Human Genes

A dissertation presented to the Faculty of the Graduate School of Arts and Sciences of

Brandeis University, Waltham, Massachusetts

by Olivier Baby

A novel approach is proposed for predicting coding and non-coding regions (i.e., exons and introns) in human genes. Unlike other approaches, the one advocated in this thesis does not use learning methods, sequence comparison, or statistical information. Instead, it relies solely on the identification of the patterns occurring at the boundaries between exons and introns, and on a set of simple constraints. These are formulated in two different ways: one in which directed acyclic graphs are generated using non deterministic finite-state automata, and the other in which more compact, directed acyclic graphs are constructed directly.

The first algorithm consists of defining one or more non-deterministic finite-state automata (NDFSA) that recognize all gene sequences satisfying the various patterns and constraints. Given the NDFSA and a sequence to be analyzed, it is possible to generate a directed acyclic graph (DAG) whose paths specify *all* possible parses, i.e., alternations of exons and introns, in that sequence. The generation of this DAG is performed in linear time even though the number of paths may be exponential. The partitioning of the transitions of the NDFSA in two types, coding and non-coding, allows one to determine gene structure by considering the subsequences of the input sequence that are predicted as non-coding in all parses.

The purpose of the second algorithm is to reduce the memory space used by the first algorithm. This is achieved by a direct construction of a more compact DAG whose paths also describe all possible

parses of the sequence being analyzed. As in the first algorithm, gene structure prediction is performed in linear time by considering the subsequences that are predicted as non-coding in all parses.

The approach has higher accuracy than that achieved by the methods combining different statistical measures, and it reveals the reason why a predicted alternation of exons and introns satisfy the specified patterns and constraints. The two proposed algorithms represent two possible directions within a spectrum of possible means to solve the problem using NDFSA and DAGs.

Contents

1 Introduction **1**

1.1 Motivations and Goals 4

1.2 Bird's-Eye View of the two Proposed Algorithms 6

1.3 Original Contributions 9

1.4 Organization 10

2 Introduction to Computational Molecular Biology **11**

2.1 An Abridged Tutorial to Molecular Biology 12

2.1.1 DNA, RNA and Proteins 12

2.1.2 Introns and Exons 18

2.1.3 On the Sesquipedalian Nature of DNA: the Human Genome Project 22

2.2 Computer Methods for Predicting Exons and Introns 24

2.2.1 Historical Background 26

2.2.2 Search by Signal 27

2.2.3 Search by Content 30

2.2.4 Integrated Gene Identification Systems 35

2.2.5 Linguistics Methods 39

2.2.6 Assessment of Prediction Accuracy 42

2.3 Epilogue: Biology is Far More Complex 47

3 Finite-State Automata Representation of Gene Models 49

3.1 Overview of the NDFSA-based Approach 50

3.2 The Approach, Explained Intuitively 53

3.2.1 Theoretical Motivations 54

3.2.2 Experimental Observations using Yeast Sequences 57

3.3 Evaluation on Human Sequences and Biological Interpretation 62

3.4 Sequential and Parallel Algorithms for Predicting Gene Models 68

3.4.1 DAG Generation 70

3.4.2 Computation of the Ratio $r_t(k)$ and Application of the Criterion 71

3.4.3 An Efficient Parallel Algorithm 74

3.5 Discussion 80

3.5.1 Effectiveness and Expressiveness 80

3.5.2 Relationship with Other Approaches 82

3.5.3 Final Remarks about using NDFSA 83

4 DAG Representation of Gene Models 89

4.1 Overview of the DAG-based Approach 90

4.2 *A priori* Constraints 93

4.2.1 Identification of the Potential Exon and Intron Fragments 93

4.2.2 Construction of the initial unlabeled DAG $\Delta(\sigma)$ 97

4.2.3 Efficient Algorithm for Construction $\Delta(\sigma)$ 99

4.3 Intrinsic Constraints 103

4.3.1 DAG Decoration 106

4.3.2 Set Summary 109

4.3.3 Decoration-Summary System 110

4.3.4 Definition of an Intrinsic Constraint 119

4.3.5 Complexity of the Proposed Construction 121

4.3.6 Composing Constraints 124

4.4 Applications 128

4.4.1 Triplicity Constraint 129

4.4.2 Open Reading Frame Constraint 131

4.4.3 Cardinality Constraint 138

4.4.4 A Simple Example of Composition using the Triplicity and Cardinality Constraints 142

4.5 *A posteriori* Constraints 143

4.6 Relationship with other Approaches 146

4.6.1 Other Approaches using a DAG 146

4.6.2 Using Scoring Functions 148

4.6.3 Determining the Most Probable Path Satisfying a given Constraint 150

4.7 Discussion and Comparison with the NDFSA-based Approach 152

5 Conclusion and Future Work **155**

5.1 Improving Prediction Accuracy 157

5.2 Dealing with Sequencing Errors 159

5.3 Using a more General Representation 160

5.4 Some Other Possible Applications 164

A Glossary **167**

List of Figures

1.1 DNA viewed as an alternation of protein coding genes and non-coding regions 3

1.2 Some of the constraints used to identify exons and introns in genes 6

1.3 Overview of the two algorithms proposed for predicting gene structure 8

2.1 Transcription of DNA into RNA and translation of RNA into protein 13

2.2 The twenty amino acids commonly found in proteins 15

2.3 The genetic code as used by the vast majority of living organisms 17

2.4 Organizational structure of messenger RNA 18

2.5 From DNA to protein: transcription, intron splicing, and translation 20

2.6 The four levels of resolution for analyzing DNA 24

2.7 Consensus sequence for the nuclear and viral protein-coding genes 28

2.8 Position weight matrix for the nuclear and viral protein-coding genes 29

2.9 Partial context-sensitive grammar describing the structure of genes 40

2.10 Sensitivity and specificity 43

3.1 A simple NDFSA generating three possible parses on input string *aaaa* 55

3.2 An NDFSA describing a gene as an alternation of exons and introns 58

3.3 An NDFSA describing a Yeast Saccharomyces Cerevisiae intron 60

3.4 An example of coding distribution 61

3.5 The NDFSA used to parse human genes 64

3.6 Comparison of various NDFSA 65

3.7 Generation of frequency distributions for the toy example of Figure 3.1 69

3.8 Running times of the sequential algorithm for the NDFSA describing human genes 73

3.9 Divide-and-Conquer prefix sum computation 75

3.10 Parallel prefix sum algorithm using a parallel computation of $\oplus$ 75

3.11 Matrix description of the algorithm 77

3.12 Accuracy versus $G + C$ content 84

3.13 Accuracy versus sequence length 85

4.1 Definition of the boolean variables for the $GT - AG$ rule 96

4.2 DAG $\Delta(\sigma)$ for the sequence σ in Figure 4.1 98

4.3 A general algorithm for constructing $\Delta(\sigma)$ 100

4.4 Values of NL and NR for the sequence in Figure 4.1 101

4.5 A linear algorithm for constructing the DAG $\Delta(\sigma)$ 102

4.6 Example of decoration and summary for a given graph G 105

4.7 Decoration and summary 112

4.8 Duplication and selection of the valid paths 113

4.9 The DAGs $G^{\star}$ and G_0 obtained from the DAG G in Figure 4.2 (triplicity constraint) . . . 115

4.10 Algorithm for constructing G_0 from G 123

4.11 Composition of the summaries 127

4.12 The sets $W_{Stop}(i)$ obtained for the DAG in Figure 4.2 134

4.13 The set $W_{Stop}(0)$ obtained for the DAG in Figure 4.2 135

4.14 The DAGs $G^{\star}$ and G_0 obtained from the DAG G in Figure 4.2 (cardinality constraint) . . 141

4.15 Algorithm for applying the criterion 145

5.1 Regular grammar describing a gene as an alternation of exons and introns 160

5.2 Tree representing all parses of the sequence σ in Figure 4.1 162

5.3 The two transformation rules used to obtain the DAG in Figure 4.2 163

Chapter 1

Introduction

In computer science, there exist several situations in which semantically useless information is interspersed within valuable information. For example, contiguous memory locations may contain alternations of useful and useless data, the latter being reclaimable by a fairly complex process called *garbage collection*, which has been extensively studied by computer scientists. Another comparable situation occurs when attempting to simplify terms in mathematical expressions that are equal to zero but may not be immediately recognizable as yielding zero as a final result, e.g.:

$$\cos^2 x - \cosh x \sqrt{1 - \tanh^2 x} + (\tan x + \cot x) \tan x + \sin^2 x - \sec^2 x$$

These examples illustrate the difficulty of distinguishing sequences of semantically useful information interrupted by useless ones within a given context. This dissertation addresses a particularly important instance of this problem that arises in molecular biology.

New developments in Computer Science and Molecular Biology are causing these two fields to interact, and even merge, in revolutionary ways. An entire new generation of computers based on our knowledge of biochemistry and potentially capable of solving NP-complete problems has attracted interest (Adleman, 1994; Bach et al., 1996), and new discoveries using silicon-based "molecular spies" are awaited in molecular biology (Nat.Gen., 1996). The osmosis between these two fields has resulted in the recent appearance of new university courses that introduce students to the fields of *bioinformatics*, also known as *computational molecular biology*, and *biocomputing*. In bioinformatics, computers are used to solve problems arising in molecular biology whereas in biocomputing, knowledge about properties of molecules is used to create new kinds of computers. But the impact of this revolution is going beyond the boundaries of academia: it is also reaching the public, whether it is through fictional movies or the use of DNA evidence in criminal trials. The leader of this revolution is the molecule called *DNA*.

The DNA molecule is the physical support of heredity and the record of the evolution process. It contains all the information necessary for building and maintaining all forms of life, from simple bacteria to complex human beings. DNA is a sequence of four elements known as *bases* (i.e., adenine, cytosine, guanine, and thymine), which when represented by their initials, can be conceived of as a long text. The most important segments in this text are the *genes*, which code for *proteins*. Proteins are the fundamental building blocks of life: they provide the structural components of cells and tissues and take part in essential biochemical reactions.

Since the discovery by James Watson and Francis Crick in 1953 of the double helical structure that forms DNA, the field of molecular biology has been growing with remarkable speed. Yet the interrelationships between the sequence, structure, and biological function of DNA and the role of DNA in evolution are still not clearly understood. A considerable amount of research is being undertaken to clarify these relationships by studying the statistical and physiochemical properties of known DNA sequences. However, despite the increasing number of computational and technological innovations for analyzing sequences (i.e., making biological inferences from the known sequences), our understanding of DNA is still incomplete. Current sequence analysis is a painful compromise between what is desired and what is possible. The two most important obstacles in sequence analysis are (see Figure 1.1):

1. The information contained in DNA is interrupted by large non-coding (intergenic) fragments that

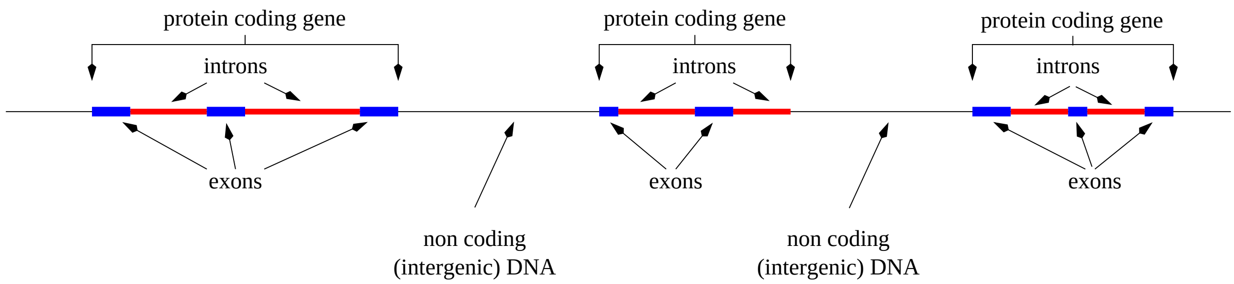


Figure 1.1: DNA viewed as an alternation of protein coding genes and non-coding regions. The genes themselves contain coding regions (exons) and non-coding regions (introns).

do not seem to play any role, i.e., only a small amount of DNA actually codes for genes or proteins.

2. Additionally, the coding information in genes is not always continuous, but scattered in separate blocks called *exons*. The non-coding fragments are called *introns* and are excised from genes by a biochemical process known as *intron splicing*.

Thus, as mentioned at the beginning of this introduction, it is also the case in molecular biology that semantically useful information is interrupted by apparently useless information[1].

Although it is possible, given a DNA sequence, to determine by laboratory methods which of its regions are coding and non-coding, prohibitive cost is a barrier to use in practice[2]. In contrast, computer identification or *prediction* of coding and non-coding regions in a known (i.e., published) DNA sequence is comparably inexpensive but very challenging because of our incomplete understanding of DNA.

This thesis addresses the simpler but still difficult subproblem of predicting exons and introns in genes. In the scientific literature, this problem is usually referred to as the gene structure prediction problem. Although there exist many computer-based techniques to predict exons and introns in a known

[1] Non-coding regions of DNA are often termed "junk DNA".

[2] Automation of DNA sample preparation and data processing yield average costs as high as several thousand dollars per sequence.

DNA sequence, the accuracy of the current existing programs remains relatively low: less than 50% of exons are on average correctly identified according to the study in (Burset and Guigo, 1996). The proposed approach presently achieves a 65% rate of success, but it paves the way for better prediction rates.

1.1 Motivations and Goals

Identifying exons and introns in genes is usually performed by combining a variety of computational approaches[3] such as: statistical tests, neural networks, hidden Markov models, dynamic programming, and formal grammars. The main shortcoming of these methods is that they operate like a *black box*, i.e., the number of (hidden) variables they use is so large that it is virtually impossible to interpret the results of these programs either mathematically or biologically.

This thesis proposes a new approach for predicting gene structure that is motivated by the desire to understand the underlying biological mechanisms. A real understanding of the biology involved requires the use of a simple and explicit model as opposed to a black box model: we will use formal grammars, and directed acyclic graphs, to represent a number of biologically relevant constraints. Our goal is not to simulate biochemical mechanisms — processes like intron splicing are far too complex and too poorly understood at the present time — but to predict exons and introns in a given gene by taking full advantage of our knowledge of the underlying biology.

Currently, there are attempts to use grammars (Searls, 1992; Searls, 1995b) or graphs (Gelfand and Roytberg, 1993; Rogozin et al., 1996; Wu, 1996) to describe genes, but unlike the approach proposed in this thesis, these methods are used as a means to describe an optimization problem in which the optimal exon-intron structures are constructed with respect to some objective function.

Our solution to the gene structure prediction problem also involves grammars and graphs. In this

[3] These are reviewed in the next chapter.

thesis, we provide two efficient algorithms for predicting gene structure. These two algorithms differ only in the way the constraints that define exons and introns in genes are specified. In the first algorithm, the constraints are formulated by using non-deterministic finite-state automata (NDFSA, for short) whereas in the second they are formulated using directed acyclic graphs (DAGs). The input of each algorithm consists of two data: a simple and general description of a gene based on our knowledge of biochemistry, and a gene sequence to analyze. The output consists of one or more solutions, each of which indicates where the coding and non-coding regions are in the input sequence.

More specifically, the first algorithm was designed with the following goals in mind:

- The output should be interpretable from a biological point of view. This rules out the use of black box methods such as neural networks or other learning procedures.

- If the program returns more than one solution, a decision algorithm with a simple mathematical interpretation should select the alternation of exons and introns that occurs in DNA.

- The algorithm should be computationally tractable: a human gene of average length (i.e., about $5,000$ bases long) should be processed in a matter of seconds. If necessary, parallelism should be used to speed up processing times for longer sequences.

- The algorithm should be general. No, or only minor, modifications should be made for detecting structures other than exons or introns.

The main motivation behind the second algorithm was to obtain an optimized version of the first one using an alternative formulation of the problem based on directed acyclic graphs. That algorithm is not as general as the first one, but it uses a more compact data structure and has a lower time complexity. Furthermore, it can be reformulated to introduce statistical information in the description of genes, and to obtain linear variants of the gene structure prediction programs that use dynamic programming and run in quadratic time. Even though both algorithms will involve the construction of DAGs, the first algorithm

Constraint	Description
$GT - AG$ rule	Every intron starts with the pattern GT and ends with the pattern AG
Length constraint	Every intron (or exon) has a minimal given length
Triplicity constraint	The total length of the appended exons is divisible by three
Cardinality constraint	The number of introns (or exons) is smaller than some given constant

Figure 1.2: Some of the constraints used to identify exons and introns in genes.

will be referred to as the NDFSA-based algorithm, and the second as the DAG-based algorithm.

1.2 Bird's-Eye View of the two Proposed Algorithms

As mentioned above, the NDFSA-based algorithm consists of describing exons and introns by using non deterministic finite-state automata. These NDFSA encode a number of very simple yet very selective constraints (see Figure 1.2). Given a DNA sequence known to encode a gene, there may be a very large number of parses of that sequence. However, by selecting the appropriate set of constraints, one can show that, for every input sequence, there is a particular set of transitions in the NDFSA that cannot be avoided no matter how the sequence is parsed (obviously this set of transitions differs from one sequence to another). The key observation in this thesis is that all bases in the DNA sequence that cause the NDFSA to make one of these unavoidable transitions are non-coding, whereas all other bases are coding. In other words, the constraints that we use to describe genes are such that certain regions in a given sequence are non-coding in every interpretation of that sequence. In the proposed algorithm, the parses of a given sequence are succinctly represented by a directed acyclic graph that can be seen as an "unfolding" of the NDFSA over the symbols of the input sequence: the edges of this DAG correspond to the successive transitions of the NDFSA on the input sequence so that the paths between the unique source and the unique sink of the DAG represent all possible parses of the input sequence. Based on the above observation, a heuristic procedure (a criterion) that selects the edges in the DAG that cannot be avoided is then used to determine the coding and non-coding regions in the input sequence.

The DAG-based algorithm relies on the fact that the constraints used in the NDFSA-based algorithm can be divided into three separate classes. It consists of three steps, each of which constructs the set of parses of the input sequence that satisfy the constraints in the corresponding class. This algorithm can be summarized as follows (see Figure 1.3):

- First step: construct a direct acyclic graph based on what we call *a priori* constraints. The simplest *a priori* constraints are the $GT - AG$ rule and the length constraint. The constructed DAG has the following fundamental property: each edge represents an exon or intron fragment, and each path from the single source to the single sink represents a parse of the input sequence satisfying the *a priori* constraints.

- Second step: eliminate the paths in the DAG that do not satisfy what we call *intrinsic* constraints. Unlike the *a priori* constraints, which define the nodes and the edges in the DAG, these constraints operate on the paths in the DAG. Two examples of intrinsic constraint are the triplicity constraint and the cardinality constraint. The DAG obtained after this step represents all parses of the input sequence that satisfy both the *a priori* and intrinsic constraints.

- Third step: further eliminate the paths in the DAG that do not satisfy what we call *a posteriori* constraints. These constraints are very general and can be thought of as heuristics that select one or more particular combinations of introns and exons. An example of *a posteriori* constraint is the above-mentioned heuristic for distinguishing exons from introns.

Note that, in the NDFSA-based algorithm, all possible combinations of exons and introns are also represented by a DAG, but unlike the one introduced above, this DAG is obtained by an unfolding of the NDFSA over the input sequence. Although there exists a one-to-one correspondence between the

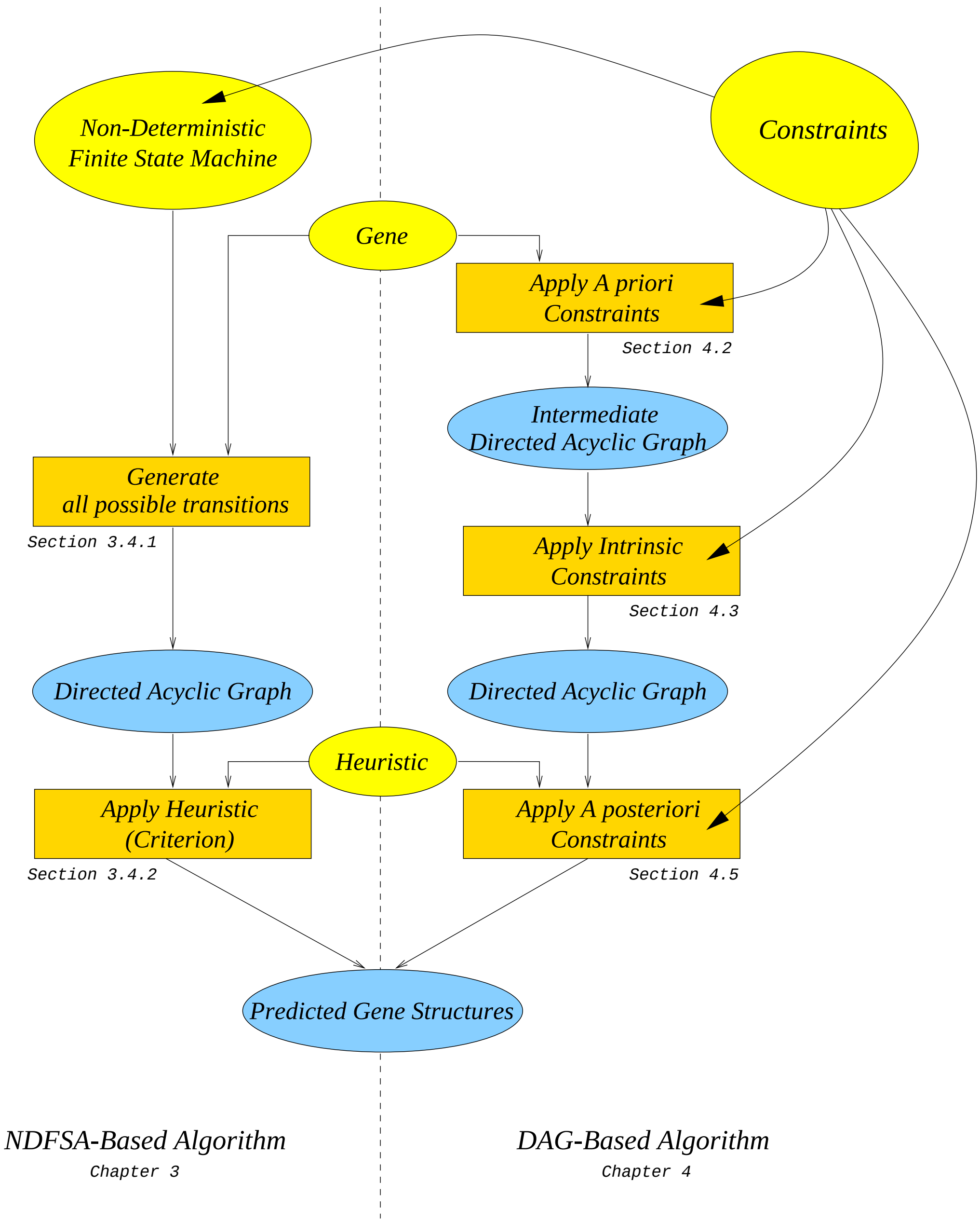


Figure 1.3: Overview of the two algorithms proposed for predicting gene structure. In the first algorithm, the constraints operating on genes are encoded by an NDFSA whereas in the second they are encoded via the concepts of *a priori*, intrinsic, and *a posteriori* constraints. The two algorithms use different DAGs to represent all predicted genes: the two DAGs constructed by the second algorithm are both significantly smaller than the DAG generated from the NDFSA.

paths in the DAG generated from the NDFSA and the paths in the DAG satisfying both the *a priori* and intrinsic constraints, the latter contains a significantly smaller number of nodes and edges than the former.

1.3 Original Contributions

The original contributions of this thesis can be summarized as follows:

- Efficient representation of *all* possible interpretations, i.e., alternations of exons and introns in a given gene, by paths in a DAG. The generation of this DAG is performed in linear time even though the number of paths may be exponential.

- Determination of the alternation that represents the most probable gene structure by considering the subsequences that are predicted as non-coding in *all* interpretations. This is also done in linear time.

- Prediction of gene structure by relying solely on the identification of the patterns occurring at the boundaries between exons and introns, and by using constraints such as those described in Figure 1.2. The approach has higher accuracy than that achieved by the methods combining different statistical measures, and it reveals the reason why a predicted alternation of exons and introns satisfies the specified patterns and constraints.

- Embodiment of the above concepts in the form of two formulations representing two possible directions within a spectrum of possible algorithms: one in which the constraints are encoded at the level of DNA bases via the states and transitions of an NDFSA, and the other in which they are encoded at the level of exons and introns via the edges and paths of a DAG.

1.4 Organization

This thesis is organized as follows:

- In the next chapter, we provide a general introduction to molecular biology and an overview of the current existing computational methods for detecting exons and introns in genes.

- In chapter 3, we give the theoretical and practical motivations for using non-deterministic finite-state automata, and describe a sequential and a parallel algorithms for predicting gene structure. We also evaluate the proposed approach on human genes and provide a biological interpretation of the results. Finally, we discuss the pros and cons of this approach and compare it to other linguistics-based work.

- In chapter 4, we define formally the notions of *a priori*, intrinsic, and *a posteriori* constraints. Given a DNA sequence, we show how one can construct efficiently the two directed acyclic graphs that are involved in the computation of all gene structures satisfying the *a priori* and intrinsic constraints. We also show that every constraint introduced in Chapter 3 (including the heuristic for discriminating exons from introns) can be reformulated as an *a priori*, intrinsic, or *a posteriori* constraint. Finally, we compare this alternate approach with the NDFSA-based one and discuss its relationship with other existing DAG-based approaches.

- In the last chapter, we briefly review the results obtained in the previous chapters and outline some of the promising future developments of this thesis. In particular, it will be seen that, although the NDFSA and DAG-based algorithms described in this dissertation seem to be unrelated, there exists a continuum between the two. The exploration of this continuum is left as an open challenging problem.

Chapter 2

Introduction to Computational Molecular Biology

The molecule that carries the set of instructions to create and maintain a living organism is called DNA. It consists of a long sequence along which some regions encode genes or proteins and others do not (the latter are termed *intergenic* DNA). The protein-coding regions are interrupted by non-coding segments termed *introns*, so that only the coding regions, called *exons*, actually code for proteins. The lack of fast and low-cost laboratory methods for detecting coding or non-coding regions in DNA makes it economically imperative to develop computer algorithms for performing the detection. However, the problem of analyzing a DNA sequence obtained by laboratory methods is far from being trivial: currently, there are no algorithms that can determine the exon-intron structure of a gene with accuracy approaching 100%.

The aim of the first part of this chapter is to acquaint the reader with the leading actors on the genetic scene, i.e., DNA, RNA, proteins, and some of the basic tenets underlying the molecular biology of the cell. The second part provides an overview of the current existing computational approaches for detecting exons and introns. Finally, the last part addresses some of the particular facets of molecular biology that make the design of reliable algorithms a complex task.

2.1 An Abridged Tutorial to Molecular Biology

This section introduces the reader to the world of DNA, RNA, and proteins, which are the bricks and mortar of life on Earth. It provides the essential definitions for understanding the rest of this thesis. Many details have been left out for the sake of simplicity and can be found in (Weinberg, 1985; Felsenfeld, 1985; Darnell Jr., 1983; Darnell Jr., 1985; Doolittle, 1985; Chambon, 1981) and (Hunter, 1993; Griffiths et al., 1993) on which the following material is based. The first part of this section describes the concepts known as *transcription* of DNA into RNA and *translation* of RNA into proteins[1]. The second part introduces the two main components of protein-coding DNA: the *exons*, which code for proteins, and the *introns*, which are intervening sequences. Finally, the last part provides the reader with an idea of the size of the sequences dealt with in this work and the international effort known as the *Human Genome Project*.

2.1.1 DNA, RNA and Proteins

The genetic information in all known living organisms is contained in one or more molecules called *deoxyribonucleic acid* or *DNA*, identical copies of which are found in virtually all cells in an individual organism. A DNA molecule consists of two strands that wrap around each other to form a braid-like structure known as the double helix. A strand of DNA is a chain of *nucleotides*, each containing a sugar (of the deoxyribose type), a phosphate, and one of the four bases: adenine, cytosine, guanine, and thymine, usually denoted by their initial: A, C, G or T. Every A on one strand pairs with a T on the opposite strand, and every C pairs with a G (see Figure 2.1). The bases A and T are said to be complementary, as are the bases C and G. The strands are oriented to indicate their biochemical polarity: one end is

[1] The processus of DNA replication is not explained here; the interested reader is referred to the above references for further information.

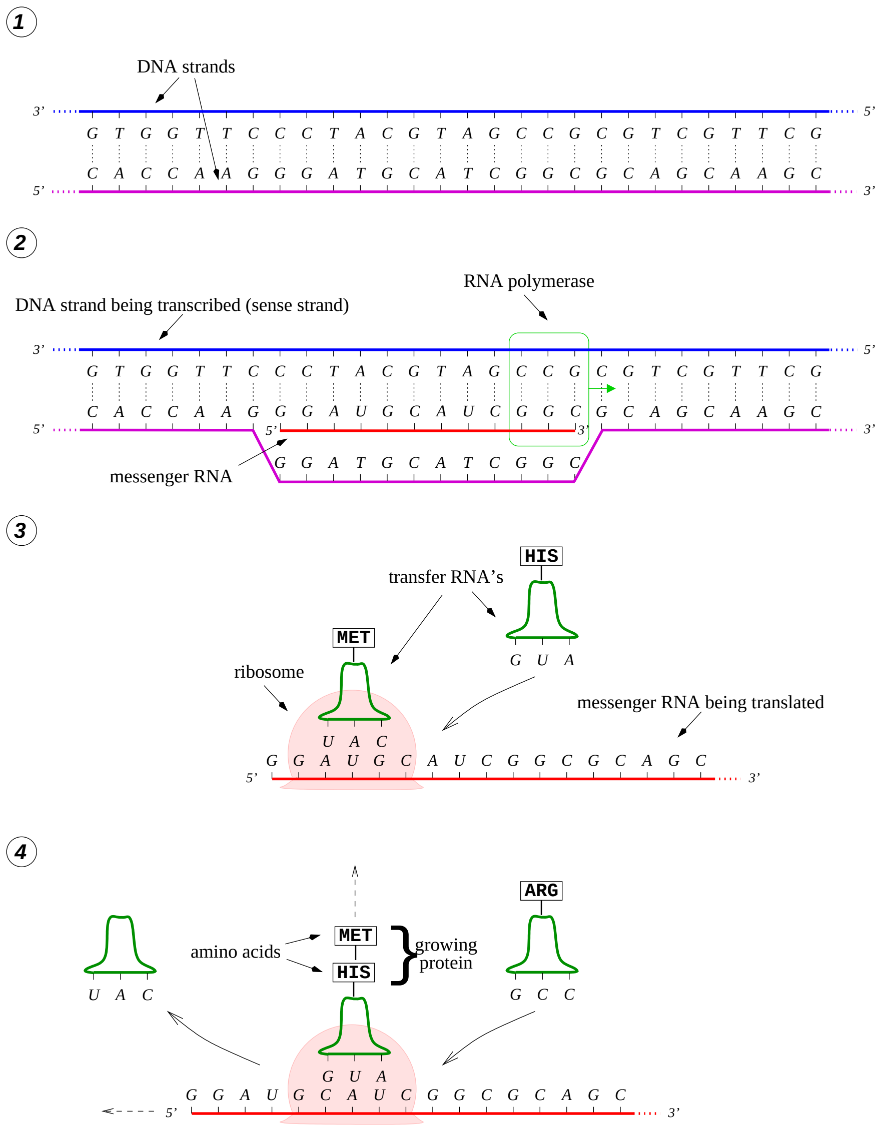


Figure 2.1: Transcription of DNA into RNA and translation of RNA into protein. The DNA helix consists of two complementary strands (1). The sense strand is transcribed into RNA by the enzyme RNA polymerase that travels along the DNA, unwinding its two strands and constructing the complementary RNA chain as it moves (2). The resulting RNA strand is translated into a protein at the ribosome (3): each successive three-base codon calls for the incorporation of a particular amino acid into the growing protein chain (4). Amino acids are abbreviated by their first three letters: *met* codes for methionine, *his* for histidine, *arg* for arginine, etc...

designated $5'$ and the other $3'$ (these numbers refer to the carbon atoms in the 5-carbon sugar that is part of each nucleotide). They run parallel but in opposite direction, i.e., the $5'$ end of one strand coincides with the $3'$ end of the other strand.

The particular order of the nucleotide bases along each strand — named a *DNA sequence* — allows one to determine the sequence on the opposite strand. It specifies the chains of *amino acids*, or *peptides*, which are the basic building blocks of proteins, or *polypeptides* (see Figure 2.2). Proteins are required for the structure, function and regulation of the body's cells, tissues, and organs. Each protein has its own unique vital functions such as oxygen transport or antibody defense. In particular, proteins known as *enzymes* act as catalysts of various chemical reactions. (A catalyst is a substance that speeds up a chemical reaction that would otherwise take too long to be useful.)

The particular regions of DNA that actually code for proteins are referred to as *genes*. Under the influence of various signals — internal or external to the cells — some of the genes are activated or *expressed*, and the base sequence that each carries is copied or *transcribed* into a *ribonucleic acid* strand, or *RNA*. An RNA molecule is identical to a DNA molecule except that it is usually single stranded, it has a ribose sugar (rather than deoxyribose) in its nucleotides, and thymine is replaced with a base named *uracil* that pairs with adenine. There exist different varieties of RNA, each of which performs different functions. For example, the particular RNA strand obtained after the step of transcription is called *messenger RNA* or *mRNA*, and it directs the synthesis of proteins. In the following, we will also introduce two other kinds of RNA: *ribosomal RNA* and *transfer RNA*, both of which also play an important role in the manufacture of proteins.

Although the same DNA is present in every cell, not all genes are active in every cell, so that brain cells for example do not produce the same proteins as muscle cells. Currently, it is not completely understood why two cells with the same set of genes can each make a different array of proteins. In particular, the mechanisms through which the cells in an organism differentiate into brain cells, muscle

	One-letter code	Three-letter code	Name
1	A	Ala	Alanine
2	C	Cys	Cysteine
3	D	Asp	Aspartic Acid
4	E	Glu	Glutamic Acid
5	F	Phe	Phenylalanine
6	G	Gly	Glycine
7	H	His	Histidine
8	I	Ile	Isoleucine
9	K	Lys	Lysine
10	L	Leu	Leucine
11	M	Met	Methionine
12	N	Asn	Asparagine
13	P	Pro	Proline
14	Q	Gln	Glutamine
15	R	Arg	Arginine
16	S	Ser	Serine
17	T	Thr	Threonine
18	V	Val	Valine
19	W	Trp	Tryptophan
20	Y	Tyr	Tyrosine

Figure 2.2: The twenty amino acids commonly found in proteins. They are often abbreviated by their one-letter or three-letter code.

cells, and so on, are not known at the present time[2].

The process of transcribing DNA into RNA, or transcription, is initiated by an enzyme called *RNA polymerase*. This enzyme binds to a particular location on DNA, the *promoter*, which is usually located just before the beginning of a gene, and moves along one of the strands, referred to as the *sense strand* of the gene[3], in the $3'$ to $5'$ direction (see Step 2 in Figure 2.1). This has the effect of unwinding some part of the double helix. The sense strand is used as a template for creating the mRNA strand: as the RNA polymerase stretches the two strands of DNA apart, the RNA nucleotides that are complementary to the DNA nucleotides on the sense strand are successively added to the $3'$ end of the growing mRNA chain. The transcription process ends when a special sequence called the *termination signal* is encountered. The mRNA then separates from the DNA: the sense strand is released and it recombines with the other strand to reform a double helix.

The mRNA obtained serves in turn as a template for the synthesis of a protein: an intracellular machine, the *ribosome*, *translates* the mRNA into a specific protein (see Steps 3 and 4 in Figure 2.1). This process, termed *translation*, begins at a particular location on the RNA called the translation start sequence (usually AUG). This sequence, as well as each following sequence of three mRNA nucleotides, is called a *codon* and encodes a specific amino acid. The table that gives the correspondence between each possible triplet and each amino acid is the so-called *genetic code* (see Figure 2.3). Translation stops when one of the three sequences UAA, UAG, UGA is encountered (see Figure 2.4). The codon AUG occurring at the beginning of a gene is called a *start codon* whereas the triplets UAA, UAG and UGA which do not code for any amino acid are the *stop codons*. The line of nucleotides between and including the start codon and the stop codon on the mRNA strand is called an *open reading frame* (ORF). The combination of the codons in the ORF forms a chain of amino acids, i.e., a protein. A given occurrence of a base is said to be in *reading frame* 0, 1, or 2, if it is respectively the first, second or third base of a

[2] The recent development of a chip that monitors which genes are expressed in a (common yeast) cell at a given time will undoubtedly have an impact on the understanding of gene expression (Nat.Gen., 1996).
[3] Genes can appear on either strand of DNA and therefore may have different sense strands.

First base	Second base				Third base
	g	a	c	u	
g	Gly	Glu	Ala	Val	g
	Gly	Glu	Ala	Val	a
	Gly	Asp	Ala	Val	c
	Gly	Asp	Ala	Val	u
a	Arg	Lys	Thr	Met	g
	Arg	Lys	Thr	Ile	a
	Ser	Asn	Thr	Ile	c
	Ser	Asn	Thr	Ile	u
c	Arg	Gln	Pro	Leu	g
	Arg	Gln	Pro	Leu	a
	Arg	His	Pro	Leu	c
	Arg	His	Pro	Leu	u
u	Trp	STOP	Ser	Leu	g
	STOP	STOP	Ser	Leu	a
	Cys	Tyr	Ser	Phe	c
	Cys	Tyr	Ser	Phe	u

Figure 2.3: The genetic code as used by the vast majority of living organisms. Every codon other than `STOP` corresponds to an amino-acid. Although there are $4^3 = 64$ different possible codons, only 20 different amino acids occur in nature. This redundancy is due to the fact that most amino acids are specified by two or more codons.

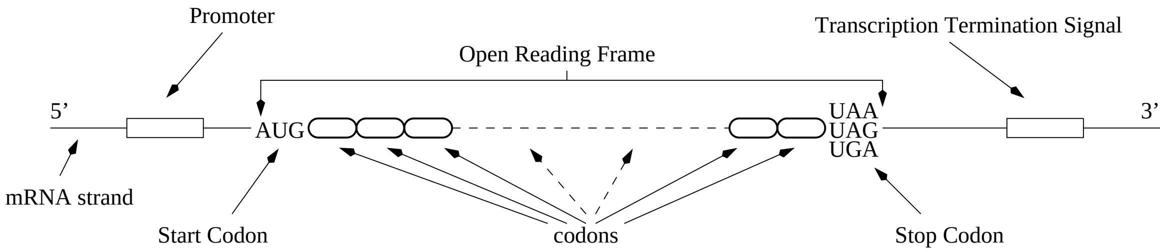


Figure 2.4: Organizational structure of messenger RNA. It consists of, from $5'$ to $3'$: a promoter, a translation start, an open reading frame, a translation stop, and a transcription stop.

codon. A codon is defined as *in-frame* if its first base is in reading frame 0.

The ribosome, where translation takes place, can be viewed as a transducer that reads the open reading frame one codon at a time and produces a protein. It consists of two subunits containing a special kind of RNA called *ribosomal RNA* (*rRNA*), that does not code for proteins, and a set of proteins. The *translation* process relies on the presence of yet another type of non protein-coding RNA molecules called *transfer RNA* (*tRNAs*), whose role is to carry the amino acids to the ribosome for attachment to the growing protein chain.

Note that it has been assumed in the above description that proteins result from the translation of entire genes. However, in general, it may be the case that the sequence of amino acids of a protein does not correspond to the sequence of base triplets of the gene it is translated from. This is because many genes have regions, known as introns, that do not encode amino acids.

2.1.2 Introns and Exons

All living organisms can be classified in four categories:

- The *eukarya*, or *eukaryotes*, include: animals (vertebrates and invertebrates), green plants (e.g. trees, flowers, grasses), fungi (yeast, molds, mushrooms), algae (unicellular to giant seaweeds) and unicellular or multicellular protists (such as amoeba, paramecium or slime molds). Among other features, the cells of eukaryotes contain a membrane-bound nucleus in which their DNA is segregated.

- The *bacteria*, also called *prokaryotes*, are ubiquitous in terms of habitat. Examples of bacteria other than those harmful to the health include the blue-green algae and Escherichia coli, or E. coli, which is a bacterium that lives in human intestines and is required for (normal) digestion. Prokaryotic cells lack a nucleus and many other internal structures characteristic of eukaryotic cells.

- The *archaea*, formerly known as *archebacteria*, share (genetic) similarities with eukarya and bacteria but also have their own marked characteristics. Archea live in extreme environments such as salt collection ponds, sea ice and hot springs.

- The *viruses* are acellular and parasitic, i.e., require a host cell to survive and reproduce. Viruses consist of a small amount of genetic material surrounded by a protein coat. They are well known for their pathogenic properties.

Unlike those in bacteria, archea and viruses, most genes in eukaryotes are discontinuous[4]. The DNA that codes for proteins contains extraneous material that is excised by a ribosome-like assembly, a *spliceosome*, right after it is transcribed into messenger RNA. The intervening non-coding sequences are called *introns* and they typically start with the *dinucleotide* (i.e., sequence of two nucleotides) GT and end with the dinucleotide AG. The other sequences, which are translated into protein, are called *exons*. The RNA obtained after the step of transcription is termed *pre-mRNA*, whereas the RNA obtained after the step of intron splicing is the one actually called mRNA (see Figure 2.5).

The introns come in three different types: introns interrupting the open reading frame between two codons are termed *type* 0 introns; those interrupting a codon between its first base and its second base are called *type* 1 introns; finally those interrupting the second base and the third base of a codon are

[4]Some isolated cases of bacteria and viruses with introns have actually been reported.

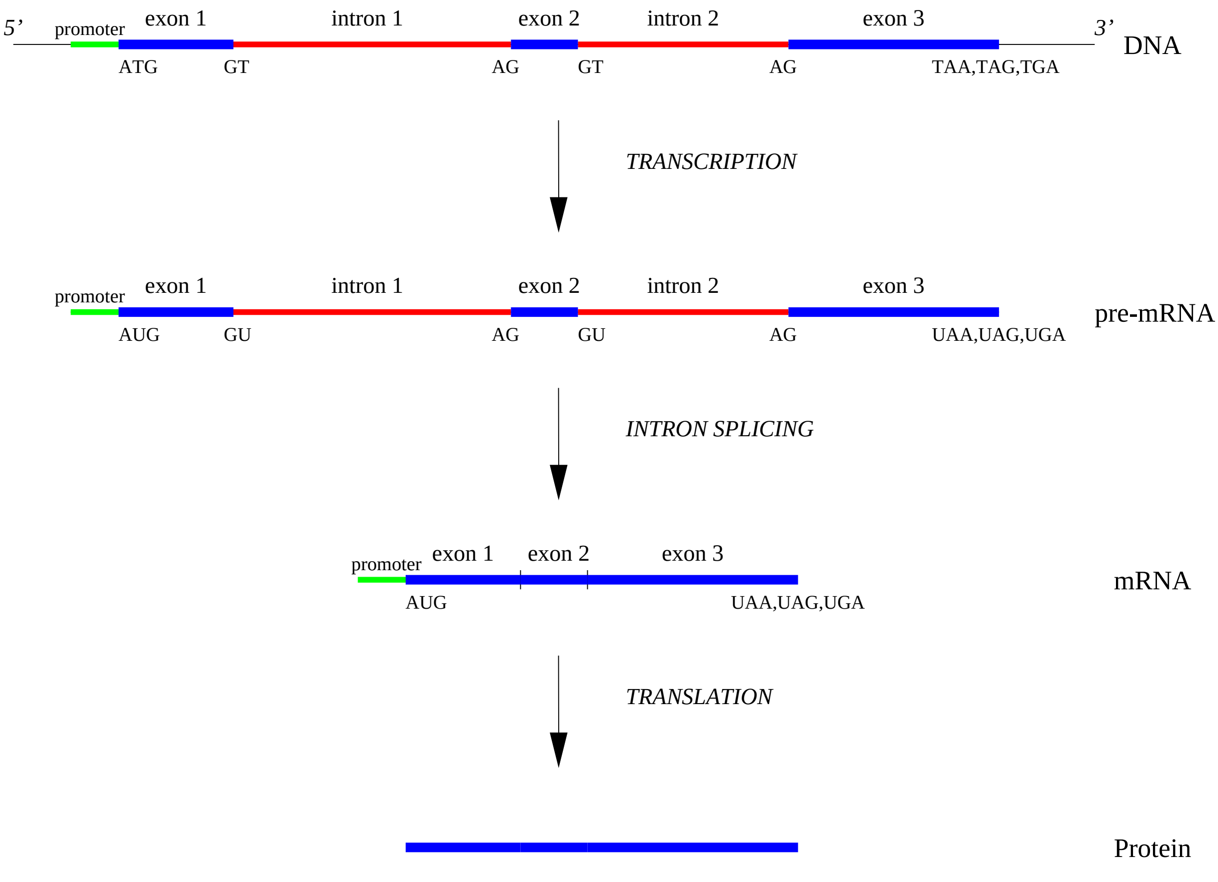


Figure 2.5: From DNA to protein: transcription, intron splicing, and translation.

named *type* 2 introns. The particular locations where a gene is spliced are called *splice junctions*. It is common to distinguish the $5'$ *splice site*, also called the *donor splice site*, which refers to the region around the beginning or $5'$ boundary of an intron from the $3'$ *splice site*, or *acceptor splice site*, which refers to the region around the end or the $3'$ boundary of the intron. It is also common to distinguish *internal exons*, i.e., exons abutted by two introns, from *initial exons* and *terminal exons*, i.e., exons located respectively at the beginning and the end of a gene.

The process of intron splicing is performed in two basic steps. During the first step, the $5'$ end of the intron is cut and it folds back toward its $3'$ end, forming a structure similar to a cowboy *lariat* at a *branch point*, which is usually an adenine upstream of the $3'$ end. In the second step, the $3'$ intron end is cut and both exon ends are joined together.

The existence of introns in eukaryotes is at the source of an unsettled debate about the role and origin of this genetic structure. Proponents of the *introns-early theory* (Gilbert et al., 1986; Senapathy, 1995) claim that introns have always been part of the genetic material and were present in the common ancestors of prokaryotes and eukaryotes but the former lost their introns over the course of evolution. This theory is attacked by the defenders of the *introns-late theory* (Rogers, 1989; Cavalier-Smith, 1985), who counter that introns were inserted into the uninterrupted genes of prokaryotes as they evolved into eukaryotes.

A theory supporting the introns-early hypothesis is that of *exon shuffling* (Long et al., 1995; Souza et al., 1996). According to this theory, today's genes evolved by combining exons from unrelated genes, a process made easier by the presence of introns. This shuffling facilitated the creation of new proteins and accelerated the evolution of organisms. Another theory claims that the presence of introns in genes is the result of a selective pressure to avoid the too frequently occurring stop codons (in exons), and increase the length of proteins (Senapathy, 1988). This theory predicts that there must have existed some mechanism, termed *stop-codon walk mechanism*, to scan a DNA sequence and mark as an intron every region with an in-frame stop codon. One of the explanations for the introns-late theory proposes that *self-splicing* introns (that can splice themselves without the help of a spliceosome) would have been inserted into mRNA and integrated into DNA by a process called *reverse transcription* (Sharp, 1985). Another explanation stipulates that introns arose by the insertion of *transposable elements*, i.e., small, mobile DNA sequences that can replicate, into preexisting genes (Cavalier-Smith, 1985).

The existence of introns has the consequence that a gene might be cut or *spliced* in more than one way and therefore yield different proteins. This phenomenon is known to biologists as *alternative splicing*. For example, the mRNA of the gene for calcitonin is spliced differently in the thyroid and in the pituitary gland, producing different hormones in each gland. Although in most cases only one protein is encoded by a given gene, a current focus of research is to understand the intron splicing mechanism, i.e., determine what proteins are made by a given gene at a given time in a given cell.

2.1.3 On the Sesquipedalian Nature of DNA: the Human Genome Project

The body of a healthy human typically contains 30 trillion cells, most of which contain the same DNA. The size of the *human genome*, i.e., the entire DNA in every human cell, is estimated at 3 billion base pairs, although only 15% of it actually codes for genes. The size of a gene can vary between a few hundred and a few million base pairs (abbreviated *bp*). Most exons have a length ranging between $50bp$ and $300bp$, but it is not unusual to see initial and terminal exons out of that range. Introns are statistically known to be ten fold longer on average than exons; no introns smaller than $60bp$ have been reported so far.

As stated in (DOEa, 1990), the Human Genome Project is a “worldwide research effort with the goal of analyzing all human DNA and determining the location of the estimated $100,000$ human genes”. In conjunction with this effort, the DNA of some model organisms, e.g., yeast and E. coli, are being studied to provide the comparative information necessary for understanding the biology of the cell. To quote (DOEa, 1990), “the information generated by the human genome project is expected to be the source book for biomedical science in the twenty first century and will be of immense benefit to the field of medicine. It will help doctors understand and eventually treat many of the more than $4,000$ genetic diseases that afflict mankind, as well as the many multifactorial diseases in which genetic predisposition plays an important role."

The vast amount of data produced by the Human Genome Project needs to be collected, stored, and distributed. According to the estimate in (DOEb, 1992), "if compiled in books, the data would fill an estimated 200 volumes the size of a Manhattan telephone book (at 1000 pages each), and reading it would require 26 years working around the clock". Four groups have been cooperating in an effort to collect all DNA sequences: one in Japan, one in Europe, and two in the United States. These groups maintain four huge databases: DDBJ (DNA Data Bank of Japan), EMBL (European Molecular Biology Laboratory), Genbank, and GSDB (Genome Sequence DataBase), that are frequently reconciled with each other so that searching one is virtually the same as searching all four. Databases of protein sequences such as: SwissProt, PIR (Protein Identification Resource), and GenPept also exist and can be searched as well.

Comparing a newly determined sequence with one of the databases is an important step in deriving useful information on DNA structure or function. A number of programs, including FASTA (Pearson and Lipman, 1988), BLAST (Altschul et al., 1990) and FLASH (Califano and Rigoutsos, 1993), as well as VLSI chips (Chow et al., 1991; Singh et al., 1996), have been designed to rapidly search each database for a query sequence. As the compared sequences are usually not identical but only similar, these programs are based on approximate matching or alignment methods (see (Altschul, 1997) for a review of these methods). Programs for analyzing the sequence properties of known genes have also been developed, and a large body of literature on the subject of gene finding and gene structure prediction has accumulated in the past twenty years.

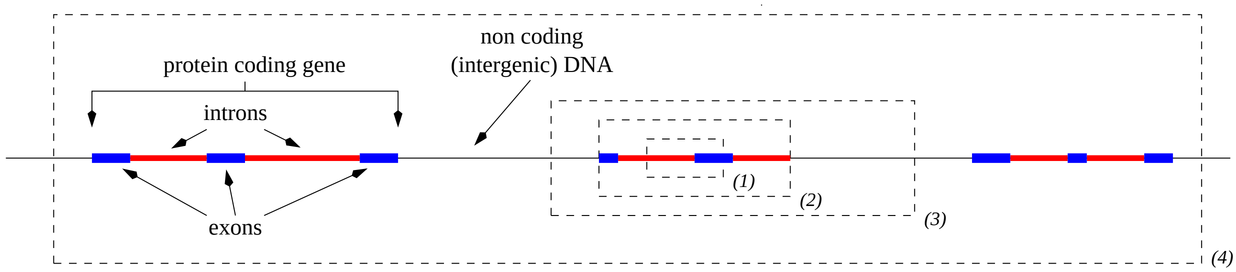


Figure 2.6: The four levels of resolution for analyzing DNA: (1) at the level of a gene fragment, (2) at the level of a complete gene, (3) at the level of DNA containing a single gene, and (4) at the level of DNA containing multiple genes.

2.2 Computer Methods for Predicting Exons and Introns

As noted previously, predicting gene structure, i.e., determining the correct alternation(s) of exons and introns in genes, is a challenging step in sequence analysis. Employing technology currently available, geneticists can chemically extract a fragment of (intron-less) mRNA and compare it with the DNA it originates from, or separate coding regions from non-coding ones (Buckler et al., 1991). However, both approaches involve determining the nucleotide sequence of DNA fragments. This process, called *sequencing*, is both time-consuming and costly: it is estimated that present automated sequencing technology can accurately sequence more than $100,000$ bases per day at a cost of \$.50 per base. As new sequences have been accumulating in the gene databases at an exponential rate, the need for developing analytical tools has been constantly growing over the past few years.

One can essentially distinguish four steps when designing a program for detecting coding regions in DNA sequences (see Figure 2.6). These steps correspond to solving problems of increasing magnitude and difficulty:

1. Given a gene fragment, identify which of its nucleotides are coding or not.

2. Given a (protein-coding) gene, determine the locations where the exons and introns lie.

3. Given a DNA sequence known to contain a unique entire gene, find where this gene is and predict its structure.

4. Given a DNA sequence containing multiple genes, determine which regions of this sequence code for genes.

On one hand, the first of these problems is simpler than the other ones because it does not involve the combinatorial explosion associated with the search over the space of all possible gene structures. Indeed, a simple calculation (Wu, 1996) shows that the number of gene models satisfying the following three constraints:

- introns start with GT and end with AG ($GT - AG$ rule),
- no stop codons but the last one occur in-frame (open reading frame constraint), and,
- the length of the appended exons is divisible by three (triplicity constraint),

grows exponentially with sequence length (more than 10^{100} possibilities for a sequence of length $10,000$)[5]. On the other hand, the constraints operating on entire genes, such as the open reading frame constraint, may be a precious source of information for locating exons and introns because they can be used to rule out impossible combinations, such as those containing in-frame stop codons. In fact, it will be seen in this section that the way that most current programs try to solve the second, third, or fourth problem, is mostly empirical and heuristic, i.e., they make intensive use of computer-based methods for analyzing the data which have been gathered in the databases.

Recall that this thesis focuses on the second of the above-mentioned problems. In what follows, we first provide a brief history of the algorithmic trends in gene structure prediction. We then present the most pertinent methods for splice site detection and gene structure prediction. Finally, we review

[5]However, the set of all gene structures can be constructed in linear time, as will be seen in the next chapter.

the various ways to assess gene structure prediction programs and discuss the importance of exceptions in biology, e.g., genes in which the role between exons and introns is reversed. It should be noted that our aim in this section is not to make an exhaustive description of all existing approaches but to provide a bird's-eye view of current approaches. The reader is referred to (Milanesi et al., 1994) and (Gelfand, 1995) for extensive bibliographies.

2.2.1 Historical Background

Early methods for detecting exons and introns in protein-coding genes were based on the assumption that the sequences bordering the splice junctions were isolated signals whose presence was sufficient and necessary for the splicing reaction to occur. In particular, statistical evidence indicated that the sequences at both ends of the introns were more or less conserved (Sharp, 1981), i.e., remained almost the same from one sequence to another. This observation led to the creation of catalogs of *consensus* sequences (Mount, 1982), which are patterns specifying the most common sequences of nucleotides surrounding the splice sites.

Soon after the publication of the first consensus, it became apparent that the specification of a pair *donor splice site consensus/acceptor splice site consensus* might not be sufficient to accurately bracket the introns (Ohshima and Gotoh, 1987; Claverie, 1992). Indeed, the main difficulties confronting gene prediction program designers are the high heterogeneity in the donor and acceptor splice site population, whether from one genome to another or in the same genome, and the exceptions to the rule, e.g., introns starting with GC instead of GT. Splice site prediction is particularly difficult for genomes in which the sequences at the splice sites are poorly conserved. This is because no particular consensus can be found to distinguish the splice sites from the other signals in the sequence.

To cope with the large variety of patterns occurring at the splice sites, an arsenal of methods have been deployed with more or less success. For purposes of exposition, we group these methods in four different categories:

- The *search-by-signal* methods, which include consensus search and its variants, rely on the presence of regulatory signals in the sequence.

- The *search-by-content* methods discriminate exons from introns by looking at their respective statistical content.

- The *integrated* methods combine search-by-signal and search-by-content methods to assign a score to every potential exon and intron, and determine the gene structure that maximizes the overall score.

- The *linguistic* methods define the rules of gene syntax by stochastic grammars and provide an effective and general formalization of integrated methods.

Another category worth mentioning is that of sequence comparison methods, which have already been discussed in Section 2.1.3.

2.2.2 Search by Signal

Search-by-signal methods rely on elaborate forms of pattern matching to detect splice sites (or other signals). The simplest form of search by signal is that provided by consensus sequences. In the most general case, a consensus is a regular expression that denotes one or more patterns, all of which have the same fixed length. The relative importance of each of the bases in the consensus can be expressed by using a complex pattern descriptor called a *position weight matrix*.

Consensus

The methods based on consensus sequences generally proceed in two steps: the first step consists of determining the consensus, i.e., the patterns that occur most frequently at the splice sites (and possibly nowhere else) in some reference set, e.g., a gene database. For example, an analysis of the respective frequencies with which the four bases A, C, G, T occur at each position in the sequences bordering the

	exon			/	intron					
Position	-3	-2	-1	/	+1	+2	+3	+4	+5	+6
Consensus	C A	A	G	/	G	T	A G	A	G	T

Figure 2.7: Consensus sequence for the donor splice sites of the nuclear and viral protein-coding genes. The symbol "/" indicates the boundary between the exon and the intron. The bases in the consensus are numbered from left to right such that the first base after the splice junction is at position +1 and the first base before the splice junction is at position -1.

donor splice sites of the nuclear and viral protein-coding genes resulted in the derivation of the consensus in Figure 2.7 (Mount, 1982)[6].

The second step consists of using a pattern matching algorithm to detect all occurrences of the consensus in newly sequenced genes. Each occurrence corresponds then to a predicted donor or acceptor splice site. Because the predictive capability of the consensus is usually low, it is in general necessary to define more than one consensus pattern to describe the patterns occurring at the splice sites. It is useful, then, to classify the consensus according to their occurrence frequencies in the reference set (Kolchanov et al., 1991; Kudo et al., 1992). An alternative and improvement to using different consensus is the technique of position weight matrices (Staden, 1984).

Position Weight Matrices (PWM)

A position weight matrix is a matrix whose rows are indexed by the four bases A, C, G, T and whose columns represent the different positions in the splice sites. For each position, the elements of the weight matrix in the corresponding column represent the logarithms of the frequencies with which the bases occur in the reference set. For example, Figure 2.8 provides the PWM corresponding to the donor splice sites of the nuclear and viral protein-coding genes (Mount, 1982).

[6] A similar analysis exists for the acceptor splice sites.

	−3	−2	−1	+1	+2	+3	+4	+5	+6
A	$\underline{-0.40}$	$\underline{-0.19}$	-1.05	$\perp$	$\perp$	$\underline{-0.21}$	$\underline{-0.17}$	-1.05	-0.77
T	-1.15	-0.89	-0.92	$\perp$	$\underline{0}$	-1.22	-0.92	-1.30	$\underline{-0.20}$
C	$\underline{-0.37}$	-0.92	-1.22	$\perp$	$\perp$	-1.70	-1.05	-1.70	-0.92
G	-1.05	-0.92	$\underline{-0.14}$	$\underline{0}$	$\perp$	$\underline{-0.54}$	-0.92	$\underline{-0.08}$	-1.05

Figure 2.8: Position weight matrix for the donor splice sites of the nuclear and viral protein-coding genes. The values in the matrix represent the logarithms ($\perp$ if undefined) of the frequencies with which the four bases occur at each position of the donor splice sites. For each position, the underlined values correspond to the bases that have an occurrence frequency f equal to or larger than 0.5, i.e., $\ln f \geq -0.69$. By selecting these bases, one obtains the consensus of Figure 2.7.

Given a weight matrix F of size $4 \times k$ and a sequence fragment $\rho = \rho_1 \ldots \rho_k$ of length k, a *score* $S_F(\rho)$ is defined as follows:

$$S_F(\rho) = \sum_{i=1}^{k} F(\rho_i, i)$$

i.e., it is the logarithm of the probability that the sequence fragment ρ is of the same nature (i.e., donor or acceptor splice site) as the fragments used to determine the weight matrix F. The search for the donor or acceptor splice sites in a given sequence σ is then performed by calculating the value of $S_F(\rho)$ for every fragment ρ of length k in σ. Fragments that have a score higher than a given threshold are then marked as donor or acceptor splice sites.

It is easily seen that a consensus is a particular case of a position weight matrix. The advantage of PWMs is that each change from a consensus base can be assigned a different probability/penalty. Thus, by using a PWM one can capture mismatches of varying degrees of severity between a sequence fragment and the signal represented by the PWM.

There exist many different ways to define position weight matrices. For example, one can define a

PWM in which the rows are indexed by dinucleotides rather than nucleotides. The elements of the PWM then represent the logarithms of the frequencies with which the dinucleotides occur at each position in the signal. The problem of selecting the most appropriate position weight matrix for a given signal is discussed in (Stormo, 1990) whereas the various existing PWM search programs are reviewed in (Frech et al., 1997).

In (Berg and von Hippel, 1987; von Hippel, 1994), the notion of position weight matrices is extended to that of *discrimination energy* for predicting the signals occurring at the protein binding sites. The central idea of this approach is to define a score that approximates the energy (or strength) of the protein binding sites. In this scenario, a high score corresponds to a pattern that is more likely to bind to a protein. Another generalization of weight matrices is provided in (Barton and Sternberg, 1990; Mulligan et al., 1984), wherein they are combined with variable-length gaps respectively with and without a scoring scheme for the gaps. The most sophisticated approach using PWMs allows *indels*, i.e., insertions and deletions in the sequence being analyzed (Gribskov et al., 1987; Gribskov et al., 1990; Bucher and Bairoch, 1994); this results in the notion of *generalized profiles*. The relationship between the latter and *Hidden Markov Models* is analyzed in detail in (Bucher et al., 1996).

Although position weight matrices allow a better description of the splice sites than simple consensus, they have limited success when not used in conjunction with other methods (Senapathy et al., 1990). Search-by-signal methods often predict more false sites, i.e., sites that are not actual exon-intron boundaries, than true sites (Brunak et al., 1991; Frech et al., 1997). Another approach for predicting exons and introns is to examine their base content.

2.2.3 Search by Content

Algorithms in this category rely on the assumption that exons and introns differ in their nucleotide content or distribution. The main difficulty is then to define a discrimination measure that captures their respective statistical characteristics. At the core of these methods is a function, called *coding* *measure*, that calculates, for several consecutive *windows* of nucleotides or codons, a quantity, called *coding potential*, that can be used to discriminate exons from introns. A window is a subsequence[7] of the sequence to be analyzed, over which the value of the coding measure is calculated. The window, which consists of nucleotides or codons, is moved along the sequence by one unit of length (i.e., a nucleotide or a codon). For each position of the window, the corresponding coding potential is calculated. As each window overlaps the previous one by its length minus one unit, one obtains a detailed analysis of each section of the sequence under scrutiny. Note that the length of the window depends on the desired resolution and the method being employed.

An exhaustive review and a comparison of existing coding measures is provided in (Fickett and Tung, 1992). The purpose of this section, therefore, is to give the essentials of the search-by-content methods via the coding measures that are known to be the most effective for distinguishing exons from introns, i.e.: codon usage, k-tuple frequency, uneven positional base frequencies, and local compositional complexity. These measures take advantage, in one way or another, of the non-random and repetitive character of DNA.

Codon Usage Measure

The codon usage measure (Staden and McLachlan, 1982) was one of the first to be proposed for detecting biologically interesting regions in DNA sequences. It relies on the fact that certain codons are more likely to occur in coding regions than in non-coding ones (these "preferred" codons vary from one organism to another). The codon usage approach consists of constructing a table that contains, for each of the 64 possible codons abc, its occurrence frequency $f(abc)$ in the sequences of a reference set. Assuming that the codons are mutually independent, the probability that a subsequence $\rho = a_1b_1c_1a_2b_2c_2 \ldots a_kb_kc_k$ of some sequence σ to be analyzed contains k (in-frame) codons is estimated by the quantity:

$$C(\rho) = f(a_1b_1c_1)f(a_2b_2c_2)\ldots f(a_kb_kc_k)$$

[7] a subsequence of a sequence S consists of nucleotides that occur *contiguously* in S.

The coding regions of σ are determined by sliding the window ρ along σ, one nucleotide at a time, and by calculating the coding potential $C(\rho)$ for each successive window. If consecutive windows have a low coding potential, then the subsequence of σ covered by these windows is unlikely to be coding. On the contrary, if a window has a high coding potential, then it is likely to represent a coding sequence that starts with an in-frame codon.

There exist a number of variants of the codon usage approach. For example, it is common to use amino acids or diamino acids instead of codons for constructing the frequency table and computing the coding potential $C(\rho)$. These variants often resort to Markov models to capture dependencies between adjacent codons (McCaldon and Argos, 1988; Borodovsky et al., 1994). Another variant of codon usage is the contrast word method (Brendel et al., 1986). The key property in this approach is that words in an unknown language can be detected as the strings that are significantly more frequent than expected on random basis. By comparing the observed and expected frequencies of words (or vocabularies) of different lengths in exons and introns, it can be shown that introns exhibit a specific non-random vocabulary containing a larger number of palindromic words than exons (Beckmann et al., 1986).

k-tuple Frequency Measure

The *k-tuple frequency analysis* relies on the property that the frequencies of nucleotide sequences of length greater than three differ in exons and introns (Claverie and Bougueleret, 1986; Claverie et al., 1990). It consists of considering two reference multisets, one denoted E containing only exons, and the other denoted I containing only introns. Given a sequence σ to be analyzed and the subsequence $\rho = \sigma_{i,i+k-1}$ of σ between and including positions i and $i+k-1$, for $1 \leq i \leq n-k+1$, the respective frequencies $f_E(i)$ and $f_I(i)$ with which $\sigma_{i,i+k-1}$ occurs in the sequences of the sets E and I are computed. For some integer $w > 0$, the following quantity:

$$d(i) = \frac{1}{2w+1} \sum_{j=i-w}^{i+w} \frac{f_E(j)}{f_E(j) + f_I(j)}$$

then represents a measure (smoothed using an averaging window of $2w+1$ nucleotides) of how likely the base at position i is coding. It has been experimentally shown that a value of $k = 6$ and a window of length between 31 and 41 nucleotides yield better exon-intron discrimination.

The *k-tuple frequency analysis* is generalized in (Zhang, 1997) wherein a method based on quadratic discriminant analysis (an extension of the classical linear discriminant analysis) is used to distinguish exons from introns. This method consists of computing the quantity $d(i)$ for two multisets, E and I, that represent different pairs of features such as: exons and introns, exons and *pseudo-exons* (sequences that are flanked by potential splice sites but are not exons), and splice sites and *pseudo-splice sites*. Based on the value of $d(i)$ obtained for each pair, the base at position i is marked as coding or non-coding.

Uneven Positional Base Frequencies Measure

This measure (Staden, 1990) takes advantage of the observed unequal use of the four bases in the three reading frames. This inequality of use is claimed to be more pronounced in exons than in introns. For a given sequence segment ρ containing k codons, let N_{ij} be the total number of occurrences of base i at positions $j+1, j+4, \ldots, j+3k$, for $i \in \{A, C, G, T\}$ and $j \in \{0, 1, 2\}$, and let E_{ij} be the expected number of base i in reading frame j, i.e., assuming an even distribution:

$$E_{ij} = \frac{1}{3}(N_{i0} + N_{i1} + N_{i2}) = \frac{1}{3}(\text{number of times } i \text{ occurs in } \rho)$$

Exons and introns are discriminated by calculating the difference between the expected and the observed positional base usage, i.e.:

$$D = \sum_{i,j} \mid E_{ij} - N_{ij} \mid$$

The value of D is calculated for each position of the window in the sequence to be analyzed, with one codon step. The regions where the value of D is maximal (or greater than a given threshold) are then considered as coding. In (Fickett, 1982), a similar measure is proposed but it is combined with the frequencies of the four bases using a linear weighted sum.

Local Compositional Complexity

This measure is based on the observation that exons are less likely to be random sequences than introns. Therefore, a simple way to locate exons and introns in a given sequence it to compute its local compositional complexity, which in (Konopka and Owens, 1990) is defined as Shannon entropy. The local compositional complexity H of a sequence σ is evaluated in a window of k nucleotides that slides along σ:

$$H = \sum_{i=A,C,G,T} \frac{N_i}{k} \log \frac{N_i}{k}$$

where N_i is the number of occurrences of base i in the window. The value of H is plotted versus the position of the middle of the window. It is observed that for a value of $k = 100$, the resulting plot assumes local maxima in the positions of exons and local minima in the positions of introns, and that the complexity of exons is systematically higher than the complexity of introns.

In (Salomon and Konopka, 1992), the local compositional complexity of a sequence is based on the concept of *repetition vector* instead of Shannon entropy. Given a window ρ of length k, the repetition vector of ρ is defined as: $\tilde{k} = (k_1, k_2, k_3, k_4)$. It represents the number of occurrences in ρ of each of the four nucleotides, ordered in decreasing order, i.e., $k_1 \geq k_2 \geq k_3 \geq k_4$. The local compositional complexity of ρ is then calculated as follows:

$$K = \frac{1}{k} \log \frac{k!}{k_1! k_2! k_3! k_4!}$$

Experimentation with various lengths indicates that when using a window of length $k = 8$, the values of K are generally higher in exons than in introns (Snyder and Stormo, 1995).

There exists a mosaic of mathematical theories for analyzing sequences based on their complexity, patterns (e.g., repeats), and periodicity. These are reviewed in (Wootton, 1997).

Search-by-content measures are often criticized because they are derived from only a fraction of known sequences and are therefore not representative of the genome as a whole. Although they can be used to locate the exons and introns in a given sequence, in general they fail to detect their edges precisely. The prediction accuracy of search-by-content methods is significantly improved when two or more coding measures are combined, and when consensus or weight matrices are used to locate the putative splice sites. This combination of methods is at the core of a number of programs that predict the complete exon-intron structure of genes.

2.2.4 Integrated Gene Identification Systems

The approaches described in the previous sections consider either short nucleotide sequences or partial gene fragments to predict splice sites, exons or introns. Because search-by-signal and search-by-content methods recognize isolated aspects of genes without taking into account gene structure, they can be easily applied to sequences containing incomplete genes. However, these methods are not adequate to predict gene structure because they do not take into account the interdependencies (such as that expressed by the triplicity constraint) that exist among the various components of genes. In this section, a general framework for predicting the entire exon-intron structure of a gene is described. The corresponding basic algorithm can be sketched as follows:

1. The potential exons and introns are determined.

2. All possible parses, i.e., combinations of exons and introns representing a gene model, are constructed; each of the parses is assigned a score.

3. The optimal parse, i.e., the parse with the highest score, is selected as the most probable gene model.

The programs that implement the above strategy are designated as *integrated gene identification systems*. These programs, reviewed in (Snyder and Stormo, 1997), mainly differ in the way the putative exons and introns are determined. As no single measure can *per se* accurately discriminate exons from introns, a multitude of tests are usually performed to quantify the various properties characteristic of these sequences. For example, by combining the methods discussed in Sections 2.2.2 and 2.2.3, one can often obtain a good estimate of where the putative exons and introns are in the sequence being analyzed. The gene structure prediction problem can then be formulated according to the methodology in (Stormo and Haussler, 1994).

Given a subsequence $\rho = \sigma_i \sigma_{i+1} \ldots \sigma_j$ of a sequence σ of length n, one can define a score $T_\mu(i,j)$ for each statistic μ in a collection "exon statistics". This score reflects how likely the sequence fragment ρ is an exon with respect to statistic μ. Similarly, one can define a score $T_\kappa(i,j)$ for each statistic κ in a collection "intron statistics". The respective scores of the sequence fragment ρ for the exon statistics and the intron statistics can be combined to obtain the composite exon and intron statistics. Those are given by:

$$L_E(i,j) = \textstyle\sum_\mu w_\mu T_\mu(i,j)$$

$$L_I(i,j) = \textstyle\sum_\kappa w_\kappa T_\kappa(i,j)$$

where w_μ and w_κ are adjustable weights that are parameters of the composite exon and intron statistics.

A gene model ϕ for the sequence σ is represented by a series of positions: $i_0, i_1, \ldots, i_{2m+1}$, such that $i_0 = 1$, $i_{2m+1} = n$, and, for $1 < k < n$, the value of i_k represents the position of the k^{th} potential splice site (donor or acceptor). The score of the gene model ϕ is then defined as:

$$\Theta(\phi) = L_E(1, i_1) + \sum_{l=1}^{m} (L_I(i_{2l-1}, i_{2l}) + L_E(i_{2l}, i_{2l+1}))$$

As noted above, determining the most probable gene model for σ is then reduced to the problem of constructing the optimal parse Φ of σ, i.e., the alternations of exons and introns with the highest score:

$$\Theta(\Phi) = max_\phi \Theta(\phi)$$

In early methods (Gelfand, 1990; Guigo et al., 1992; Fields and Soderlund, 1990; Fields et al., 1991; Uberbacher et al., 1993; Milanesi et al., 1993), the most probable gene model Φ was determined by performing an exhaustive search over the set of all possible gene structures. Because the size of the search space generated by these methods grew exponentially in the number of putative splice sites, only a fraction of the set of all putative exons and introns could be examined, and only the sequences of reasonable length could be considered for analysis. These "brute-force" methods were soon followed by more sophisticated algorithms using dynamic programming (Snyder and Stormo, 1993; Xu et al., 1994). In particular, there exists a general dynamic programming algorithm for calculating Φ when the score function Θ is linear in the adjustable weight parameters w_μ and w_κ (Stormo and Haussler, 1994), or monotonically increasing (Gelfand and Roytberg, 1993; Gelfand et al., 1996b).

For each exon statistic μ and intron statistic κ, the weights w_μ and w_κ are determined from a training set of sequences whose gene structure is known. The main difficulty is to choose a set of weights such that, for every sequence σ in the training set, the optimal parse of σ represents its exon-intron structure. There have been two tentative approaches to solve this problem, resulting in two different algorithms. The first algorithm (Snyder and Stormo, 1993; Snyder and Stormo, 1995) finds a weighting such that most, but not all, sequences in the training set are correctly parsed. In contrast, the second algorithm (Stormo and Haussler, 1994) finds a weighting such that most, but not all, optimal parses correspond to actual gene structure. In the first approach, a quadratic time dynamic programming algorithm is combined with a neural network that is trained to maximize the number of correct parses of the sequences in the training set. The training procedure can be succinctly described as follows:

1. For every exon statistic μ and intron statistic κ, the weights w_μ and w_κ are initialized to random values.

2. For $1 \leq i < j \leq n$, the scores $T_\mu(i,j)$ and $T_\kappa(i,j)$ are computed for every sequence in the training set $\mathcal{T}$.

3. The composite scores $L_E(i,j)$ and $L_I(i,j)$ are computed for every sequence in $\mathcal{T}$.

4. Dynamic programming is used to determine the most probable parse of each sequence in $\mathcal{T}$.

5. If the desired proportion of correct parses is reached, then the procedure ends. Otherwise, the score differences between the optimal parses and the correct parses are used by a neural network to find new weights that give higher scores to the correct parses.

6. Steps 3-4-5 are repeated.

In the second approach, the weight learning phase by neural network is replaced with a gradient descent method which is used to maximize the probabilities of the correct parses (i.e., the optimal parses may not be correct but they have as high a probability as possible). In (Rogozin et al., 1996; Gelfand et al., 1996a), the most probable gene model is obtained by searching the gene databases for the sequence of amino acids that best fits the concatenated exons. This strategy relies on the assumption that a newly discovered gene is likely to have a known homolog, as gene databases, which already contain an enormous number of genes, continue growing rapidly.

Integrated gene-finding programs are known to perform much better in predicting gene structures than stand-alone coding measures. It is estimated that the increase in accuracy obtained by incorporating overall gene structure information is approximately ten-fold (Fickett, 1996b). In our view, the explicit formulation of the various statistics involved in the description of gene structure facilitates the task of the learning procedure that is often used to optimize parsing accuracy. Therefore, it is not surprising that the methods described in this section also yield better results than those relying only on neural nets (Brunak et al., 1991), in which all parameters or statistics have to be "learnt". However, integrated gene prediction programs have some limitations as well (Fickett, 1996a): they are very organism specific; they assume that the sequence contains a single entire gene; they are sensitive to sequencing errors and they are not designed for particular situations such as alternative splicing and overlapping genes (see Section 2.3). An alternative and more general approach to describing all possible parses of a sequence (or other structure) is through the use of formal grammars.

2.2.5 Linguistics Methods

Because DNA sequences are not random but organized in distinctive parts (i.e., promoters, start codons, introns, exons, stop codons, and so on), and the living world DNA corresponds only to a small subset of all possible sequences over the bases A, C, G, T, it is tempting to think of nucleotide sequences as sentences of a language that can be described by some grammar. The mission of the geneticist or linguist is then to understand what the sequence is "trying to say" (Gribskov, 1992).

The work in (Brendel and Busse, 1984) seems to be the first attempt to use the theory of formal languages to describe the structural properties of genes: in this work, a finite transducer is used to simulate the translation of mRNA (i.e., with no introns) into amino acid sequence. In (Kudo et al., 1987), a generalization procedure is proposed to infer, from a training set of splice site sequences, a stochastic, deterministic regular grammar that describes the patterns at the $5'$-splice sites of mammalian genes. A deterministic finite-state machine is also used in (Smith, 1988) but to detect *restriction sites* (i.e., a short specific nucleotide sequence at which a particular protein, called *restriction enzyme*, cuts the DNA). In (Collado-Vides, 1989; Collado-Vides, 1991b; Collado-Vides, 1992; Collado-Vides, 1993), transformational grammars whose terminal symbols denote molecular categories (i.e., promoter, structural gene, and so forth) are used to formalize the regulation of gene expression.

In (Searls, 1992), different grammars in the Chomsky hierarchy are proposed to attempt to describe the various linguistic patterns that occur in DNA (or RNA) such as repeats and palindromes. In particular, a context-sensitive grammar is used to provide a (simplified) description of the structure of protein-coding genes (see Figure 2.9). In (Searls, 1993), it is shown that the language of DNA is beyond context-free but within the class of indexed languages. A formalism called *string variable grammar*, or *SVG*, that is less general but easier to handle than indexed grammars is then derived (Searls, 1995b). A string variable grammar allows variables denoting substrings of unbounded length to appear on the right-hand sides of otherwise context-free rules. For example, repeats can be represented by the simple

$$gene \rightarrow upstream\ transcript\ downstream$$

$$transcript \rightarrow 5'\_untranslated\_region\ start\_codon\ coding\_region\ 3'\_untranslated\_region$$

$$coding\_region \rightarrow codon\ coding\_region \mid stop\_codon \mid splice\ coding\_region$$

$$\begin{aligned} codon \rightarrow\ & lys \mid asn \mid ile \mid thr \mid met \mid ser \mid gln \mid his \mid arg \mid pro \mid \\ & asp \mid glu \mid ala \mid gly \mid val \mid tyr \mid trp \mid cys \mid phe \mid leu \end{aligned}$$

$$start\_codon \rightarrow met \qquad stop\_codon \rightarrow \mathbf{taa} \mid \mathbf{tag} \mid \mathbf{tga}$$

$$\begin{array}{lll} lys \rightarrow \mathbf{aa}\ purine & asn \rightarrow \mathbf{aa}\ pyrimidine & ile \rightarrow \mathbf{at}\ pyrimidine \mid \mathbf{ata} \\ thr \rightarrow \mathbf{ac}\ base & met \rightarrow \mathbf{atg} & ser \rightarrow \mathbf{ag}\ pyrimidine \mid \mathbf{tc}\ base \\ gln \rightarrow \mathbf{ca}\ purine & his \rightarrow \mathbf{ca}\ pyrimidine & arg \rightarrow \mathbf{cg}\ base \mid \mathbf{ag}\ purine \\ pro \rightarrow \mathbf{cc}\ base & asp \rightarrow \mathbf{ga}\ pyrimidine & glu \rightarrow \mathbf{ga}\ purine \\ ala \rightarrow \mathbf{gc}\ base & gly \rightarrow \mathbf{gg}\ base & val \rightarrow \mathbf{gt}\ base \\ tyr \rightarrow \mathbf{ta}\ pyrimidine & trp \rightarrow \mathbf{tgg} & cys \rightarrow \mathbf{tg}\ pyrimidine \\ phe \rightarrow \mathbf{tt}\ pyrimidine & leu \rightarrow \mathbf{tt}\ purine \mid \mathbf{ct}\ base & \end{array}$$

$$pyrimidine \rightarrow \mathbf{c} \mid \mathbf{t} \qquad purine \rightarrow \mathbf{a} \mid \mathbf{g} \qquad base \rightarrow pyrimidine \mid purine$$

$$splice \rightarrow intron \qquad intron \rightarrow \mathbf{gt}\ intron\_body\ \mathbf{ag}$$

$$\begin{array}{llll} splice\ \mathbf{a} \rightarrow \mathbf{a}\ intron & splice\ \mathbf{c} \rightarrow \mathbf{c}\ intron & splice\ \mathbf{g} \rightarrow \mathbf{g}\ intron & splice\ \mathbf{t} \rightarrow \mathbf{t}\ intron \\ \mathbf{a}\ splice \rightarrow intron\ \mathbf{a} & \mathbf{c}\ splice \rightarrow intron\ \mathbf{c} & \mathbf{g}\ splice \rightarrow intron\ \mathbf{g} & \mathbf{t}\ splice \rightarrow intron\ \mathbf{t} \end{array}$$

Figure 2.9: Partial context-sensitive grammar describing the structure of genes.

SVG rule: "$S \rightarrow X, X.$", where X denotes a string variable. The combination of string variable grammars with definite clause grammars constitutes the core of an elaborate gene parser that features a built-in cost mechanism to account for mismatches (or other characteristics) in the string being parsed, and an efficient procedure to handle gaps (Searls and Dong, 1993; Searls, 1994).

In this gene parser, the SVGs consist of two types of rules: the *leaf rules* and the *node rules*. The leaf rules describe partial DNA fragments (such as exons or introns) over which a cost associated with a desired property (such as coding potential) is calculated. This cost can be specified in many different ways; for example, it can be a weight matrix or k-tuple frequency score, or any other user-defined function. The node rules are used to combine the sequences, and the scores of the sequences, denoted by the leaf rules or other node rules, the cost of a sequence being a weighted combination of the costs of the sequences it is derived from. The sequence or parse with the minimum top-level cost is then selected as the most probable gene model. Like the other integrated gene prediction systems, a training procedure is used to determine the weights in the score combination rules that optimize parsing accuracy.

Note that the string variable grammars that describe gene structure are highly non-deterministic. However, the exhaustive search of all the possible gene structures is avoided by the incorporation of chart parsing techniques and control directives that constrain both the span of a syntactic subtree of the sequence to analyze and the time spent in attempting to derive a given non-terminal.

Other existing linguistics-based approaches resort to stochastic grammars to deal with the formidable combinatorics involved in the construction of gene models. In (Krogh et al., 1994), a *Hidden Markov Model (HMM)* is designed to find (intron-free) genes in *E. coli* DNA. Essentially, an HMM is a special kind of transducer in which every transition and every output symbol is assigned a probability (Rabiner, 1989). Given an HMM, a procedure known as the Viterbi algorithm can be used to determine the most probable parse of an input sequence. In (Krogh et al., 1994), the probabilities of the HMM are estimated automatically by a learning procedure known as the *forward-backward* algorithm, which is a special case of the more general Expectation Maximization method. In (Kulp et al., 1996), separate HMMs are built and trained separately to detect specific regions of human DNA, such as exons and introns. These HMMs are then combined together to form a unique HMM that describes complete biological structures such as genes. Finally, this HMM is further trained to increase prediction accuracy.

On one hand, the main advantage of using grammars is that they can describe the genetic and evolutionary operations (such as mutations) that are essential for understanding the structure of DNA (Searls, 1993; Yokomori and Kobayashi, 1995). The use of grammars can be justified by the following metaphor (Gribskov, 1992): just as languages change over time and location, proteins arising from the same gene may also diverge in their functions. On the other hand, the static nature of grammars is in conflict with the dynamic nature of molecules, and their lack of specificity is in contradiction with gene expression which is tissue-specific. A grammar-based formalism has been proposed in (Searls, 1995a; Bentolita, 1996) as a preliminary effort to surmount these shortcomings: it takes into account the fact that DNA, RNA and proteins interact with each other.

2.2.6 Assessment of Prediction Accuracy

There exist a number of ways to assess the quality of the prediction made by a program on a given sequence. For example, one can:

- Compare the predicted protein with the actual one: the similarity between the chain of amino-acids encoded by the predicted gene and that of the actual gene is measured by an alignment algorithm such as the one proposed in (Pearson and Lipman, 1988).

- Count the number of correctly predicted exons (or introns): the number of accurately predicted exons (or introns) is expressed as a fraction of the number of actual exons (or introns).

- Count the number of correctly predicted splice sites: the number of accurately predicted donor and acceptor splice sites is expressed, respectively, as fractions of the numbers of actual donor and

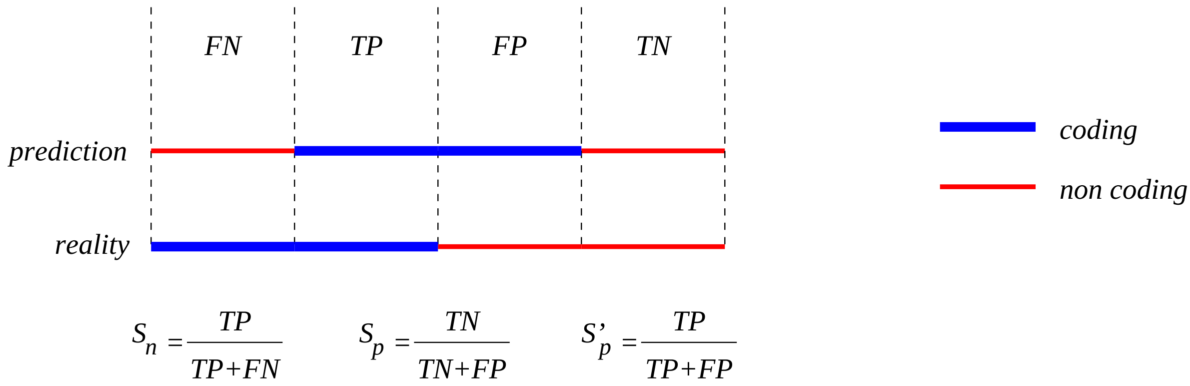


Figure 2.10: Sensitivity (S_n) and specificity (S_p and S'_p) defined in terms of true positives (TP), true negatives (TN), false negatives (FN) and false positives (FP).

acceptor splice sites.

- Count the overlaps between predicted exons (or introns) and actual exons (introns): the number of bases predicted to be coding (or non-coding) is compared with the true number of coding (or non-coding) bases. Often designers of programs also indicate the overlaps between exons that have at least one correctly-detected end and the actual exons.

These accuracy measures more or less reflect the degree of concordance between the predicted and actual exon locations. The four following quantities are usually calculated for measuring the overlaps (see Figure 2.10):

- *True positives* (TP): this is the number of coding nucleotides that have been correctly predicted as coding.

- *True negatives* (TN): this is the number of non-coding nucleotides that have been correctly predicted as non-coding.

- *False negatives* (FN): this is the number of coding nucleotides that have been predicted as non-coding.

- *False positives* (FP): this is the number of non-coding nucleotides that have been predicted as coding.

The proportions of nucleotides that have been correctly predicted as belonging to an exon and an intron are, respectively:

$$S_n = \frac{TP}{TP + FN} \qquad S_p = \frac{TN}{TN + FP}$$

These two quantities are respectively termed *sensitivity* and *specificity*. As introns are in general much longer than exons, the number of true negatives TN tends to be much larger than the number of false positives FP. This results in values of specificity S_p that are systematically close to 100% and therefore non informative. For this reason, the specificity is usually defined as the proportion of predicted coding nucleotides that are actually coding, i.e.:

$$S'_p = \frac{TP}{TP + FP}$$

Note that the quantity S'_p can be thought of as a conditional probability: it is the probability that a nucleotide belongs to an exon, given that it has been predicted as coding. Similarly, S_n can be interpreted as the probability that a base is predicted coding, given that it belongs to an exon. Although a value of 100% for both sensitivity and specificity correspond to a perfect prediction, it is in general difficult to interpret S_n and S'_p in terms of prediction accuracy. For example, a program that predicts most nucleotides to be coding is likely to have a high sensitivity but a low specificity whereas a program that predicts most nucleotides to be non-coding is likely to have a high specificity and low sensitivity.

A better, unique measure that has been widely used to capture the global accuracy of a prediction is the *correlation coefficient*. It is given by the formula (assuming that the quantity in the denominator is not null):

$$CC = \frac{(TP \times TN) - (FN \times FP)}{\sqrt{(TP + FN) \times (TN + FP) \times (TP + FP) \times (TN + FN)}}$$

The correlation coefficient ranges from -1 to 1: a value of 1 indicates a perfect prediction whereas a value of 0 corresponds to a random prediction. The correlation coefficient does not only capture the sensitivity and specificity in predicting coding nucleotides (i.e., S_n and S'_p) but also the sensitivity and specificity in predicting non-coding nucleotides. However, it is not defined when any of the quantities: $TP + FN$, $TN + FP$, $TP + FP$ or $TN + FN$ is zero, i.e., when there is no actual gene or no predicted gene in the sequence to be analyzed. An alternative to CC is the *average conditional probability* measure (Burset and Guigo, 1996), which is defined as the average of the quantities: S_n, S_p, $S'_n = TN/(TN + FN)$ and S'_p. The average is actually taken over those quantities that are defined (always at least two). If all quantities are defined, the average conditional probability is:

$$ACP = \frac{1}{4}(\frac{TP}{TP + FN} + \frac{TN}{TN + FP} + \frac{TN}{TN + FN} + \frac{TP}{TP + FP})$$

In (Burset and Guigo, 1996), it is observed that the scalar:

$$AC = 2 \times ACP - 1$$

termed *approximate correlation* is in practice closed to CC whenever CC is defined.

It should be noted that the notions of sensitivity and specificity can be extended to exons (and introns) in the following way: the sensitivity is the proportion of actual exons that are correctly predicted and the specificity is the proportion of predicted exons that are true exons. One can also define the *missing exons*, i.e., the proportion of true exons that do not overlap with any predicted exons, and the *wrong exons*, i.e., the proportion of predicted exons that do not overlap with any actual exons.

The definitions provided so far to measure the accuracy of a program on a given sequence can be extended to measure the accuracy on a set of sequences. This evaluation can be performed in two different ways: either calculate the quantities TP, TN, FN, and FP for each sequence, and average over all sequences for which these quantities are defined (average by sequence); or sum these quantities as if all the sequences were concatenated, and calculate the various assessment statistics from these overall numbers (average by base).

Comparing the predictive accuracy of a method with another is an important step to understand the strength and weakness of each program. Two reference databases, one containing 353 human sequences[8] and the other 570 vertebrate sequences[9], among which 176 are human sequences, have been developed for testing programs. Using the latter database, a detailed discussion of the predictive accuracy of various gene structure prediction programs was carried out in (Burset and Guigo, 1996). It was observed that the programs that perform similarity searches with the sequences in the protein databases yield better results than the other programs. The two best programs (Guigo et al., 1992; Snyder and Stormo, 1995) based on protein sequence homology have both the same approximate correlation, i.e., $AC = 0.86$, when averaged by sequence over the 176 human sequences, whereas the best program based on sequence information only (Solovyev et al., 1994) has $AC = 0.80$ as approximate correlation.

Finally, it is worthwhile to note that the performance of the existing programs is in general sensitive to the ratio of $G+C$ occurring in the sequences (Guigo and Fickett, 1995; Burset and Guigo, 1996): they perform better in $G+C$ rich sequences. As a general rule, prediction accuracy decreases with both gene length and intron quantity. These observations are compatible with those made in (Duret et al., 1995) and (Burge, 1997), wherein it is shown that (a) intron length tends to decrease — and exon length to increase — as a function of $G+C$ content, and (b) long genes appear to be poorer in $G+C$ nucleotides than short sequences.

An issue closely related to performance assessment is the quality of the data. A sequence submitted to a database may contain errors. The rate of error is estimated to range between 0.1% and 1% (Marshall, 1995). Although algorithms have been proposed to correct sequencing errors in DNA

[8] `http://www-hgc.lbl.gov/inf/genesets.html`
[9] `http://www.imim.es/GeneIdentification/Evaluation/Index.html`

sequences (States and Botstein, 1991; Posfai and Roberts, 1992; Fichant and Quentin, 1995; Xu et al., 1995), only a few gene structure prediction programs perform well in the presence of errors. In addition to human or experimental errors, prediction accuracy is complicated by the constantly growing number of exceptions.

2.3 Epilogue: Biology is Far More Complex

In this chapter, we have described a number of biological/logical rules and statistical/empirical properties. These can be summarized as follows:

- Biological/logical rules:
  - $GT - AG$ rule: the vast majority of introns begins with the sequence GT and ends with the sequence AG.
  - Triplicity constraint: the length of the appended exons is divisible by three.
  - Open Reading Frame constraint: no in-frame stop codons occurs within an exon or in the boundary of two appended exons.
- Statistical/empirical properties:
  - Consensus, position weight matrices: the sequences at the splice sites are more or less conserved.
  - Codon usage: the 64 codons are not used evenly.
  - k-tuple frequency: short nucleotide sequences have different frequencies in exons and introns.
  - Uneven positional base frequencies: nucleotides are not used uniformly in the three reading frames.
  - Local compositional complexity: the local nucleotide content is more uniform in exons than in introns.

Although the $GT - AG$ rule was also derived from empirical observation, it is more widely accepted than the other above-mentioned properties.

Many other statistical laws have been inferred from the analysis of DNA sequences: they are reviewed in (Gelfand, 1995). As the process of intron splicing is better understood, the need for empirical rules will certainly decrease. In designing a gene structure prediction program, one should keep in mind that the deciphering of DNA is hampered by a number of complications:

- Some splice sites do not obey the $GT - AG$ rule: an important exception is the subclass of introns with a GC dinucleotide at their $5'$ end.
- Splice sites are not always recognized in isolation: a mutation that inactivates a splice site can sometimes be compensated by the activation of a "cryptic" splice site. If not, the matching splice site may be also inactivated and an exon may be excluded from being translated (this process is known as *exon skipping*).
- The pattern of intron removal is sometimes tissue specific: certain genes code for several different proteins as the result of alternative splicing.
- Genes can be overlapping: the same base sequence can be read in more than one reading frame, and still encode a gene.
- some genes are located in the introns of other genes: these introns are difficult to detect because they look like exons.
- Some genes are "inside-out": the role between exon and intron is reversed, i.e., the introns display characteristics that are typical of exons and they encode proteins.

These molecular "vagaries" constitute nontrivial challenges for all existing gene structure prediction programs. The approach proposed in the next chapter to deal with this staggering complexity is to use finite-state machines to express *all* possible ambiguous interpretations of a sequence.

Chapter 3

Finite-State Automata Representation of Gene Models

In the previous chapter, a number of methods have been delineated for predicting gene structure. In particular, it has been seen that the approaches using formal languages combine in a very simple and general manner a number of structural and/or statistical properties inherent to genes. The aim of these methods is not only to improve prediction accuracy but also create a realistic model of the underlying biochemistry. However, all existing methods use statistics or serendipitous learning procedures to resolve parsing ambiguities. As a consequence, the knowledge needed to understand the biochemistry involved is partially hidden in a multitude of parameters or weights. Because these parameters or weights can only be optimized for a finite training set of sample sequences, it is often the case that the actual alternation of exons and introns is missed by the gene structure prediction program.

The aim of the approach described in this chapter and further developed in the next is to reconcile the two poles of interest that are prediction and explication. In what follows, some of the rules that govern the synthesis of proteins are compiled into a non deterministic finite-state machine which is used to generate *all* parses of a given input sequence. This strategy guarantees that no potential gene models are missed.

The results obtained with simple Yeast saccharomyces cerevisiae genes[1] and more complex human genes indicate that, in most sequences, the non-coding regions correspond to the subsequences that have a unique interpretation. This key observation is at the source of the development of fast efficient algorithms. In the following, a sequential algorithm and its parallel counterpart are described. The soundness of the proposed strategy and its relationship with other linguistic approaches are also discussed.

3.1 Overview of the NDFSA-based Approach

The approach described in this chapter is based on grammars but it differs significantly from previous linguistics-based work (Brendel and Busse, 1984; Kudo et al., 1987; Smith, 1988; Searls, 1994) in the sense that Non-Deterministic Finite-State Automata (NDFSA, for short) are used to generate all possible parses of highly ambiguous gene sequences. Formal language specialists do not seem to have been interested in describing parsing ambiguities using NDFSA. Perhaps the rationale behind it has to do with a well known theorem that guarantees the construction of an equivalent deterministic finite state automata from any NDFSA (Hopcroft and Ullman, 1979). However, that construction defeats the purpose of specifying NDFSA: not only the ambiguities are lost, but the time required for constructing the deterministic finite-state automata may be prohibitive since it is exponential.

In this chapter, NDFSA are used to describe a number of simple but very selective constraints that must be satisfied by protein-coding genes. It is worth noting that a string accepted by an NDFSA may have an exponential number of parses, each of which corresponds to a sequence of transitions between

[1]Saccharomyces cerevisiae is commonly used as baker's yeast and for some types of fermentation. In particular, it is utilized to ferment the sugars of rice, wheat, barley, and corn to produce alcoholic beverages, and in the baking industry to raise dough. Saccharomyces cerevisiae exhibits many of the fundamental properties of cells in other eukaryotes such as human cells. Moreover, yeast cells are easy to grow and handle biochemically, and they are amenable to sophisticated genetic manipulations. Unlike other mammals, yeast sequences contain only one or two introns and the consensus at the splice sites and the branch site are highly conserved.

the initial state and the final state of that NDFSA[2]. However, these parses can be succinctly represented by a directed acyclic graph (DAG) that can be constructed by an efficient sequential or parallel algorithm. Intuitively, one can infer that given: (1) a labeled transition graph representing an NDFSA, and (2) a finite string accepted by that automaton, it is possible to generate a DAG whose paths from the unique source to the unique sink represent all the possible parses of that string.

An informal argument to justify the existence of the DAG is as follows: consider the given string and exhaustively list all transition sequences of the NDFSA starting from the initial state and ending on the final state. These transition sequences can be condensed into a DAG by factorizing common prefixes and suffixes. Obviously, the algorithm that actually constructs the DAG avoids that exhaustive enumeration of transition sequences.

It is worth keeping in mind that in computer science there exist linear algorithms that can generate a compact representation for an exponential number of objects. One such example is the elimination of epsilon rules in context-free grammars (Harrison, 1978). The approach described below is of similar nature. A DAG is constructed when given an NDFSA, and a string accepted by that NDFSA. The problem is to select in this DAG one or more paths having a semantic significance chosen by the user. The DAG has a number of edges corresponding to the transitions of the NDFSA upon reading the symbols in the string being parsed. One can view the construction of the DAG as an unfolding of the NDFSA when provided the given string. The DAG contains multiple paths, each of which corresponds to an ambiguity (due to non-determinism) in parsing the given string. A parse is therefore represented by a sequence of edges in the DAG. This sequence of edges corresponds to one of the transition sequences of the NDFSA that allows acceptance of the string.

Every node in the DAG can be uniquely represented by a pair (k, i) where k refers to the k^{th} symbol in the input sequence and i refers to some state q_i of the NDFSA. The following strategy is employed to select the semantically significant paths in the DAG: first, determine, for each edge

[2]The NDFSA considered in this chapter have a unique initial state and a unique final state.

$\mu = (k-1, i) \rightarrow (k, j)$ from node $(k-1, i)$ to node (k, j), the probability $P_{ij}(k)$ that edge μ is passed through by a path from the source node to the sink node (these nodes respectively correspond to the initial state and the final state in the NDFSA). If $P_{ij}(k) = 0$, then state q_i is not needed to reach state q_j upon reading the k^{th} input symbol. Similarly, if $P_{ij}(k) = 1$, then every transition that leads to state q_j after reading the k^{th} input symbol originates from state q_i. However, if the value of $P_{ij}(k)$ lies between 0 and 1, then an additional criterion must be applied to decide whether or not state q_i is needed to make a transition to state q_j upon reading the k^{th} input symbol. It will be seen that the DAG and the quantities $P_{ij}(k)$'s can be determined in $O(ne)$ time in sequential, and $O(\frac{ne}{p})$ time on a PRAM with $O(p)$ processors, where n is the length of the input string and e the number of transitions of the NDFSA.

For the purpose of specifying a criterion, every transition in the NDFSA, and therefore edge in the DAG, is assigned a type chosen by the user to specify a semantically important characteristic that he or she wishes to detect, e.g., coding or non-coding. The definition of $P_{ij}(k)$ is then extended to the set of edges sharing the same type: for a given type t, the ratio $r_t(k)$ is defined as the sum of the probabilities $P_{ij}(k)$'s over all transitions from state i to state j of type t upon reading the k^{th} input symbol. Based on the value of $r_t(k)$, the criterion is used to determine if a sequence of adjacent symbols — a subsequence of the input sequence — has the property associated with type t. For example, if all symbols of a subsequence have a value of $r_t(k)$ equal to one, then the subsequence exhibits the property associated with t.

In the following, we first provide a general framework for analyzing DNA sequences. Then, we define an NDFSA that describes baker's yeast genes, and we show that by plotting $r_{coding}(k)$ as a function of k, for the type *coding* associated with the property "nucleotide is coding", we can detect where the exons and introns are in these genes. Finally, we provide a general strategy for parsing human genes and we evaluate the proposed approach on the 176 human genes of the test set proposed in (Burset and Guigo, 1996).

3.2 The Approach, Explained Intuitively

We believe that gene structure prediction should be performed in two steps: the first should optimize sensitivity, i.e., locate all regions that are likely to be coding, whereas the second should optimize specificity, i.e., detect the false positives produced during the first step. Since most gene structure prediction programs perform reasonably well in maximizing specificity (Burset and Guigo, 1996), we focus on the first step.

The following strategy is employed to maximize sensitivity: a sequence fragment is marked as coding even if it has a very small probability to be coding. In this way, no important information that could contribute to the final protein product is discarded. Therefore, the proposed gene structure prediction program predicts exons and introns by distinguishing the regions that *may* be coding from the regions that *cannot* be coding. In other words, the approach consists of defining a set of rules that unequivocally identifies which fragments of a given sequence are non-coding.

This set of rules is represented by non-deterministic finite-state automata. These are used to calculate, for each nucleotide in a given input sequence, its probability of being generated by a transition of type coding. An initial NDFSA is designed to mimic the possible alternations of exons and introns in baker's yeast genes. By plotting the curves expressing the probabilities that the nucleotides in the analyzed sequences are coding, it is observed that every exon has a non-null probability to be coding whereas every intron has a null probability to be coding. This key property leads to the following conjecture: "If a nucleotide can be coding, it will." This simple assertion, to be clarified below, is the most important contribution of this thesis. It will be used in the next section to predict coding and non-coding regions in human genes, and to interpret the result of these predictions in terms of biochemical processes.

3.2.1 Theoretical Motivations

Let $\mathcal{M} = (Q, \Sigma, T, q_0, q_f)$ be a finite-state automaton:

- $Q = \{q_0, \ldots, q_f\}$ is a finite set of states ($f \geq 0$).
- q_0 is the unique initial state of M.
- q_f is the unique final state of M.
- Σ is a finite alphabet: in the case of DNA, $\Sigma = \{a, c, g, t\}$.
- T is the transition relation of $\mathcal{M}$, i.e., a subset of $Q \times \Sigma \times Q$.

The sequence of transitions made by $\mathcal{M}$ upon reading the symbols of an input sequence $\sigma \in \Sigma^n$ of length n accepted by $\mathcal{M}$ is called a *parse* of σ. If $\mathcal{M}$ is non-deterministic then the sequence σ may have one or more parses. Let N be the total number of parses of σ, and $N_{ij}(k)$ the number of parses in which a transition of $\mathcal{M}$ is made from state $q_i \in Q$ to state $q_j \in Q$ upon reading the k^{th} symbol of σ, denoted σ_k. Assuming that $N > 0$, the probability that σ_k causes $\mathcal{M}$ to make a transition from state q_i to state q_j is:

$$P_{ij}(k) = \frac{N_{ij}(k)}{N} \tag{3.1}$$

Define $\mathcal{T}$ as a set containing a finite number of elements called *types*, and τ as a function that assigns a type denoted $\tau(i, j)$ to every pair of states $(q_i, q_j) \in Q \times Q$. Intuitively, a type corresponds to a semantically important characteristic that one wishes to detect in the input sequence, for example, coding regions in genes. By extension, the type of a transition $(q_i, x, q_j) \in T$ is defined as the type of (q_i, q_j). Therefore, the type of the transition (q_i, x, q_j) does not depend on the input symbol x. For a given type $t \in \mathcal{T}$, the probability that the k^{th} symbol of σ causes $\mathcal{M}$ to make a transition of type t is:

$$r_t(k) = \sum_{\tau(i,j)=t} P_{ij}(k) \tag{3.2}$$

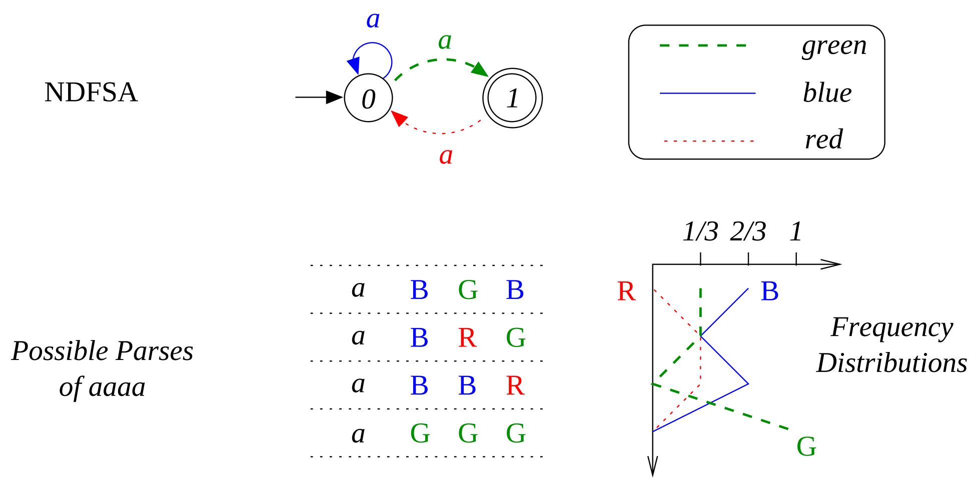


Figure 3.1: A simple NDFSA generating three possible parses on input string $aaaa$.

If $r_t(k) = 0$, then σ_k is not generated by a transition of type t. Similarly, if $r_t(k) = 1$, then σ_k is necessarily generated by a transition of type t, in which case the k^{th} symbol of σ is said to be of type t. If the value of $r_t(k)$ lies between 0 and 1, then a heuristic or criterion must be applied to decide whether or not σ_k is of type t. A criterion ξ is a binary relation over $\mathcal{T} \times [0,1]$: the k^{th} symbol of σ is said to be of type t if and only if $(t, r_t(k)) \in \xi$. The criterion is used to determine, based on the value of r_t, if a sequence of adjacent symbols has the property associated with type t. In other words, if the criterion is satisfied for successive values of k in a subsequence of the input sequence, then this subsequence exhibits the property associated with t.

Although the purpose of the criterion is to select a particular parse of the input string, it may be the case that the parse obtained after its application is not generated by the NDFSA. This is because the criterion is applied separately to every symbol of the input string rather than the entire sequence. When possible, it is generally desirable to select a criterion that guarantees that the final parse is actually generated by the NDFSA.

As a toy example of NDFSA and criterion, consider the automaton in Figure 3.1. This automaton has two states: 0 (initial) and 1 (final), and three transitions which have been colored to represent all sequences of a's satisfying the following rules:

1. Every red a is preceded by a green a.

2. No blue a is preceded by a green a.

3. No two consecutive green a can occur.

4. The sequence ends with a green a.

In this particular example, each color represents a different type: the property associated with a transition of a given type or color is "a is of this color". Given the above set of rules, it can be seen that there are three possible ways of coloring the string $aaaa$. In Figure 3.1, the probability that each a is of a given color is represented by three curves which correspond to the frequency distributions of each type, i.e., the functions r_{blue}, r_{green} and r_{red}, respectively. A criterion can then be applied to select a particular combination of colors. For example, if one specifies, in addition to the four above rules, that an a is red if and only if its probability of so being is not null, i.e.:

$$(red, r) \in \xi \text{ if and only if } r > 0$$

then the second and third a in the string $aaaa$ are red. However, note that this parse is not generated by the NDFSA in Figure 3.1. Furthermore, if one also specifies that an a is blue if and only if its probability of so being is not null, then the second and third symbols have two contradictory types, i.e., they are both *blue* and *red*. Therefore, when defining criteria, one should be careful not to introduce ambiguities and make sure that the solutions selected by the criterion are generated by the NDFSA.

In the context of gene analysis, it may be the case that not every gene sequence is recognized by the NDFSA $\mathcal{M}$. For example, this may happen if $\mathcal{M}$ describes syntactic features that are not present in every gene. In such a situation, it is useful to have a list of alternative NDFSA: $\mathcal{M}_1, \ldots, \mathcal{M}_g$, such that the NDFSA in this list provide different descriptions of a gene sequence, and each NDFSA is preferred over the next in the list. Parsing is then performed as follows. Given a sequence σ to be analyzed, the most strongly preferred automaton $\mathcal{M}$ is used to parse σ. If $\mathcal{M}$ produces one or more parses, then a criterion is applied to select some of these parses. Otherwise, the sequence σ is analyzed using the next NDFSA in the list. This process is repeated until an NDFSA accepting σ is found, or the last automaton in the list, i.e., $\mathcal{M}_g$, is tried. Note that $\mathcal{M}_g$ can be chosen such that it accepts every sequence that is rejected by the other NDFSA. For example, $\mathcal{M}_g$ can be a very general description of a gene. It is also reasonable to order the various NDFSA from the most *specific* (i.e., describing particular features of genes) to the most *sensitive* (i.e., the most general) so that the sequences recognized by a given NDFSA are also recognized by the next in the list.

Although the above strategy may seem overly simplistic when compared to the gene structure prediction methods described earlier, actual results confirm its validity. Nevertheless, one should keep in mind that care should be taken in defining the NDFSA and the criterion. In the next section, an NDFSA and a criterion are proposed to identify the coding and non-coding regions in yeast genes.

3.2.2 Experimental Observations using Yeast Sequences

The alternations of exons and introns in protein coding genes can be described by a simple NDFSA encoding the various constraints that are characteristic of those sequences. Figure 3.2 depicts the state transition diagram of an NDFSA, subsequently denoted F_0, that recognizes the set of sequences satisfying the constraints mentioned in Section 2.3, i.e., the $GT - AG$ rule, the triplicity constraint and the open reading frame constraint. Note that some edges in this diagram correspond to more than one transition. For example, the edge between states 0 and 1 corresponds to three transitions involving the symbols a, c and g. Therefore, F_0 consists of 24 states and 69 transitions. State 0 is both the initial state and the final state.

In the state transition graph of F_0, the thin, red edges represent the transitions of F_0 on non-coding bases. The other edges represent the transitions on coding bases and are partitioned in two

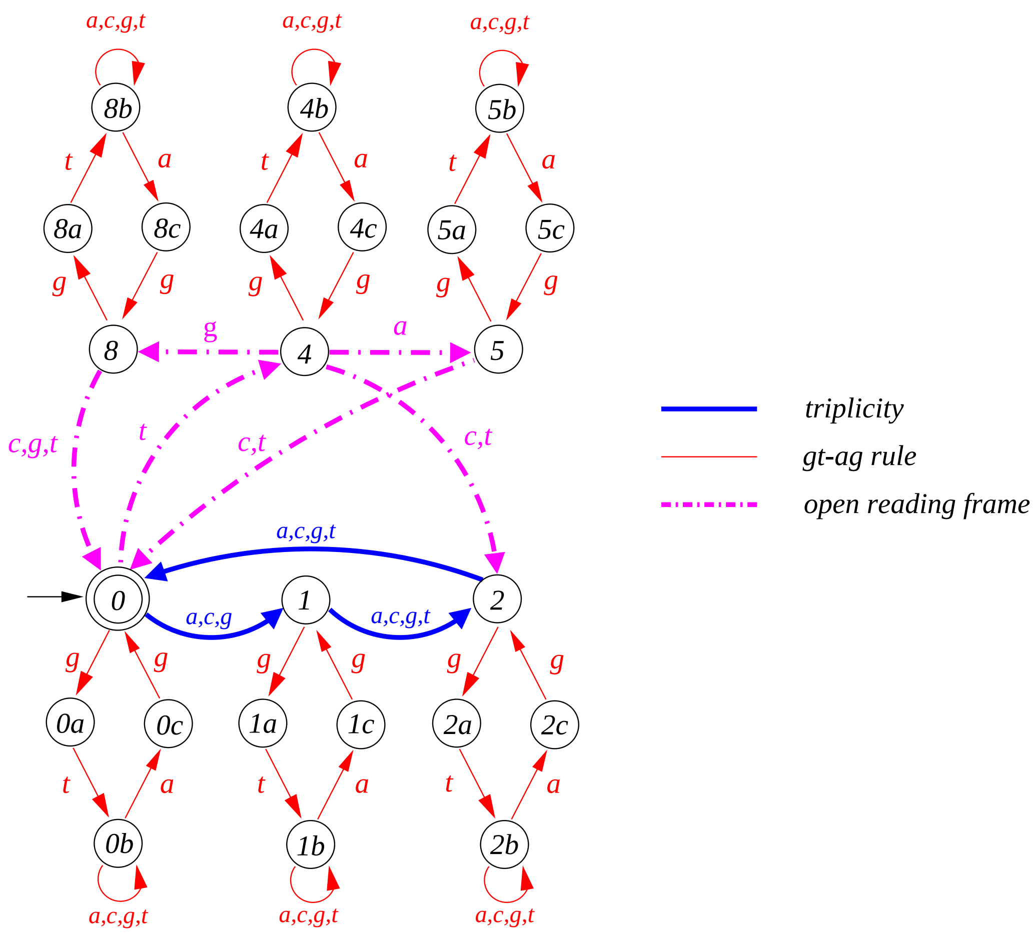


Figure 3.2: An NDFSA describing a gene as an alternation of exons and introns.

groups: the thick, continuous blue edges and the thick, discontinuous magenta edges, which respectively denote the transitions encoding the triplicity constraint and the open reading frame constraint. Note that the magenta color was chosen only to enhance the figure: for detecting exons and introns, the magenta (discontinuous) edges should be considered as blue (continuous). Every time F_0 leaves state 0, 1, or 2, it reads a symbol in reading frame 0, 1, or 2, respectively. Every time F_0 reaches state 4, it has just read the first base t of an in-frame stop codon; every time it reaches state 5 or 8, it has just read the second base g or a of an in-frame stop codon. From states 5 and 8 a transition is only possible if the combination of the first two bases and the third base does not result in the formation of one of the stop codons TGA, TAA, or TAG. The remaining states of F_0 are used to define the introns and correspond to the $GT - AG$ rule.

Slightly more complicated NDFSA than F_0 can be defined to describe various entities such as splice site and branch site consensus. For example, the results compiled in (Ruby and Abelson, 1991) were used to define the NDFSA in Figure 3.3 that describes the typical structure of a Yeast Saccharomyces Cerevisiae (YSC) intron. In this NDFSA, state AZ, which is both the initial and the final state, represents the following states of F_0: 0, 1, 2, 4, 5, 8.

Let us consider the set of types $\mathcal{T} = \{coding, non - coding\}$. By assigning the type *coding* to every transition represented by a (thick) blue or magenta edge and the type $non - coding$ to every other edge, one can obtain for a given sequence a discontinuous curve representing the function r_{coding} (or $r_{non-coding}$). This curve is subsequently referred to as the *coding distribution* of the given sequence. When plotting the coding distributions of each of the 112 Yeast Saccharomyces cerevisiae protein-coding genes found in Genbank 91.0 [3], it can be observed that in almost all sequences, the introns coincide exactly with the regions where the nucleotides have a null probability to be coding (Figure 3.4). More specifically, every intron starts at some position i such that $r_{coding}(i-1) > 0$ and $r_{coding}(i) = 0$, and 97% of all introns end at some position j such that $r_{coding}(j) = 0$ and $r_{coding}(j+1) > 0$. For the remaining

[3] The list of these sequences can be found at `http://www.cs.brandeis.edu/~obaby/thesis.html`

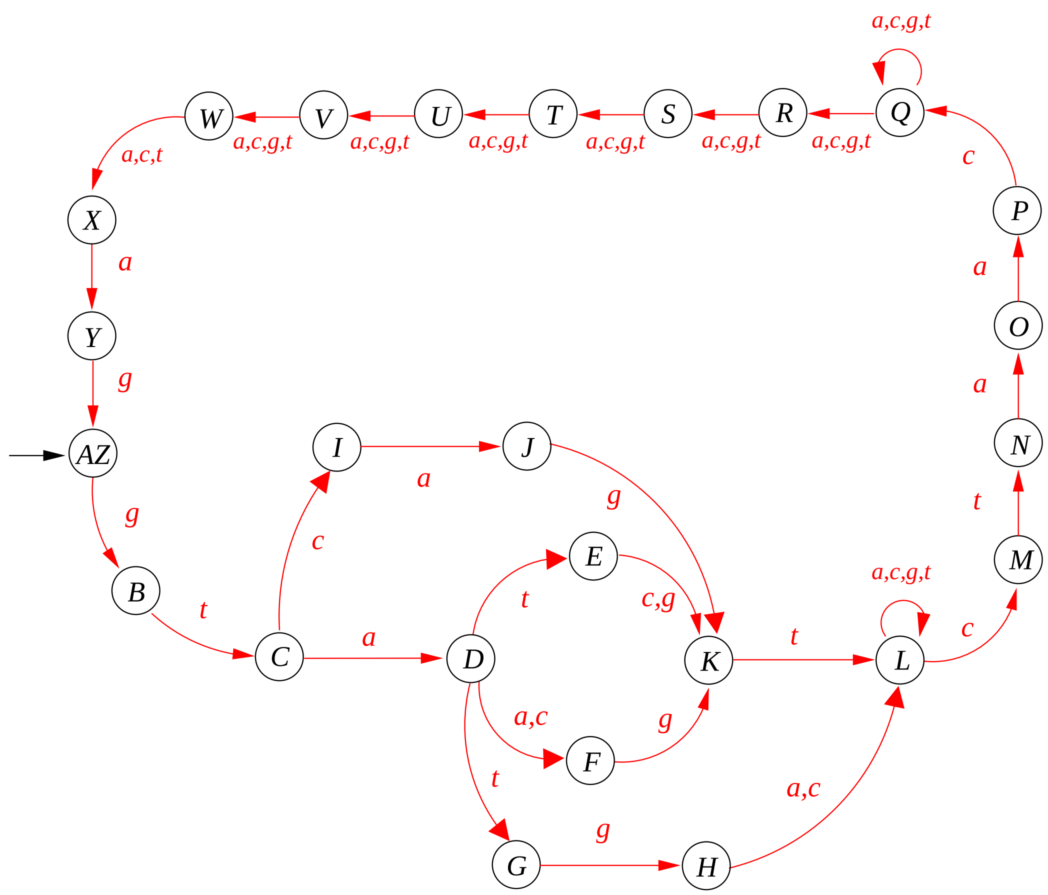


Figure 3.3: An NDFSA describing a Yeast Saccharomyces Cerevisiae (YSC) intron as a sequence starting with any of the $5'$ splice site consensus described by states AZ through L, and ending by any of the $3'$ splice site consensus described by states W through AZ. States L to Q denote the particular consensus corresponding to the branch site, $CTAAC$, that must be present within the intron. Note that the branch site consensus for the YSC is usually $TACTAAC$, but the first two bases are often altered due to base mutations. Finally, states Q to AZ constrain the branch site to be 9 or more bases upstream of the end of the intron.

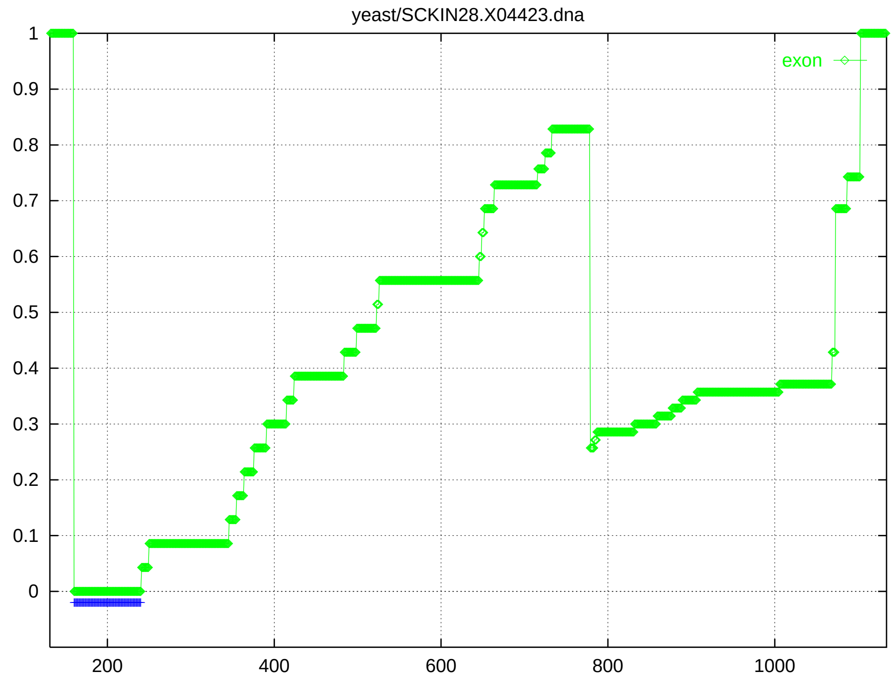


Figure 3.4: An example of coding distribution obtained for the Yeast Saccharomyces Cerevisiae gene of locus $SCKIN28$. The darker line indicates where the intron is. It coincides with the bases that have a null probability of being coding.

3%, the actual splice sites are located only a few bases away from the positions j such that $r_{coding}(j) = 0$ and $r_{coding}(j+1) > 0$.

The above results suggest the use of the following criterion to select a gene model when more than one parse of the input sequence is found: a given nucleotide is coding if there exists at least one parse of the input sequence that makes it coding. Using the formalism introduced in the previous section, this can be formulated as:

$$(coding, r) \in \xi \text{ if and only if } r > 0$$

or equivalently:

$$(non - coding, r) \in \xi \text{ if and only if } r = 0 \tag{3.3}$$

Although the sequence obtained after applying this criterion may not be accepted by the "yeast NDFSA", it is the case in every tested yeast sequence that the resulting parse is indeed a string generated by that NDFSA. However, it should be noted that yeast sequences contain only a few introns and have highly conserved splice sites. We now demonstrate the well-foundedness of the above approach on human genes. Those are known to have more introns and more weakly conserved splice sites than yeast genes.

3.3 Evaluation on Human Sequences and Biological Interpretation

The 238 human multi-exon genes of the learning set of GENSCAN (Burge, 1997) were used to construct an NDFSA denoted F_1 describing the sequences likely to occur at the splice sites in human genes. The NDFSA F_1 mainly differs from the NDFSA F_0 of Figure 3.2 in that states and transitions have been added to describe — whenever possible — every five nucleotide sequence occurring upstream (i.e., before GT or AG) and downstream (i.e., after GT or AG) of the splice sites of the genes in the learning set. This resulted in 276 distinct consensus for the donor splice site, and 404 distinct consensus for the acceptor splice site. It should be noted that the addition of the consensus patterns was done by preserving the triplicity and open reading frame constraints. For example, the addition of the sequence $ATGAT$, which occurs upstream of the donor splice site, was done such that (1) the first A is not preceded by an in-frame TA or TG, (2) the triplet TGA starting at the second position is not in-frame, and (3) the ending t, if in reading frame 0, is not followed by AA, AG or GA.

The following constraints have also been considered in the design of the final NDFSA as they are often observed in human genes: (a) no exons but the initial one can be shorter than 5 codons, (b) no intron is less than 70 nucleotides long, and (3) no AG can occur between the fifth and twentieth nucleotide upstream of the acceptor site. The first two constraints — referred to as the *length constraints* in the sequel — resulted in an NDFSA denoted F_2, and the last in an NDFSA denoted F_3 containing 9191 states and 21373 transitions (see Figure 3.5).

Sequence analysis is performed in a hierarchical way. Each of the NDFSA F_0, F_1, F_2, and F_3, is characterized by a collection of constraints that belong to different levels of preference. The constraints defining F_0 are required (*hard* constraints) whereas those defining the other NDFSA express preferences (*soft* constraints), with F_i being preferred over F_{i-1}, for $1 \leq i \leq 3$. Given a sequence σ to analyze, first the most specific NDFSA, i.e., F_3, is used to parse σ. If F_3 produces one or more parses, then the criterion of equation 3.3 is applied to determine the exon-intron structure of σ. Otherwise, the next most strongly preferred NDFSA in the hierarchy is used. This process is repeated until an NDFSA producing one or more parses is found. Note that in the last resort, the most general NDFSA F_0 can always be used to predict gene structure.

Also note that the particular order of the NDFSA in the hierarchy of Figure 3.5 was specifically chosen to maximize prediction accuracy for the 176 human sequences of the test set proposed in (Burset and Guigo, 1996). The sensitivity and the specificity of the proposed approach on these 176 human sequences were respectively $S_n = 0.91$ and $S_p = 0.83$. The approximate correlation and correlation

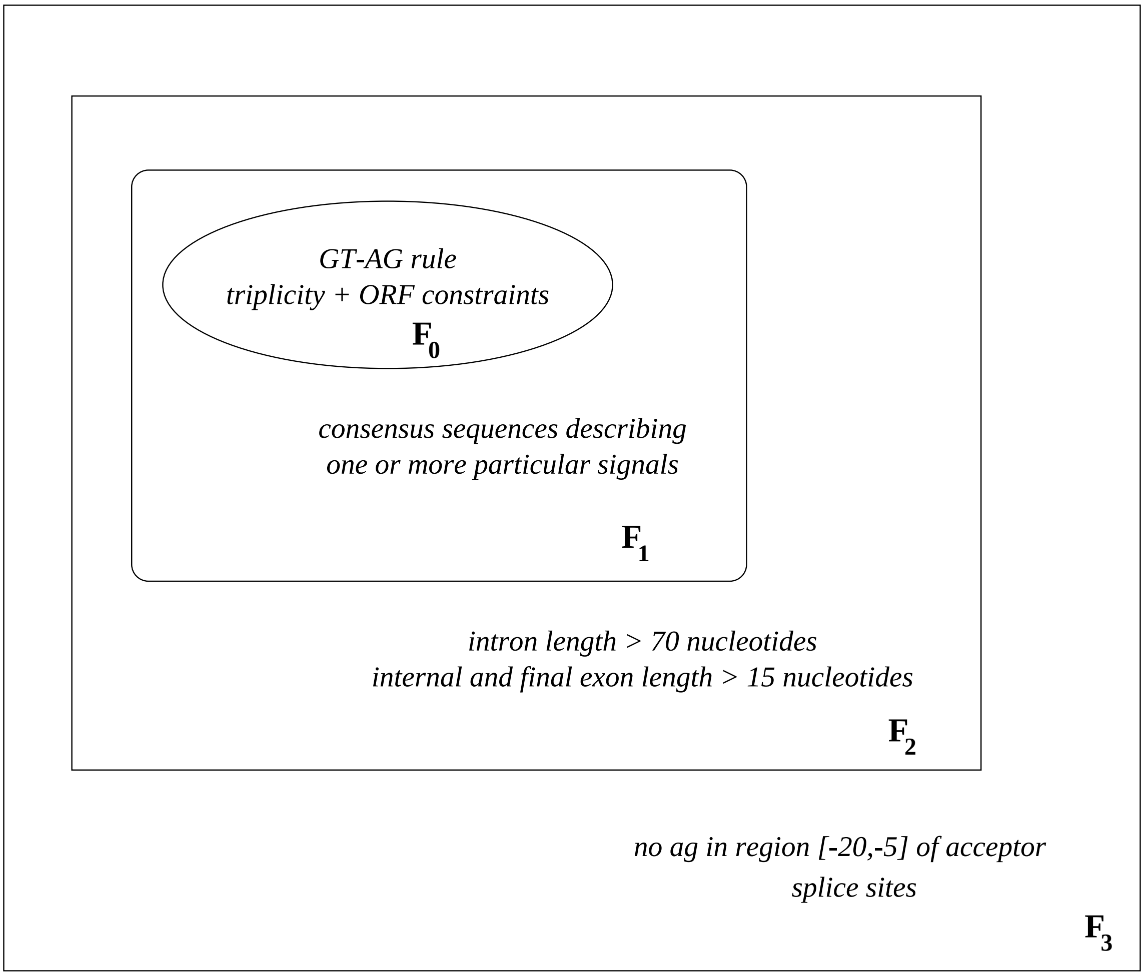


Figure 3.5: The NDFSA used to parse human genes: F_0 is the most general model; F_3 the most specialized.

most general NDFSA	R	S_n	S_p	AC	CC	order of application
F_3	0.92	0.91	0.83	0.80	0.80	F_3, F_2, F_1, F_0
F_2	0.98	0.99	0.72	0.75	0.74	F_2, F_1 F_0
F_1	1.00	0.99	0.70	0.73	0.71	F_1, F_0
F_0^+	1.00	0.99	0.38	0.27	0.25	F_0^+, F_0
F_0	1.00	1.00	0.34	0.13	0.08	F_0
F_1^-	1.00	1.00	0.33	−0.33	−0.33	F_1^-
F_3^-	1.00	0.99	0.36	−0.18	−0.19	F_3^-

Figure 3.6: Comparison of various NDFSA: R is the recognition rate of each model, i.e., the ratio of sequences for which one or more parses are found. For example, F_3 was able to predict gene structure in 92% of the sequences. The NDFSA F_0^+ is equivalent to F_2 except that splice sites are identified by the $GT - AG$ rule only, i.e., $F_0^+ = (F_2 \backslash F_1) \cup F_0$. The NDFSA F_1^- describes the consensus at the splice sites only, i.e., $F_1^- = F_1 \backslash F_0$. The NDFSA F_3^- is identical to F_3 except that it allows in-frame stop codons to occur in genes. The values of S_n, S_p, AC and CC, were calculated for each sequence and averaged over all sequences. Note that these values were obtained by applying first the most general NDFSA, and then the other, alternative NDFSA.

coefficient were $AC = CC = 0.80$. Although the value of AC obtained is the same as the best existing program that does not use sequence homology (Solovyev et al., 1994), it should be noted that in our approach this accuracy is obtained with a much simpler and more explicit model. In addition, the average proportion of exons correctly identified (with exact splice site boundaries) was 66%, while the average proportion of predicted exons that were correct was 65%. Therefore, the program is capable of predicting exons with a reasonable degree of success.

Figure 3.6 shows the performance of the various NDFSA on the above-mentioned test set. First, note the trade-off between the sensitivity S_n and the specificity S_p, which reflects the fact that the NDFSA F_3, F_2, F_1, and F_0, are ordered from the most stringent to the most general one. Also note that, even though it is expected that the sensitivity S_n increases with the ratio R of genes for which one or more parses are found, there is no *a priori* reason that R approximate S_n, as is the case in Figure 3.6.

Also observe that the model F_1^-, which relies only on the description of the consensus at the splice sites (with no additional constraints), is insufficient to allow a realistic discrimination between exons and introns. Such a "pattern matching" model predicts most nucleotides to be coding (with $S_n = 1$) on the above-mentioned test set, but has a low specificity ($S_p = 0.33$), and negative approximate correlation and correlation coefficient: $AC = CC = -0.33$. In contrast, the model F_0^+, which considers all of the aforementioned constraints but uses the $GT - AG$ rule instead of the five nucleotide consensus, performs better than the model F_1^-. The sensitivity of F_0^+ is almost the same as that of F_1^- ($S_n = 0.99$), but the specificity is higher ($S_p = 0.38$), and the approximate correlation and correlation coefficient are this time positive, with respective values $AC = 0.27$ and $CC = 0.25$.

The significant increase in the values of S_p, CC, and AC, between F_0 and F_1 indicates that prediction accuracy greatly benefits from the specification of the splice site consensus. The requirement that no AG occur in the region $[-20, -5]$ of the donor splice site also results in improved prediction (see F_2 and F_3), although 8% of the sequences tested did have an AG in that region. Constraining the length of exons and introns seem to have an impact on prediction accuracy too, although the difference in prediction accuracy is more pronounced when splice site consensus are not specified (as in F_0) than when they are (F_1). Finally, the most striking difference in prediction accuracy is observed when the open reading frame constraint is dropped. Indeed, with $AC = -0.18$ and $CC = -0.19$, it is unlikely that the NDFSA F_3^-, which is identical to F_3 except that it allows illegal strings having in-frame stop codons, predicts exons and introns correctly.

A possible explanation for the interesting results obtained for the human genes (and the yeast genes) is that intron splicing occurs by recognition of short sequences at the splice sites (and possibly also at the branch sites). Many such sequences can be recognized by the spliceosome but only certain arrangements of splice sites are compatible with the basic biological rules (e.g., no in-frame stop codon). The many parses obtained for a given gene correspond to the various ways this gene can be spliced. Thus non-determinism is used to express that intron removal is tissue specific. The conjecture proposed in this thesis, and to be further investigated by additional testing, is that (a) there are certain regions in genes that are always non-coding no matter how splicing is performed, and (b) splicing occurs exactly where these regions are.

A few remarks are in order. First of all, the rules (a) and (b) are compatible with the following observation (Ruby and Abelson, 1991), which was made for the yeast genes containing a single intron (most of them do): if more than one potential donor (or acceptor) splice sites are detected, then the splice site closer to the branch site is the actual one. In other words, the sole intron of a yeast gene is defined as the subsequence of this gene for which no alternative parses can be found. Obviously, the rules (a) and (b) are a generalization of the rule for yeast genes in the case the sequence being analyzed contains more than one intron.

Secondly, the predicted genes (i.e., after applying the criterion) are generated by the NDFSA significantly more often than expected on a random basis. Indeed, assume that the rules (a) and (b) do not hold. Then, it would be expected that about one third of all predicted genes satisfy the triplicity constraint, and that less than one third of all predicted genes satisfy all constraints encoded by the NDFSA. However, almost 61% of all predicted genes satisfy the constraints encoded by the NDFSA. This result strongly supports the above conjecture. It is expected that the ratio of predicted genes satisfying all constraints defined by the NDFSA will increase as more constraints are used to describe gene models.

Finally, the significant decrease in prediction accuracy when in-frame stop codons are ignored supports the theory of stop-codon walk mechanism mentioned in Section 2.1.2 (Senapathy, 1988). Recall that, according to this theory, the splicing mechanism originally evolved to avoid the too-frequent occurrences of stop codons in genes. In other words, the role of the splicing machinery is to discard every region of a gene that contains one or more in-frame stop codons. By rejecting all parses that do not satisfy the open reading frame constraint, the NDFSA proposed above actually simulate the stop-codon walk mechanism, i.e., they assign the type $non-coding$ to every region of a gene that contains in-frame stop codons.

The careful reader may have noticed that we were able to obtain the coding distributions of various yeast and human genes even though the number of parses of these genes may grow exponentially with their length. The objective of the next section is to provide the algorithms that have been used to compute efficiently the quantities $P_{ij}(k)$ and $r_t(k)$ defined in Equations 3.1 and 3.2.

3.4 Sequential and Parallel Algorithms for Predicting Gene Models

In this section, we describe a sequential and a parallel algorithm for determining which of the nucleotides in a given sequence are coding or non-coding. For clarity, we first use a directed acyclic graph (DAG) to describe the sequential algorithm. Every node in this DAG represents a state of the NDFSA at a given time whereas every edge corresponds to a possible transition of the NDFSA between two states at consecutive discrete time intervals. The proposed algorithm consists of three steps which are illustrated in Figure 3.7 for the toy example of Figure 3.1. The first step constructs the DAG, the second step computes the ratios $r_t(k)$, and the third step applies the criterion for distinguishing exons from introns. Note that in practice, the construction of the DAG can be avoided. This property allows us to design an efficient parallel version of the sequential algorithm.

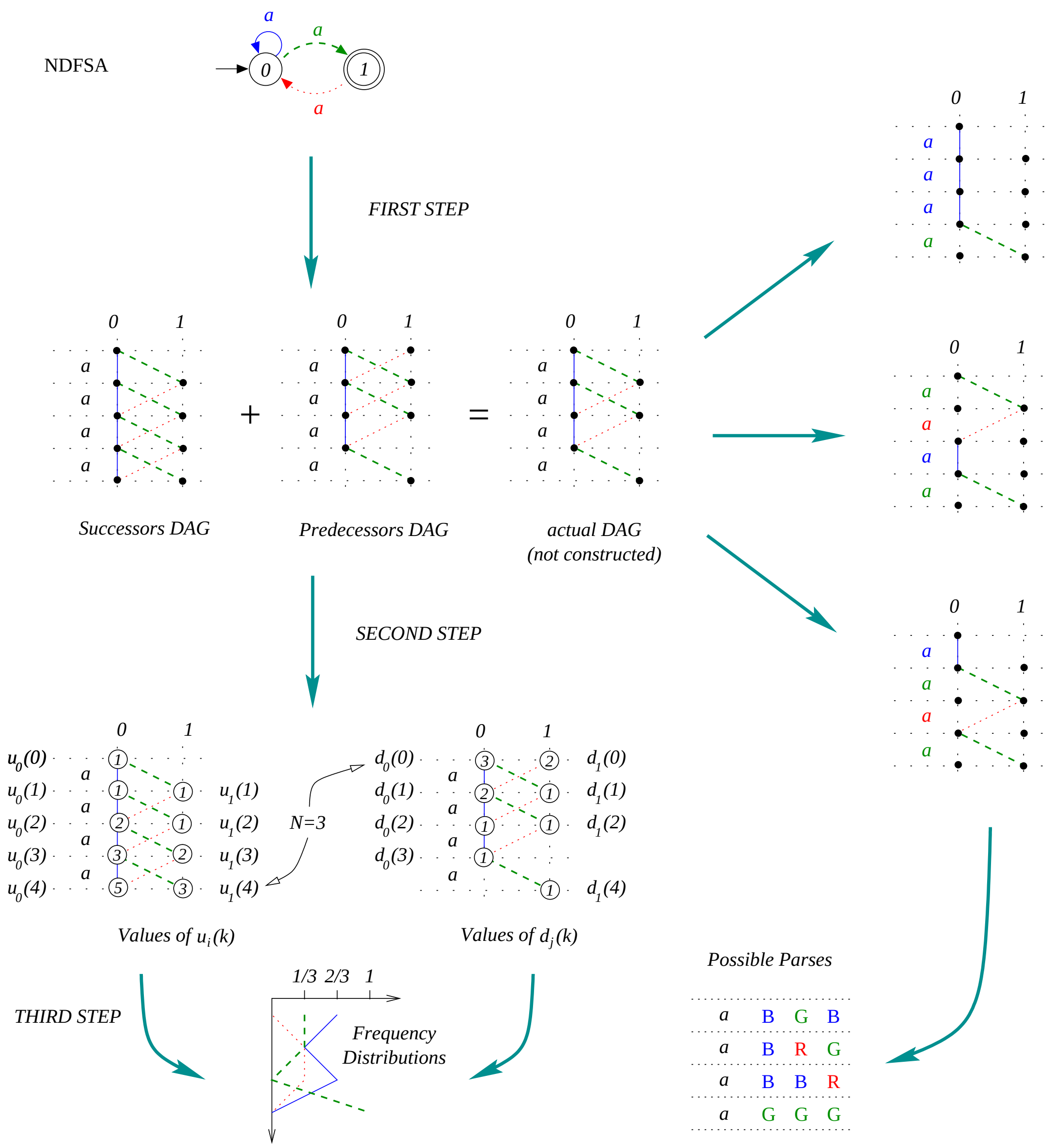


Figure 3.7: Generation of frequency distributions for the toy example of Figure 3.1. The DAG obtained when considering the input string *aaaa* is represented by two arrays, *Successor* and *Predecessor*. In this DAG, there are three paths going from the source to the sink node, reflecting the fact that there are three possible ways of coloring the string *aaaa*.

3.4.1 DAG Generation

The DAG being generated can be represented by two two-dimensional arrays, $Successor$ and $Predecessor$, whose respective elements are *sets* that represent the successors and the predecessors of every node in the DAG. An element j belongs to $Successor[k, i]$ if there exists a transition sequence from the initial state to state q_i, and it is possible to make a transition from state q_i to state q_j upon reading the k^{th} input symbol. Note that this does not necessarily mean that there is a sequence of transitions from state q_j to the final state. Similarly, j belongs to $Predecessor[k, i]$ if there exists a transition sequence from state q_i to the final state, and it is possible to make a transition from state q_j to state q_i upon reading the $(k-1)^{\text{th}}$ input symbol. Note again that this does not guarantee that there is a sequence of transitions from the initial state to state q_j. The array $Successor$ is constructed by determining the states that are successively reached from the initial state upon reading the symbols of the input sequence. The array $Predecessor$ is constructed similarly but by determining the states that are successively reached from the final state, assuming that the sequence is read in reverse order.

Every element j in an entry $[k, i]$ of the $Successor$ array represents an edge of the DAG linking the nodes (k, i) and $(k+1, j)$. Similarly, every element j in an entry $[k, i]$ of the $Predecessor$ array represents an edge of the DAG linking the nodes $(k-1, j)$ and (k, i). In practice, the number of elements in every entry $[k, i]$ of the $Successor$ and $Predecessor$ arrays is small. The nodes $(k-1, i)$ and (k, j) are said to be at depth k and $k+1$, respectively. There exist two particular nodes in the DAG, one at depth 0 and the other at depth n, which correspond to the left and right boundaries of the input string. These are the node $(0, 0)$, which represents the source of the DAG, and the node (n, f), which represents its sink.

From the above description, it follows that the maximum number of nodes in the DAG is nf where n is the length of the input string and f the number of states in the NDFSA. Also, the number of edges in the DAG is at most ne, where e is the number of transitions in the NDFSA. Therefore, all

Successor and *Predecessor* sets can be computed in $O(ne)$ time in the worst case.

3.4.2 Computation of the Ratio $r_t(k)$ and Application of the Criterion

For $1 \leq k \leq n$, the computation of the ratio $r_t(k)$ is done in three steps. First, a pair of integers $(u_i(k-1), d_j(k))$, which respectively denote the number of paths from node $(k-1, i)$ to the source and the number of paths from node (k, j) to the sink, is assigned to every edge $\mu = (k-1, i) \rightarrow (k, j)$ of the DAG. Then, the probability $P_{ij}(k)$ that every edge μ is passed through by a path from the source node to the sink node is calculated. Finally, the quantity $r_t(k)$ representing the probability that the k^{th} input symbol is of type t is computed for every position k and every type t.

The pair $(u_i(k-1), d_j(k))$ can be computed for every edge using a two-pass algorithm similar to the forward-backward algorithm (Rabiner, 1989). The first pass computes the quantities $u_i(k-1)$, and the second the quantities $d_j(k)$. The first pass starts by marking the source node for which $u_0(0) = 1$. Then, for every node (k, i) at depth k, the set of predecessors of (k, i) is determined using the *Predecessor* array. The number of paths between the source node and node (k, i) is obtained by summing up the number of paths between the source node and the predecessors of (k, i) that are marked. The node (k, i) is then marked unless it has no marked predecessor. The successors of every node at depth k are then determined using the *Successor* array, and the above process is repeated until the sink node is reached, i.e., $k = n$. The second pass proceeds in a similar way but by starting with the sink node for which $d_f(n) = 1$. The number of paths between every node at depth k and the sink is obtained from the values computed for the nodes at depth $k+1$. This process is repeated until the source node is reached. Since there are at most e edges between two consecutive depths in the DAG, all quantities $u_i(k-1)$ and $d_j(k)$ can be computed in $O(ne)$ time in the worst case.

Let N denote the total number of paths from the source to the sink: $N = u_f(n) = d_0(0)$. The total number of paths $N_{ij}(k)$ passing through a particular edge $\mu = (k-1, i) \rightarrow (k, j)$ is obtained by multiplying $u_i(k-1)$ by $d_j(k)$. The likelihood that edge μ is passed through by a path from the source to the sink is then expressed by the ratio:

$$P_{ij}(k) = \begin{cases} \frac{u_i(k-1)d_j(k)}{N} & \text{if } (q_i, x_k, q_j) \in T \\ \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

which can be calculated in $O(ne)$ time as well.

Given a set of transitions S having a particular type t, some of the edges linking the nodes at depth $k-1$ and the nodes at depth k may correspond to a transition in S. The sum of the probabilities associated with those edges represent, then, the probability that the k^{th} symbol is of type t, i.e., the quantity $r_t(k)$ as defined in Equation (3.2). By calculating the probability of every symbol in the input sequence to be of type t, one obtains the desired frequency distribution. The time complexity of this step depends on the size $|S|$ of the set S. In the worst case, the frequency distribution can be obtained in $O(n|S|)$ time, or $O(ne)$ time since $|S| \leq e$. The user-defined criterion ξ can be applied to determine, based on the values of the ratios $r_t(k)$, which of the paths in the DAG represent valid parses. The complexity of this step mainly depends on the definition of ξ. For example, the criterion proposed in equation (3.3) to discriminate introns from exons can be computed in constant time.

Note that for the purpose of applying the particular criterion in equation (3.3), it is not necessary to compute the ratios $r_t(k)$. Indeed the only information needed in that case is whether or not a transition of type t can be made upon reading the k^{th} input symbol. This information can be obtained by determining the edges in the DAG that are both in the transitive closure of the source and in the transitive closure of the sink. However, the computation of the ratios $r_t(k)$ is essential when designing a new NDFSA, as a careful examination of the frequency distributions may indicate the absence of new desirable properties or states. Also note that it is useful to compute the number of paths between the source and the sink as it is a rough estimator of the quality of the prediction: a low value of N is likely to correspond to a more accurate prediction than a large value of N.

From the above complexity analysis, it follows that the coding and non-coding regions of a gene

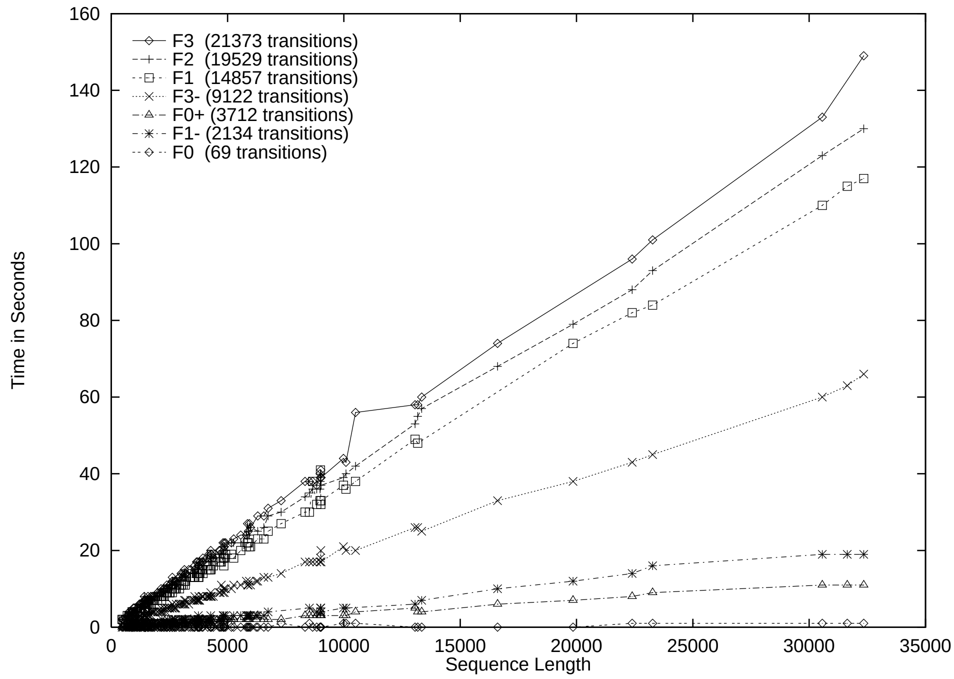


Figure 3.8: Running times of the sequential algorithm for the NDFSA describing human genes. Although F_1^- contains fewer transitions than F_0^+, parsing a sequence with F_1^- takes more time than with F_0^+. This is because the former contains "bottleneck" states corresponding to intron boundaries: every bottleneck state has an incoming and outgoing transition for each of the donor or acceptor splice site consensus. In F_0^+, there is no bottleneck because the only consensus used is that given by the $GT - AG$ rule.

of length n can be determined in $O(ne)$ time, where e if the number of transitions of the NDFSA. The running times of the sequential algorithm are plotted in Figure 3.8 for each of the NDFSA of Figure 3.6. Although it takes only a few seconds to analyze an average size sequence (i.e., of length $5,000bp$), it is expected that parsing a sequence containing multiple genes will take much more time. For this reason, we also propose a parallel version of the sequential algorithm.

3.4.3 An Efficient Parallel Algorithm

We now use a prefix sum to reformulate the sequential algorithm. This reformulation allows us to derive two parallel algorithms. The first uses a divide-and-conquer approach to compute the prefix sum. It is a generalization of the parallel algorithm in (Hillis and Steele, 1986) which determines if a given string is generated by a given deterministic finite-state automaton. The main drawback of the first algorithm is its worst time complexity: it takes $O((\frac{n}{p} + \log p)f^3))$ time on a PRAM with $O(p)$ processors. The second algorithm has a lower time complexity: it runs in $O(\frac{nf^2}{p})$ time on a PRAM with $O(p)$ processors, and in $O(\frac{ne}{p})$ time when using a sparse representation of the NDFSA. The second algorithm consists of computing the prefix sum as in sequential but by applying the prefix operator in parallel.

Given a set of initial values $a_0, a_1, \ldots, a_{m-1}$, a prefix sum takes a binary associative operator $\otimes$ and returns the vector $b = (b_0, b_1, \ldots, b_{m-1})$ such that:

$$b_k = \begin{cases} a_0 & \text{if } k = 0 \\ b_{k-1} \otimes a_k & \text{for } 0 < k < m \end{cases}$$

For $p \geq 1$, let $T^p_\otimes$ be the time taken to perform the operation $\otimes$ with p processors. In general, there are two ways to compute the vector b on a PRAM architecture (see Figure 3.9 and 3.10):

1. The vector b can be computed by a divide-and-conquer algorithm, in which case the algorithm takes $O((\frac{m}{p} + \log p)T^1_\otimes)$ time with $O(p)$ processors. More generally, if the binary operator $\otimes$ is semi-associative, i.e., there exists a binary associative operator $\odot$ such that: $\forall x.y.z,\ (x \otimes y) \otimes z = x \otimes (y \odot z)$, then b can be computed in $O((\frac{m}{p} + \log p)(T^1_\otimes + T^1_\odot))$ time using $O(p)$ processors, where $T^1_\odot$ is the sequential time taken by $\odot$ (Blelloch, 1993). The operator $\odot$ is called the *companion operator* of $\otimes$.

2. The vector b can be computed as in sequential except that the computation of b_k from b_{k-1} and a_k is performed in parallel, i.e., $\oplus$ is applied in parallel to its operands. This algorithm takes $O(mT^p_\otimes)$

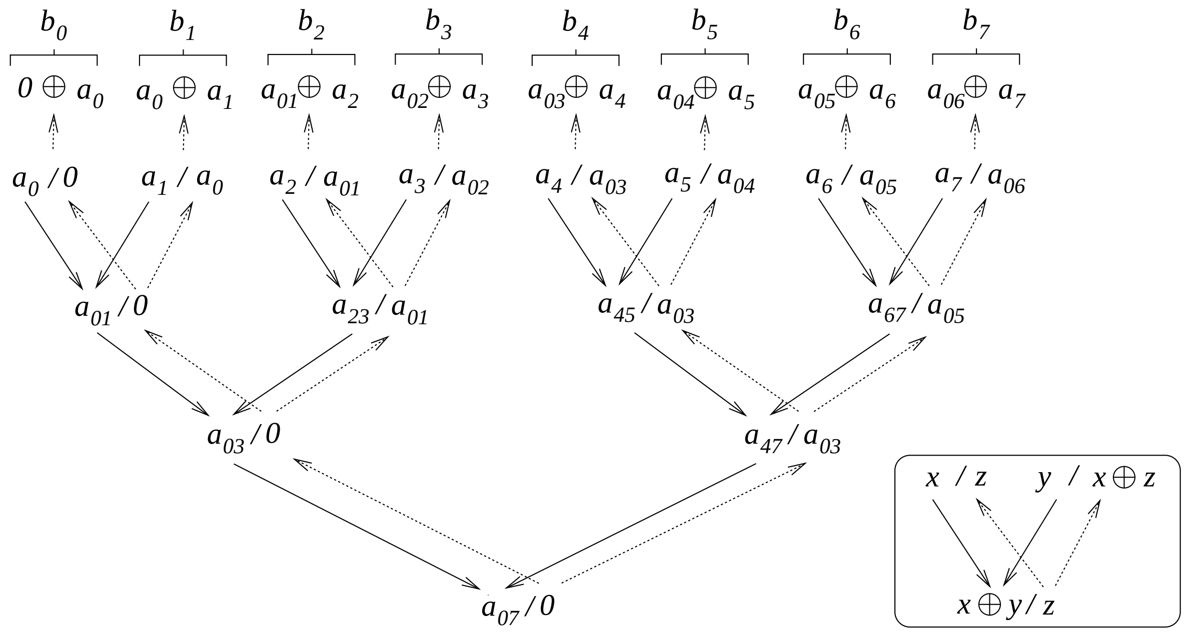


Figure 3.9: Divide-and-Conquer prefix sum computation. The algorithm proceeds in two passes. The first pass (plain arrows) computes the value of $b_{m-1} = a_{07}$ ($a_{ij} = a_i \oplus \cdots \oplus a_j$) by successively applying $\oplus$ (in sequential) to the values x and y from the children of each vertex of the binary tree. The second pass (dotted arrows) then uses the partial sums obtained at each vertex to compute the vector b: the root of the tree is first initialized with 0; then on each step, each vertex passes to its left child its own value z, and to its right child, $\oplus$ sequentially applied to the value x from the left child from the first pass and its own value z.

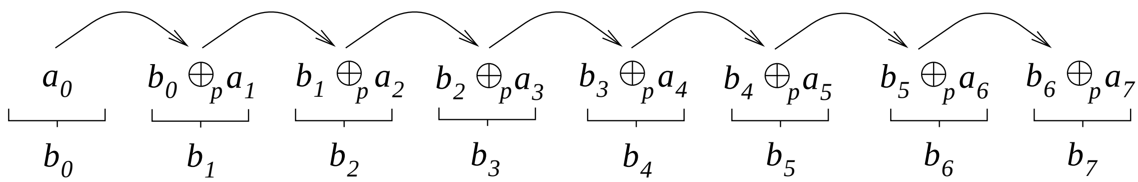


Figure 3.10: Parallel prefix sum algorithm using a parallel computation of $\oplus$: b_k is computed after b_{k-1}, but the operation $\oplus$ is performed in parallel (denoted $\oplus_p$ above).

time with $O(p)$ processors.

Many known parallel algorithms can be expressed as a prefix sum. In particular, it is shown in (Hillis and Steele, 1986) that the parsing of strings generated by deterministic finite-state automata can be reduced to a simple prefix sum computation. In the following, this result is extended to the non-deterministic case; that is, for every NDFSA, the successive sets of states of the NDFSA is expressed in terms of a prefix sum. The key idea of this approach is to represent the set of all the current states of the NDFSA by a single vector corresponding to the coefficient a_k in the above equation (note that in the deterministic case this vector contains a single element). Even though the transformation of a non deterministic finite-state automaton into an equivalent deterministic one is exponential, it is shown below that the deterministic simulation of an NDFSA with $O(e)$ transitions on a particular string of length n can be done efficiently in $O(\frac{ne}{p})$ time on a PRAM with $O(p)$ processors whereas that simulation by a divide-and-conquer algorithm takes $O((\frac{n}{p} + \log p)f^3)$ time.

Actually, since the problem is to determine the probability that each symbol in the input string is generated by a given transition, the vectors in the proposed algorithm do not represent the states of the NDFSA that are reached upon reading the symbols of the input string; instead, they represent the number of transition sequences leading to or starting from a given state at a given position in the input string. In the following, it is shown that these vectors can be computed by performing vector-matrix or matrix-matrix multiplications. The various definitions provided below are exemplified in Figure 3.11 for the simple NDFSA of Figure 3.7.

Let, as above, $\mathcal{M} = (Q, \Sigma, T, q_0, q_f)$ be a finite-state automaton such that $Q = \{q_0, q_1, \ldots, q_f\}$, q_0 being the only initial state and q_f the only final state. The transition relation T of $\mathcal{M}$ can be described by $|\Sigma|$ boolean matrices; for each symbol $x \in \Sigma$, the matrix $T(x)$ describes the transitions allowed on x,

$$\mathcal{M} = (\overbrace{\{0,1\}}^{Q}, \overbrace{\{a\}}^{\Sigma}, \overbrace{\{(0,a,0),(0,a,1),(1,a,0)\}}^{T}, \overbrace{0}^{q_0}, \overbrace{1}^{q_1}) \qquad \sigma = \sigma_1\sigma_2\sigma_3\sigma_4 = a\ a\ a\ a$$

$$T(a) = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \qquad\Big|\qquad T(a)^T = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$u(0) = (\overbrace{1}^{u_0(0)}\ \overbrace{0}^{u_1(0)}) \qquad\Big|\qquad d(4) = (\overbrace{0}^{d_0(4)}\ \overbrace{1}^{d_1(4)})$$

$$u(1) = u(0)\ T(\sigma_1) = (1\ 0)\ T(a) = (\overbrace{1}^{u_0(1)}\ \overbrace{1}^{u_1(1)}) \qquad\Big|\qquad d(3) = d(4)\ T(\sigma_4)^T = (0\ 1)\ T(a)^T = (\overbrace{1}^{d_0(3)}\ \overbrace{0}^{d_1(3)})$$

$$u(2) = u(1)\ T(\sigma_2) = (1\ 1)\ T(a) = (\overbrace{2}^{u_0(2)}\ \overbrace{1}^{u_1(2)}) \qquad\Big|\qquad d(2) = d(3)\ T(\sigma_3)^T = (1\ 0)\ T(a)^T = (\overbrace{1}^{d_0(2)}\ \overbrace{1}^{d_1(2)})$$

$$u(3) = u(2)\ T(\sigma_3) = (2\ 1)\ T(a) = (\overbrace{3}^{u_0(3)}\ \overbrace{2}^{u_1(3)}) \qquad\Big|\qquad d(1) = d(2)\ T(\sigma_2)^T = (1\ 1)\ T(a)^T = (\overbrace{2}^{d_0(1)}\ \overbrace{1}^{d_1(1)})$$

$$u(4) = u(3)\ T(\sigma_4) = (3\ 2)\ T(a) = (\overbrace{5}^{u_0(4)}\ \underbrace{\overbrace{3}^{u_1(4)}}_{N}) \qquad\Big|\qquad d(0) = d(1)\ T(\sigma_1)^T = (2\ 1)\ T(a)^T = (\underbrace{\overbrace{3}^{d_0(0)}}_{N}\ \overbrace{2}^{d_1(0)})$$

$$\begin{array}{cc} & d(k) \\ u(k-1) & \underbrace{\begin{pmatrix} u_0(k-1)d_0(k)/N & u_0(k-1)d_1(k)/N \\ u_1(k-1)d_0(k)/N & 0 \end{pmatrix}}_{P(k)} \end{array}$$

$$\begin{array}{cc} & \overbrace{(2\ 1)}^{d(1)} \\ \overbrace{(1\ 0)}^{u(0)} & \underbrace{\begin{pmatrix} 2/3 & 1/3 \\ 0 & 0 \end{pmatrix}}_{P(1)} \end{array} \qquad \begin{array}{cc} & \overbrace{(1\ 1)}^{d(2)} \\ \overbrace{(1\ 1)}^{u(1)} & \underbrace{\begin{pmatrix} 1/3 & 1/3 \\ 1/3 & 0 \end{pmatrix}}_{P(2)} \end{array} \qquad \begin{array}{cc} & \overbrace{(1\ 0)}^{d(3)} \\ \overbrace{(2\ 1)}^{u(2)} & \underbrace{\begin{pmatrix} 2/3 & 0 \\ 1/3 & 0 \end{pmatrix}}_{P(3)} \end{array} \qquad \begin{array}{cc} & \overbrace{(0\ 1)}^{d(4)} \\ \overbrace{(3\ 2)}^{u(3)} & \underbrace{\begin{pmatrix} 0 & 3/3 \\ 0 & 0 \end{pmatrix}}_{P(4)} \end{array}$$

Figure 3.11: Matrix description of the algorithm. $\mathcal{M}$ denotes the NDFSA provided in Figure 3.7; $T_{ij}(a) = 1$ if there exists a transition on input symbol a from state i to state j; $T(a)^T$ denotes the transpose of $T(a)$; $\sigma = aaaa$ is the input sequence of $\mathcal{M}$, of length $n = 4$; $u_i(k) = \alpha$ if there exist α distinct transition sequences between initial state 0 and state i assuming that the first k symbols of σ have been read; $d_j(k) = \beta$ if there exist β distinct transition sequences between state j and final state 1 assuming that $n - k$ symbols of σ remains to be read; $P_{ij}(k) = \gamma$ if $\mathcal{M}$ can make a transition from state i to state j with probability γ upon reading the k^{th} symbol of σ.

i.e., for $0 \leq i, j \leq f$:

$$T_{ij}(x) = \begin{cases} 1 & \text{if } (q_i, x, q_j) \in T \\ 0 & \text{otherwise} \end{cases}$$

Using the above definition, the number of distinct transition sequences $u_i(k)$ that lead to state q_i upon reading the k^{th} symbol of an input sequence σ is, for $1 \leq k \leq n$:

$$u_i(k) = \sum_{(q_j, \sigma_k, q_i) \in T} u_j(k-1) = \sum_{0 \leq j \leq f} u_j(k-1) T_{ji}(\sigma_k) \tag{3.5}$$

with as base case:

$$u_i(0) = \begin{cases} 1 & \text{if } i = 0 \\ 0 & \text{for } 0 < i \leq f \end{cases} \tag{3.6}$$

Let $u(k) = (u_0(k), u_1(k), \ldots, u_f(k))$ be the vector containing the values of $u_i(k)$ for every state $q_i \in Q$. Using a matrix notation, Equations (3.5) and (3.6) can be rewritten as a prefix sum involving vector-matrix multiplications:

$$u(k) = \begin{cases} (1, 0, \ldots, 0) & \text{if } k = 0 \\ u(k-1) T(\sigma_k) & \text{for } 1 \leq k \leq n \end{cases} \tag{3.7}$$

Similarly, the number of distinct transition sequences $d_i(k)$ starting on state q_i and ending on final state q_f after reading the last $n - k$ symbols of σ is, for $0 \leq k < n$:

$$d_i(k) = \sum_{(q_i, \sigma_{k+1}, q_j) \in T} d_j(k+1) = \sum_{0 \leq j \leq f} T_{ij}(\sigma_{k+1}) d_j(k+1) \tag{3.8}$$

with as base case:

$$d_i(n) = \begin{cases} 1 & \text{if } i = f \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

Let $d(k) = (d_0(k), \cdots, d_i(k), \cdots, d_f(k))$ be the vector containing the values of $d_i(k)$ for every state $q_i \in Q$. Denoting $T(\sigma_k)^T$ the transpose of $T(\sigma_k)$, Equations (3.8) and (3.9) can be rewritten as the following prefix sum:

$$d(k) = \begin{cases} d(k+1)T(\sigma_{k+1})^T & \text{for } 0 \le k < n \\ (0, \ldots, 0, 1) & \text{if } k = n \end{cases} \tag{3.10}$$

Each component of vectors $u = (u(0), \ldots, u(n))$ and $d = (d(0), \ldots, d(n))$ is a vector containing $f+1$ elements whose product with any of the matrices $T(x)$ or $T(x)^T$, for $x \in \Sigma$, can be computed in $O(f^2)$ time. Therefore, the vectors u and d can be computed in $O(nf^2)$ in sequential. Furthermore, it is well known that the prefix operator, vector-matrix multiplication, can be applied in $O(\frac{f^2}{p})$ time when using $O(p)$ processors ($p \ge 1$). Therefore, u and d can be obtained in $O(\frac{n}{p}f^2)$ time. In practice, the matrices $T(x)$ and $T(x)^T$ are rather sparse so that only the non zero values are used for computing and storing the results of the vector-matrix multiplications (Barrett et al., 1994). Therefore, it is possible to compute u and d in $O(\frac{ne}{p})$ time with $O(p)$ processors, where e is the number of transitions of the NDFSA.

Note that the vectors u and d can also be computed by a divide-and-conquer algorithm since vector-matrix multiplication is semi-associative, and it has matrix multiplication as companion operator. However, this approach involves both vector-matrix and matrix-matrix products; It results that the divide-and-conquer algorithm has $O((\frac{n}{p} + \log p)(f^2 + f^3)) = O((\frac{n}{p} + \log p)f^3)$ time complexity on a PRAM with $O(p)$ processors. Therefore it is not as efficient as the one proposed above.

For $1 \le k \le n$, the probability that the k^{th} symbol of σ causes $\mathcal{M}$ to make a transition from state q_i to state q_j is as given by Equation (3.4), i.e.:

$$P_{ij}(k) = \frac{u_i(k-1)d_j(k)}{N} T_{ij}(x_k)$$

where N is the number of parses of the input, i.e., $N = u_f(n) = d_0(0)$. Provided that the vectors u and d have been computed, all quantities $P_{ij}(k)$, for $1 \le k \le n$, can be computed in $O(\frac{n}{p}f^2)$ time on a PRAM with $O(p)$ processors, or $O(\frac{ne}{p})$ time if using sparse matrices.

Finally, for a given type t, the set of state pairs having the type t can be represented by the following boolean matrix:

$$\tau_{ij}(t) = \begin{cases} 1 & \text{if } \tau(i,j) = t \\ 0 & \text{otherwise} \end{cases}$$

The probability that the k^{th} symbol of σ causes $\mathcal{M}$ to make a transition of type t is then given by:

$$r_t(k) = \sum_{1 \le i,j \le f} P_{ij}(k)\tau_{ij}(t)$$

which can be calculated in $O(\frac{n}{p}f^2)$ time on a PRAM with $O(p)$ processors if the quantities $P_{ij}(k)$ are already known, or $O(\frac{ne}{p})$ time when using sparse matrices.

Therefore, the complexity of the algorithm for calculating the frequency distribution of a given type is $O(\frac{ne}{p})$ time on a PRAM with $O(p)$ processors ($p \ge 1$). Since the criterion in equation (3.3) can be applied in constant time in parallel for each value of k, it results that the exon-intron structure of a gene can be predicted in$O(\frac{ne}{p})$ time on a PRAM with $O(p)$ processors.

3.5 Discussion

The results obtained for the human and the yeast genes should be interpreted with care. In the first part of this section, we address the problem of finding the "best" NDFSA for a set of sample sequences; that is, the problem of finding the appropriate trade-off between the fit to data and the complexity of the model. In the second part, we relate the present approach to the other approaches using formal grammars, and Hidden Markov Models. Finally, in the last part, we discuss the advantages and pitfalls of using finite-state automata.

3.5.1 Effectiveness and Expressiveness

The results in Figure 3.6 indicates that the triplicity constraint, the open reading frame constraint and the specification of splice site consensus are all crucial for inferring gene structure, whereas the other constraints operating on genes, such as the length constraints, are simply prediction "fine-tuners". In general, when adding a new constraint to an existing NDFSA, one should always make sure that this addition results in an effective increase in prediction accuracy. Note that two given constraints may have a limited effectiveness when used separately, but be very effective when used together. In other words, the effectiveness of a constraint mainly depends on the other constraints that are used. For example, most constraints introduced in this chapter are effective only when they are complemented by the open reading frame constraint.

Also, when an ordered list of NDFSA needs to be defined, it is desirable that the first NDFSA that recognizes a given input gene be also the one that maximizes parsing accuracy. However, in practice, it may be the case that a sequence is incorrectly parsed by a given NDFSA, but is correctly parsed by the next, more general NDFSA in the list. In other words, the parsing accuracy of a given NDFSA is not necessarily related to its specificity, nor is it related to its sensitivity. Note that, given a finite set of sequences S, one can always design a finite-state automaton that parses correctly every element in S. However, if the accuracy of an NDFSA in parsing sample input sequences is high, it may not be worth adding more states just to make it acceptable to a new sample sequence. This is because the *expressiveness* of the NDFSA, as defined below, may be reduced.

Ideally, each NDFSA in Figure 3.5 should be constructed to parse correctly the largest number of sequences with the minimum number of transitions. The expressiveness of the NDFSA can be measured by the pair $E = (AC, t)$, where AC if the approximate correlation and t is the number of transitions in the NDFSA. Given an approximate correlation close to 1, a large value of t means that many transitions are needed to accurately define a valid parse. A smaller value of t indicates an "optimized" NDFSA which contains fewer transitions and is therefore more expressive. The NDFSA for the yeast genes has expressiveness $E_{yeast} = (0.996, 375)$; the NDFSA F_3 has expressiveness $E_{human} = (0.737, 21373)$ when used in isolation.

A question then arises: given an NDFSA, can one design another NDFSA that would improve the expressiveness obtained? In other words, are the NDFSA obtained for the yeast genes or human genes the most probable models for the recognition of exons and introns? This is known to be a difficult problem in general (Cheeseman, 1990). In defining an NDFSA, one has to be careful that an increase in the number of transitions t results in improved prediction, i.e., an increase in the value of AC. On the other hand, one should not overfit the data by defining NDFSA that are too specific. The solution that we propose in this thesis to increase prediction accuracy is to define a hierarchy of NDFSA such that each level in the hierarchy corresponds to a larger set of constraints than the previous one.

3.5.2 Relationship with Other Approaches

Former attempts to identify splice sites by finite-state automata had limited success. The main reason is because these automata only described the patterns occurring at the splice sites (Kudo et al., 1987), with no special concern about the overall gene structure. In contrast, the NDFSA introduced in this chapter describe complete gene structures. As in (Brendel and Busse, 1984), these NDFSA describe the various components of genes but, in addition, they take into account the non-determinism introduced by introns. It should be noted that in (Searls, 1994), non determinism is also used to express parsing ambiguities, but a different approach relying on stochastic context-free grammars is used to predict gene structure. In this approach, the exhaustive search over the combinatorial space of gene assemblies is avoided by considering only the best 100 gene structures. In contrast, our method considers all possible gene assemblies.

Statistically based methods such as Hidden Markov Models (HMMs) have been fairly successful in parsing genes (Kulp et al., 1996; Henderson et al., 1997). One can view those algorithms as the unfolding of an NDFSA, the transitions of which are later labeled by probabilities. In these methods, dynamic programming is often used to determine "the most likely path in the DAG" satisfying a given property. The approach proposed in this thesis differs substantially from HMMs or other linguistics-based methods in the following aspects:

1. It has a simple intuitive interpretation: interpreting the results of prediction is possible because the model is clearly defined; no hidden or adjustable weights needs to be computed.

2. It is sound: all gene structures are considered to infer the final result, not just one.

3. Its time complexity is linear in the length of the input sequence: for a fixed NDFSA, the number of transitions e in the expression $O(ne)$ can be considered as a constant, since the only parameter that varies when performing gene structure prediction is the length of the sequence being analyzed.

4. It is structural: instead of determining the probabilities of every transition in the NDFSA from statistical information (as in HMMs), the NDFSA is used to assign probabilities to every symbol in the input sequence; in other words, a refinement of the model consists of introducing new states in the NDFSA instead of simply modifying data containing probabilities.

Unlike other programs, prediction accuracy seems to be higher in $G+C$ poor sequences (see Figure 3.12). However, like other programs, performance tends to decrease with sequence length (Figure 3.13). This is because the number of putative splice sites increases with sequence length. Thus, the number of possible gene structures increases too, and, as previously mentioned in Section 3.4, the accuracy of the prediction decreases.

3.5.3 Final Remarks about using NDFSA

An important step before testing more sequences is the refinement of the existing NDFSA. We have seen in Section 3.3 that splice site consensus contribute a lot to parsing accuracy. Therefore, it is very likely that prediction accuracy will be even higher when more information about the patterns occurring at the branch sites become available (currently, branch points are not annotated in the gene databases because their localization requires additional, costly experiments). It would also be interesting to see how prediction accuracy varies when longer patterns are used to describe splice site signals. As mentioned earlier, the problem that may arise when considering longer patterns is the overfit of the data.

GC Group	$GC\%$	AC	CC
1	0.38	0.92	0.91
2	0.40	0.95	0.95
3	0.44	0.85	0.84
4	0.48	0.85	0.85
5	0.50	0.80	0.78
6	0.53	0.70	0.68
7	0.56	0.73	0.72
8	0.59	0.77	0.77
9	0.62	0.73	0.73
10	0.67	0.71	0.71

Figure 3.12: Accuracy versus $G+C$ content. The 176 human sequences of the test set were ordered by increasing $G+C$ content and divided into 10 groups, each of which contained 18 sequences except for the last group which contained 14 sequences. The GC ratios and the values of AC and CC were averaged separately for each group.

Length Group	Length Range	Number of Parses	$GC\%$	AC	CC
1	467-996	2	0.58	0.90	0.90
2	1103-1424	2	0.49	0.91	0.91
3	1424-1486	11	0.47	0.88	0.88
4	1487-2086	11	0.56	0.86	0.86
5	2144-2625	11	0.50	0.86	0.86
6	2628-3148	150	0.57	0.61	0.59
7	3180-3947	62	0.49	0.78	0.78
8	4110-5264	168	0.50	0.73	0.72
9	5568-9014	804	0.48	0.78	0.77
10	9023-32343	18348	0.50	0.68	0.66

Figure 3.13: Accuracy versus sequence length. The 176 human sequences of the test set were ordered by increasing length and divided into 10 groups, each of which contained 18 sequences except for the last group which contained 14 sequences. The GC ratios, the number of parses, and the values of AC and CC were averaged separately for each group. In addition, the number of parses were rounded to the closest integer. Also indicated are the length of the shortest and the longest sequence in each group.

The use of learning techniques may ease the process of refining the NDFSA, although previous attempts to infer NDFSA from finite sets of patterns had limited success (Kudo et al., 1987). New algorithms have been recently proposed for NDFSA learning (Brave, 1996; Brāzma and Čerāns, 1997), but they perform poorly in capturing even very simple constraints such as triplicity. In addition, these programs are not designed for problems like gene structure prediction in which there are only positive examples (each known human gene is one). Negative examples, i.e., sequences that do not represent correct (human) alternations of exons and introns, are difficult to construct because there is no complete definition of what a correct or incorrect gene is. Thus, the use of learning techniques might have to be postponed until all human genes are known.

Although the NDFSA designed for the yeast and human genes yield good results for the prediction of exons and introns, it may be necessary in subsequent developments to consider more general grammars. On one hand, there is evidence that the language of genes is beyond context-free (Collado-Vides, 1991a; Searls, 1993). On the other hand, the use of more general grammars should be avoided unless efficient parsers can be designed. Indeed, it is well known that the parsing of strings in languages described by context-free grammars (CFGs) take $O(n^3)$ time, where n is the length of the string to be parsed (Harrison, 1978). That complexity renders impossible the parsing of potentially ambiguous strings containing (tens of) thousands of symbols, even using massively parallel computers. Note that it is not always necessary to write a grammar-based parser to test if a given string belongs to a context-free language. For example, testing if a given sequence is a palindrome can be done without defining a grammar.

Also note that even though CFGs can define languages that are not describable by Finite-State Grammars (FSGs), whenever finite strings are considered, there are FSGs that mimic the corresponding CFGs. However, it is important to keep in mind that the resulting FSGs may have a very large number of states, as each state represents a possible configuration of a finite stack. New algorithms have been recently developed to compute finite-state approximations of context-free grammars (Rood, 1996; Bertsch and Nederhof, 1996; Pereira and Wright, 1997). Therefore, it may well be possible to avoid the use of CFGs.

Finally note that with the present approach, the number of states and transitions of the NDFSA is expected to increase very quickly as more constraints are added to the definition of gene models. This is because finite-state automata are limited in their expressive power. For example, it is known that most human proteins consist of a hundred or more amino acids, thus it would be useful to have a constraint that places a lower limit on the number of codons encoded by a gene. Although it is not difficult to design an NDFSA that recognizes every gene containing more than a given number of codons — the NDFSA describing a codon simply needs to be repeated the desired number of times — the number of states and transitions of this NDFSA may be very large. Therefore, it is important to find a better way to represent constraints.

In the above-mentioned example, the explosion in the size of the automaton can easily be avoided by using a variable that keeps track of the number of codons in the sequence being analyzed. The use of variables, or more exactly, the use of "labels" to reduce the size of the data structure is at the core of the DAG representation introduced in the next chapter.

Chapter 4

DAG Representation of Gene Models

The use of non deterministic finite-state automata for encoding various constraints which operate on gene structure provided a sound theoretical basis for a new algorithm that detects exons and introns in genes. Even though NDFSA can describe a variety of constraints, the number of states and transitions required to encode certain properties may be prohibitive both spacewise and timewise.

The aim of this chapter is to define a more compact data structure for representing gene models. This is achieved by partitioning the set of constraints introduced in the previous chapter into three different classes:

- The *a priori* constraints, which define the splice sites in the sequence σ being analyzed.
- The *intrinsic* constraints, which define the valid combinations of exons and introns in σ.
- The *a posteriori* constraints, which are heuristics that select one (or possibly a few) among the many alternations of exons and introns satisfying the *a priori* and *intrinsic* constraints.

Gene structure prediction is then performed in three steps, each of which corresponds to the application of a given type of constraints. In this chapter, it will be seen that the main advantage of partitioning the constraints into three different classes is that it is possible to represent the gene models satisfying the *a* *priori* and intrinsic constraints by labeled DAGs which are significantly smaller than the DAG introduced in the previous chapter (note that in the proposed approach there is no need to use a DAG to represent the few gene models that satisfy the *a posteriori* constraints). In addition, it will be shown that the formalism introduced in this chapter can be used to: (a) incorporate coding potential information into the DAG, and (b) reduce the complexity of the gene structure prediction algorithms that use dynamic programming. A comparison with other DAG-based approaches will be provided, and the pros and cons of the proposed approach versus the one using NDFSA will be discussed from both the algorithmic and biological point of views.

4.1 Overview of the DAG-based Approach

Like the approach introduced in the previous chapter, the key idea is to predict gene structure by constructing a DAG in which the paths between the source and the sink represent all possible alternations of exons and introns in a given gene. However, this new DAG is not generated by unfolding a finite-state automaton on a given input sequence, but by considering three screening classes of constraints which define the inter-relationships between splice sites, exons and introns.

The constraints belonging to the first class are called *a priori* constraints. They define the putative splice sites (or putative exons and introns) in the sequence σ to be analyzed. Examples of *a priori* constraints are the $GT - AG$ rule and the length constraints. The parses of σ that satisfy the *a priori* constraints are represented by a first DAG denoted $\Delta(\sigma)$. The nodes and the edges of this DAG respectively represent the putative splice sites and the potential exon and intron fragments of σ. Every path starting on the source of $\Delta(\sigma)$ and ending on its sink corresponds to a parse of σ satisfying the *a priori* constraints.

The second set of constraints attempts to minimize the possibly large number of paths existing between the source and the sink of $\Delta(\sigma)$. Unlike the *a priori* constraints which define the nodes and edges of $\Delta(\sigma)$, these constraints operate on entire paths. They are called *intrinsic* constraints. Both the triplicity and open reading frame constraints are intrinsic. Intuitively, a constraint $\mathcal{C}$ is intrinsic if it expresses a property that the paths in $\Delta(\sigma)$ must satisfy to represent valid gene models. The paths in $\Delta(\sigma)$ that satisfy $\mathcal{C}$ then form a new DAG denoted $\Delta_{\mathcal{C}}(\sigma)$. Therefore, the paths in $\Delta_{\mathcal{C}}(\sigma)$ represent the parses of σ that satisfy not only $\mathcal{C}$ but also the *a priori* constraints.

Since the number of paths in $\Delta_{\mathcal{C}}(\sigma)$ may still be very large, a final set of general constraints, the *a posteriori* constraints, is used to screen out most of these paths. *A posteriori* constraints are essentially heuristics that select only the path — or the few paths — representing the best gene models. An example of *a posteriori* constraint is the criterion used in the previous chapter for distinguishing exons from introns.

In this chapter, we develop a formalism for *a priori* constraints (Section 4.2) and intrinsic constraints (section 4.3). The purpose of this formalization is to derive an efficient algorithm to construct the DAGs $\Delta(\sigma)$ and $\Delta_{\mathcal{C}}(\sigma)$. Note that the class of *a posteriori* constraints is too general to be captured by an efficient algorithmic model. The *a priori* constraints are defined using a number of boolean variables whose values indicate where the exon and intron fragments are in the sequence being analyzed (Section 4.2.1). These boolean variables are used in Section 4.2.2 to define the initial DAG $\Delta(\sigma)$. They are also used as the input of the sequential algorithm that constructs $\Delta(\sigma)$ from σ (Section 4.2.3).

The intrinsic constraints are defined by introducing the concepts of *decoration*, *summary* and *decoration-summary system*. Essentially, one can simply imagine that every edge in the DAG $\Delta(\sigma)$ is labeled (decorated) with some quantity. We then define two operators: the first operator, denoted $\odot$, takes as operands the labels of two consecutive paths, and returns the label of the path resulting from their concatenation (Section 4.3.1). For example, if the labels are defined as numbers representing the lengths of the paths, then the $\odot$ operator can be defined as the addition of two numbers. The second operator, denoted $\oplus$, defines a summary or choice between the labels of two or more paths having common initial and ending nodes (Section 4.3.2). An example of $\oplus$ operator is the function *min* which given two or more paths yields the paths with minimum (combined) label. The definition of the $\oplus$ operator is such that the summary of a set of labels may contain more than one label. For example, this property allows one to select the B shortest paths ($B \geq 1$) rather than just the shortest path(s).

The relations that link the two operators $\odot$ and $\oplus$ are formulated through the concept of *decoration-summary system* or *DS-system* (Section 4.3.3). The particular properties that a DS-system must satisfy allows us to derive an efficient algorithm which can select the paths in the DAG whose label is in a given subset. For example, one can define a DS system that prunes every path between the source and the sink whose label is not minimal by taking $\odot = +$ and $\oplus = min$. A constraint $\mathcal{C}$ is then said to be intrinsic whenever it is possible to find a DS-system and a set L^0 — that can be seen as a path selector — such that there is a one-to-one mapping between the genes satisfying $\mathcal{C}$ and the paths labeled with an element of L^0 (Section 4.3.4). In this chapter, we show that the DAG $\Delta_{\mathcal{C}}(\sigma)$, which only contains these particular paths, can be constructed efficiently from $\Delta(\sigma)$ (Section 4.3.5), and we provide a scheme for composing two or more intrinsic constraints (Section 4.3.6).

In Section 4.4, we give a formal proof that the triplicity and open reading frame constraints are intrinsic (Sections 4.4.1 and 4.4.2). We also show that, given a sequence σ to be parsed, the DAG that represents the gene models satisfying the triplicity and open reading frame constraints is significantly smaller than the DAG introduced in the previous chapter. We also define a new constraint, the *cardinality constraint*, which places an upper limit on the number of exons and/or introns that can occur in genes (Section 4.4.3). Using the cardinality constraint, we provide a striking example of the reduction in size obtained when using the new representation. We also provide an example of composition involving the triplicity and cardinality constraint (Section 4.4.4).

The *a priori* and intrinsic constraints being formalized, we then focus our attention on the *a posteriori* constraints (Section 4.5). In particular, we show how the criterion used earlier for distinguishing exons from introns can be reformulated in the context of DS-systems. Finally, we conclude this chapter by comparing the new proposed approach with the other existing approaches (Section 4.6) and the approach using NDFSA (Section 4.7).

4.2 *A priori* Constraints

The *a priori* constraints designate the potential exon and intron fragments in a given DNA sequence. In this section, we start by defining formally the notion of gene fragments, viewing a gene as a sequence of *partial* exons and introns rather than *entire* exons and introns. Given a sequence σ to be analyzed, we show that we can represent by an initial DAG $\Delta(\sigma)$ the set of parses of σ that satisfy the *a priori* constraints, and we provide two algoritms for constructing $\Delta(\sigma)$: one takes as input the potential exon and intron fragments and runs in $O(T + n^2)$ time, where T is the time taken to evaluate the *a priori* constraints and n is the length of σ; the other takes as input the positions of the putative splice sites and run in $O(T + n)$ time.

4.2.1 Identification of the Potential Exon and Intron Fragments

Given a DNA sequence σ of length n starting with a start codon (atg) and ending with a stop codon (taa, tag or tga), let us define the two following boolean functions: the first, $L(i)$, is $true$ if and only if there may be a potential intron in σ starting at position i; the other, $R(i)$, is $true$ if and only if there may be a potential exon in σ starting at position i. Note in particular that $R(1) = true$. To simplify the proposed construction, we assume the existence of two virtual nucleotides used as sentinels: one at position 0 and the other at position $n + 1$; we define $L(n + 1)$ as $true$ and $L(0)$, $R(0)$ and $R(n + 1)$ as $false$. The following set:

$$\mathcal{S}(\sigma) = \{i \mid L(i) \ \vee \ R(i) = true, \ 0 \le i \le n + 1\}$$

then contains the potential positions where either an intron or exon may start.

For example, if the $GT - AG$ rule is used to determine the potential splice sites, then for a sequence of length n, the variables $L(i)$ and $R(i)$, for $0 \le i \le n+1$, are defined as follows:

$$\begin{cases} L(0) = false, R(0) = false, R(1) = true \\ L(n) = false, L(n+1) = true, R(n+1) = false \\ \forall j\ 1 \le j < n: \ L(j) \leftrightarrow (\sigma_j = g) \ \wedge \ (\sigma_{j+1} = t), \ R(j+1) \leftrightarrow (\sigma_{j-1} = a) \ \wedge \ (\sigma_j = g) \end{cases}$$

Although it is possible to determine from the values of the funtions L and R where the potential exons and introns are in σ, it is often simpler to define these directly. In what follows, the potential exon and introns are specified by distinguishing four types of sequence fragments:

- An lr-sequence is a sequence fragment starting on a potential donor splice site and ending on a potential acceptor splice site.
- An rr-sequence is a sequence fragment between two potential acceptor splice sites.
- an rl-sequence is a sequence fragment starting on a potential acceptor splice site and ending on a potential donor splice site.
- An ll-sequence is a sequence fragment between two potential donor splice sites.

In addition, we require that no fragment of a given type may be the prefix of another fragment of the same type.

Formally, given the four boolean functions defined as follows:

- $lr(i,j) \leftrightarrow L(i) \ \wedge \ R(j+1) \ \wedge \ \forall k, i \le k < j, \neg lr(i,k)$
- $rr(i,j) \leftrightarrow R(i) \ \wedge \ R(j+1) \ \wedge \ \forall k, i \le k < j, \neg rr(i,k)$
- $rl(i,j) \leftrightarrow R(i) \ \wedge \ L(j+1) \ \wedge \ \forall k, i \le k < j, \neg rl(i,k)$
- $ll(i,j) \leftrightarrow L(i) \ \wedge \ L(j+1) \ \wedge \ \forall k, i \le k < j, \neg ll(i,k)$

the sequence fragment $\sigma_{i,j}$ between and including positions i and j, for $1 \leq i \leq j \leq n$, is said to be an *lr-sequence*, *rr-sequence*, *rl-sequence* or *ll-sequence* if and only if the value of the corresponding boolean function lr, rr, rl, or ll is $true$ for the boundaries i and j.

For example, the $GT - AG$ rule can be formulated by defining the exon and intron fragments as follows (using regular expressions) [1]:

- $lr(i,j) \leftrightarrow \sigma_{i,j} = G.T.\overline{A.G}.A.G$
- $rr(i,j) \leftrightarrow \sigma_{i-2,i-1} = A.G \ \wedge \ \sigma_{i,j} = \overline{A.G}.A.G$
- $rl(i,j) \leftrightarrow (\sigma_{i-2,i-1} = A.G \ \vee \ i = 1) \ \wedge \ \sigma_{i,j} = \overline{G.T} \ \wedge \ (\sigma_{j+1,j+2} = G.T \ \vee \ j = n)$
- $ll(i,j) \leftrightarrow \sigma_{i,j} = G.T.\overline{G.T} \ \wedge \ (\sigma_{j+1,j+2} = G.T \ \vee \ j = n)$

where:

- $\overline{A.G} = (A.(A + C + T) + G + C + T)^{\star} + A$
- $\overline{G.T} = (G.(A + C + G) + A + C + T)^{\star} + G.$

Figure 4.1 provides the values of the above variables for a sequence σ of length $n = 38$. For this sequence, the set $\mathcal{S}(\sigma)$ contains nine elements which represent the potential boundaries between the coding and non-coding regions of σ: $\mathcal{S}(\sigma) = \{1, 6, 13, 14, 20, 21, 28, 35, 39\}$.

The boolean functions lr, rr, rl and ll are particularly useful when expressing constraints that operate on sequence fragments rather than individual splice sites. The length constraints introduced in Section 3.3 are typical examples of such constraints. Recall that the length constraints require that the lengths of exons and introns be larger than some given constants M_E and M_I. Now consider the two following boolean funtions:

$$Ex(i,j) \leftrightarrow ll(i,j) \ \vee \ rl(i,j) \qquad In(i,j) \leftrightarrow lr(i,j) \ \vee \ rr(i,j)$$

[1] The definitions of rl and ll have been simplified for clarity. The special case in which a G occurs just before the stop codon (thus creating the sequence GT) should be added to the definitions of rl and ll. In addition, the case in which a T occurs after the start codon should also be added to the definition of rl.

i,j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
σ		a	t	g	c	t	g	t	g	g	c	a	g	g	g	t	c	c	c	a	g	t	c	t	g	a	c	c	g	t	c	g	c	a	g	a	t	a	a	
$L(i)$	f	f	f	f	f	f	t	f	f	f	f	f	f	f	t	f	f	f	f	f	t	f	f	f	f	f	f	f	t	f	f	f	f	f	f	f	f	f	f	t
$R(i)$	f	t	f	f	f	f	f	f	f	f	f	f	f	t	f	f	f	f	f	f	f	t	f	f	f	f	f	f	f	f	f	f	f	f	f	t	f	f	f	f
$lr(i,j)$							<	−	−	−	−	−	>		<	−	−	−	−	−	>								<	−	−	−	−	−	>					
																					<	−	−	−	−	−	−	−	−	−	−	−	−	−	>					
$rr(i,j)$														<	−	−	−	−	−	−	>	<	−	−	−	−	−	−	−	−	−	−	−	−	>					
$rl(i,j)$		<	−	−	−	>								<>								<	−	−	−	−	−	>								<	−	−	>	
$ll(i,j)$							<	−	−	−	−	−	−	>	<	−	−	−	−	>	<	−	−	−	−	−	−	>	<	−	−	−	−	−	−	−	−	−	>	

Figure 4.1: Definition of the boolean variables L, R, lr, rr, rl, and ll for the $GT-AG$ rule. The value of $L(i)$ is $true$ if and only if a putative intron may start at position i. Similarly, the value of $R(i)$ is $true$ if and only if a putative exon may start at position i. The values of $lr(i,j)$, $rr(i,j)$, $rl(i,j)$, and $ll(i,j)$ are true, respectively, if the sequence fragment between and including the nucleotides at position i and j is a lr-sequence, rr-sequence, rl-sequence, and ll-sequence.

Then the sequence between and including the i^{th} and j^{th} nucleotide is a potential exon fragment if and only if $Ex(i,j) = true$, and it is a potential intron fragment if and only if $In(i,j) = true$. Assuming that the functions lr, rr, rl and ll are defined, the length constraints can then be expressed as follows:

$$Ex(i,j) \leftrightarrow (ll(i,j) \ \vee \ rl(i,j)) \ \wedge \ (j - i > M_E) \qquad In(i,j) \leftrightarrow (lr(i,j) \ \vee \ rr(i,j)) \ \wedge \ (j - i > M_I)$$

We now show that the parses of σ that satisfy a given set of *a priori* constraints as defined by the above boolean variables can be represented by a DAG denoted $\Delta(\sigma)$.

4.2.2 Construction of the initial unlabeled DAG $\Delta(\sigma)$

The DAG $\Delta(\sigma)$ is obtained as follows. Every node in $\Delta(\sigma)$ corresponds to an element of $\mathcal{S}$ so that each node is referred to by the element of $\mathcal{S}$ it represents. There are two types of nodes (see Figure 4.2):

- *left* nodes, which represent the potential donor splice sites (except the sink node which represents the stop codon): node $i \in \mathcal{S}$ is a left node if $L(i) = true$,
- and *right* nodes, which represent the potential acceptor splice sites (except the source node which represents the start codon): node $i \in \mathcal{S}$ is a right node if $R(i) = true$.

There exists an edge between two nodes i and j if and only if the sequence fragment between and including positions i and $j-1$ is an exon or intron fragment, i.e., $Ex(i, j-1) = true$ or $In(i, j-1) = true$.

Therefore, the DAG $\Delta(\sigma)$ contains four kinds of edges: lr-edges, rr-edges, rl-edges and ll-edges. For every integer i and j, $1 \le i < j \le n+1$, the edge $i \to j$ linking nodes i and j is a *lr-edge*, *rr-edge*, *rl-edge* or *ll-edge* if and only if its corresponding boolean variable $lr(i,j-1)$, $rr(i,j-1)$, $rl(i,j-1)$ or $ll(i,j-1)$ is $true$. In other words, an lr-edge represents an lr-sequence; an rr-edge an rr-sequence; an rl-edge an rl-sequence; and an ll-edge an ll-sequence. The paths of interest are the full paths of $\Delta(\sigma)$, i.e., the paths starting on the source node and ending on the sink node of $\Delta(\sigma)$, as they represent complete gene models. A path in $\Delta(\sigma)$ passing through the nodes $i_0, i_1, \ldots, i_{m-1}, i_m$ ($m \ge 1$) will be denoted

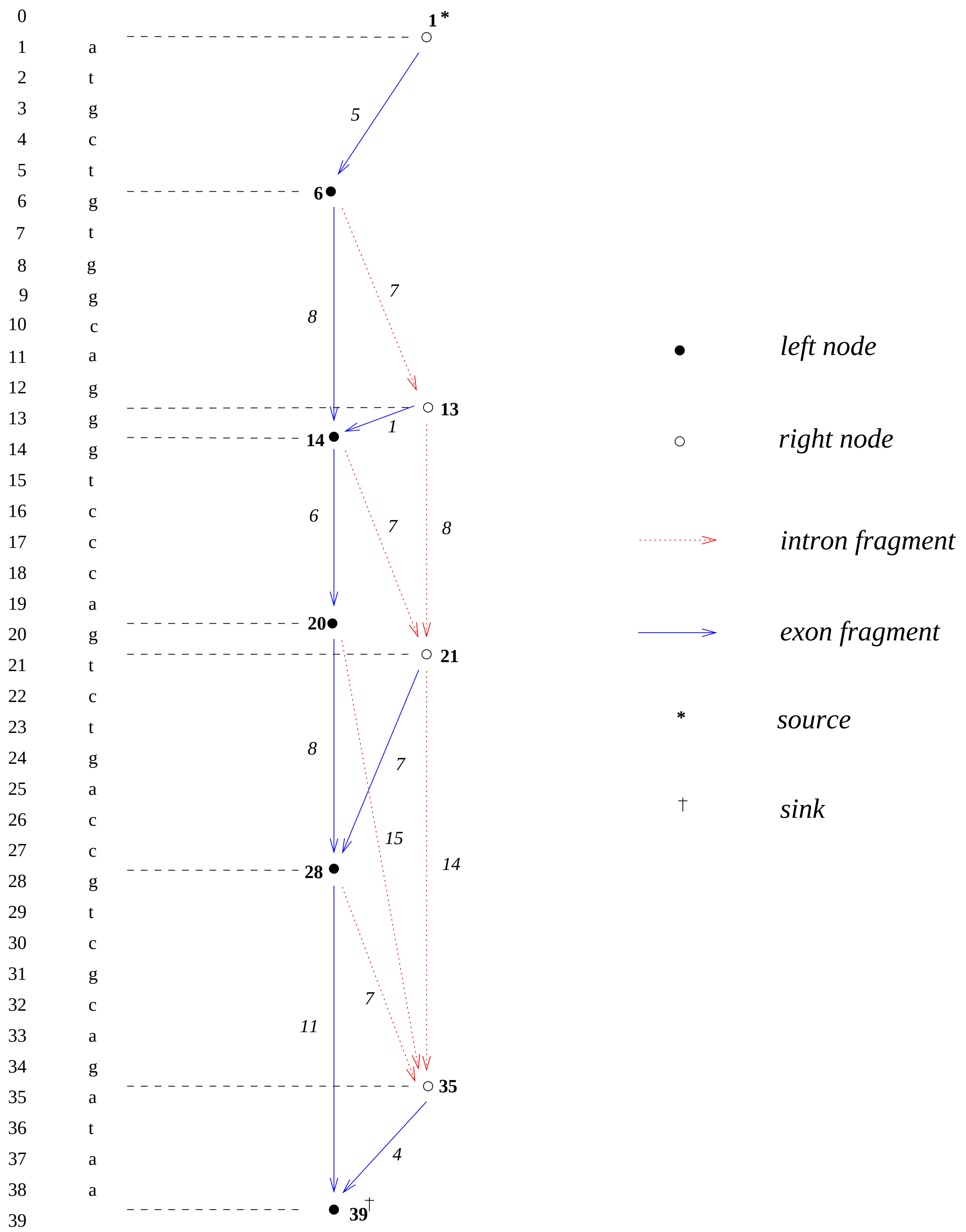


Figure 4.2: DAG $\Delta(\sigma)$ for the sequence σ in Figure 4.1. For clarity, every edge in $\Delta(\sigma)$ is labeled with the length of the gene fragment it represents.

$i_0 \to i_1 \to \ldots \to i_{m-1} \to i_m$. It is easily seen that every full path $i_0 \to i_1 \to \ldots \to i_{m-1} \to i_m$ of $\Delta(\sigma)$ represents an alternation of exon and intron fragments, i.e.:

- $i_0 = 1$, $Ex(i_0, i_1 - 1)$.
- $\forall k\ 1 \leq k < m-1 :\ In(i_k, i_{k+1} - 1) \vee Ex(i_k, i_{k+1} - 1)$.
- $i_m = n+1$, $Ex(i_{m-1}, i_m - 1)$.

If the sequence σ contains $s-1$ potential splice sites ($s \geq 1$), then the DAG $\Delta(\sigma)$ contains $s+1$ nodes since two extra nodes are added to represent the start codon and the stop codon. The number of edges in $\Delta(\sigma)$ is at most $2(s-1)$: the source has one outgoing edge; the node corresponding to the $(s-1)$-th splice site has at most one outgoing edge; and the other nodes have at most two outgoing edges. More precisely, since there is exactly one *ll*-edge or *rl*-edge coming out of each node but the sink, there are exactly s edges of type ll or rl in $\Delta(\sigma)$, and at most $s-2$ edges of type lr or rr. Since the number of splice sites cannot exceed the number of bases in σ, the number of edges in $\Delta(\sigma)$ is at most $2(n-1)$ and is therefore linear in the length of σ.

4.2.3 Efficient Algorithm for Construction $\Delta(\sigma)$

We now provide two sequential algorithms for constructing $\Delta(\sigma)$ from σ. The first algorithm takes as input the positions of the potential exon and intron fragments as given by the functions Ex and In. It runs in quadratic time. The second algorithm takes as input the positions of the putative splice sites as given by the functions L and R, and constructs a DAG which represents all parses of σ satisfying the length constraints. It runs in linear time.

It has been seen in the previous section that every node i in $\Delta(\sigma)$ has a left successor l if and only if $Ex(i, l-1) = true$ and a right successor r if and only if $In(i, r-1) = true$. Therefore, the DAG $\Delta(\sigma)$ can be constructed by simply scanning the values of $Ex(i,j)$ and $In(i,j)$, for $1 \leq i \leq j \leq n$, and creating a left or right successor $j+1$ for every node i such that $Ex(i,j)$ or $In(i,j)$ is $true$, respectively.

```
begin
 for i ← 1 to n do
    for j ← i to n do
       if Ex(i,j) = true then Nextleft[i] ← j+1 else Nextleft[i] ← ⊥
       if In(i,j) = true then Nextright[i] ← j+1 else Nextright[i] ← ⊥
end
```

Figure 4.3: A general algorithm for constructing $\Delta(\sigma)$ based on the values of the functions Ex and In. The left and right successors of every node i are stored in the arrays $Nextleft$ and $Nextright$, respectively.

The resulting algorithm has $O(n^2)$ time complexity (see Figure 4.3). Note that this complexity does not take into account the time taken for determining the values of the functions $Ex(i,j)$ and $In(i,j)$.

We now show how the complexity of this algorithm can be decreased to linear time when the length constraints hold and the putative exons and introns are specified by their edges (i.e., by the functions L and R). Let M_E and M_I be the respective minimum lengths that an exon or intron can have ($M_E \geq 1$, $M_I \geq 1$), and NL and NR the functions defined as follows:

$$NL(i) = \begin{cases} \perp & \text{if } i > n - M_E \\ n+1 & \text{if } i = n - M_E \\ i + M_E & \text{if } i < n - M_E \text{ and } L(i+M_E) = true \\ NL(i+1) & \text{otherwise} \end{cases}$$

$$NR(i) = \begin{cases} \perp & \text{if } i \geq n - M_I \\ i + M_I & \text{if } i < n - M_I \text{ and } R(i+M_I) = true \\ NR(i+1) & \text{otherwise} \end{cases}$$

Then, for every node i such that $L(i) = true$, the left successor of i is $NL(i)$, and for every node i such

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
σ	a	t	g	c	t	g	t	g	g	c	a	g	g	g	t	c	c	c	a	g	t	c	t	g	a	c	c	g	t	c	g	c	a	g	a	t	a	a	
$L(i)$	f	f	f	f	f	t	f	f	f	f	f	f	f	t	f	f	f	f	f	t	f	f	f	f	f	f	f	t	f	f	f	f	f	f	f	f	f	f	t
$R(i)$	t	f	f	f	f	f	f	f	f	f	f	f	t	f	f	f	f	f	f	f	t	f	f	f	f	f	f	f	f	f	f	f	f	f	t	f	f	f	f
$NL(i)$	6	6	6	6	6	14	14	14	14	14	14	14	14	20	20	20	20	20	20	28	28	28	28	28	28	28	28	39	39	39	39	39	39	39	39	39	39	$\perp$	$\perp$
$NR(i)$	13	13	13	21	21	21	21	21	21	21	21	35	35	35	35	35	35	35	35	35	35	35	35	35	35	35	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Figure 4.4: Values of NL and NR for the sequence in Figure 4.1 ($M_E = 1$ and $M_I = 10$).

that $R(i) = true$, the right successor of i is $NR(i)$. For example, for $M_E = 1$ and $M_I = 10$, one has to replace in the DAG of Figure 4.2 edges $6 \xrightarrow{7} 13$ and $14 \xrightarrow{7} 21$ with edges $6 \xrightarrow{15} 21$ and $14 \xrightarrow{21} 35$, and remove edge $28 \xrightarrow{7} 35$ (Figure 4.4 provides the corresponding values of NL and NR).

The corresponding algorithm is provided in Figure 4.5. The time complexity of this algorithm is $O(n)$. Assuming the existence of a preprocessing phase that computes the functions L and R in $O(T)$ time, the algorithm for constructing $\Delta(\sigma)$ from σ takes, then, $O(T + n)$ time. In particular, if the potential splice sites are defined by two sets of consensus patterns (as was the case for human genes), one denoted $\mathcal{D}$ corresponding to the donor splice site consensus and the other denoted $\mathcal{A}$ corresponding to the acceptor splice site consensus, then the problem of determining the functions L and R is reduced to the *exact set matching* problem.

The exact set matching problem is to find all occurences in σ of any pattern in a finite set of patterns $\mathcal{P}$. It can be solved in $O(t + n + k)$ time, where t is the total length of all the patterns in $\mathcal{P}$, n the length of σ, and k the number of occurences in σ of the patterns from $\mathcal{P}$ (Gusfield, 1997). Let us assume that t represents the total length of all the possible patterns (i.e., the consensus) which can occur at the splice sites. An NDFSA recognizing all these patterns would contain $O(t)$ transitions in the worst case, and the time taken to obtain all gene models using this NDFSA would be $O(tn)$. In contrast, the time taken for finding all potential splice sites — and therefore all gene models — using the algorithm

begin

$NL \leftarrow n+1$ $\{$*Computation of the left successors*$\}$

for $i \leftarrow n+1$ **downto** $n - M_E + 1$ **do** $Nextleft[i] \leftarrow \perp$

for $i \leftarrow n - M_E$ **downto** 1 **do**

  if $L(i) = true$ **then** $Nextleft[i] \leftarrow NL$ **else** $Nextleft[i] \leftarrow \perp$

  if $L(i + M_E) = true$ **then** $NL \leftarrow i + M_E$

$NR \leftarrow \perp$ $\{$*Computation of the right successors*$\}$

for $i \leftarrow n+1$ **downto** $n - M_I + 1$ **do** $Nextright[i] \leftarrow \perp$

for $i \leftarrow n - M_I$ **downto** 1 **do**

  if $R(i) = true$ **then** $Nextright[i] \leftarrow NR$ **else** $Nextright[i] \leftarrow \perp$

  if $R(i + M_I) = true$ **then** $NR \leftarrow i + M_I$

end

Figure 4.5: A linear algorithm for constructing the DAG $\Delta(\sigma)$ representing all the gene models satisfying the length constraints.

for the set matching problem is $O(t+n+k)$. Since, in our approach, all patterns are distinct and have the same length, we have: $k \leq n$, so that the latter complexity is actually $O(t+n)$. Note that the DAG $\Delta(\sigma)$ in this case contains at most $2k$ edges whereas the DAG generated by the NDFSA contains at most αk edges, where α is the common length of the consensus. Since, in practice, we have: $\alpha > 2$ (recall that $\alpha = 12$ for human genes), the DAG $\Delta(\sigma)$ is smaller than the NDFSA-generated DAG by an order of magnitude.

Note that unless the *a priori* constraints are very selective, it is very likely that the DAG $\Delta(\sigma)$ will contain a large number of paths between its source and its sink. The aim of the next screening set of constraints, i.e., the intrinsic constraints, is to reduce that number.

4.3 Intrinsic Constraints

In this section, we introduce the notion of *intrinsic* constraint. Informally stated, a constraint $\mathcal{C}$ is intrinsic if it expresses a property that the paths in $\Delta(\sigma)$ must satisfy to represent valid gene models. For example, assume that every *ll*-edge and *rl*-edge of $\Delta(\sigma)$ is labeled with the length modulo three of the sequence fragment it represents, and every *rl*-edge and *rr*-edge is labeled with zero. Also assume that the label of every path is defined as the sum (modulo three) of the labels of its edges. Then, every path between the source and the sink of $\Delta(\sigma)$ whose label is zero modulo three represents a gene model satisfying the triplicity constraint. We will say that the triplicity constraint is intrinsic.

It should be noted that the terminology and notations used in this section are very similar to those used in (Cormen et al., 1990), wherein a general framework is developed to solve problems involving directed graphs. Although the approach below is also based on directed graphs, it is perfected for the case in which these graphs are acyclic and for the particular problem of gene structure prediction. Our goal and approach are therefore radically different from those in (Cormen et al., 1990).

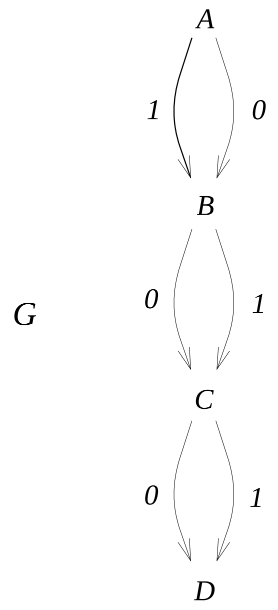
The definition of an intrinsic constraint relies on the concepts of *decoration*, *summary* and *decoration-summary system*:

- Given a graph G, a decoration labels every edge and every path in G. The purpose of defining a decoration is to associate with every edge and every path in G a semantic value related to the constraint to be applied. In the case of the DAG $\Delta(\sigma)$, the label of a path allows one to determine if this path represents a parse satisfying the desired intrinsic constraint. Therefore, a naive way to determine all parses of σ satisfying a given intrinsic constraint is to perform an exhaustive search in $\Delta(\sigma)$.

- Given a set S, a summary selects some of the elements in S according to some specified criterion. The purpose of defining a summary is to obtain a condensed representation of S that retains only the elements of S that are relevant to the constraint to be applied. For example, if one is interested in finding the shortest path in a weighted graph, then one can take as summary the function that selects the minimum of two weights. A summary can be thought of as the label of a set of paths. Thus, labels and summaries are both represented by the same object, i.e., sets. It will be seen that the most general summary is the set containing the label of every path.

- Given a graph G, a decoration-summary system or DS-system is the combination of a decoration and a summary. The decoration labels every edge and path in G, and the summary selects representative subsets of the labels of the paths in G. It will be seen that the above-mentioned exhaustive search can be avoided by summarizing the path labels that are identical.

To distinguish a path labeled with a given quantity from a path labeled with another, every node in G will be divided into a set of subnodes (see Figure 4.6 for a toy example). It will be seen that each subnode corresponds to the label that a set of paths have in common, i.e., an element belonging to the summary of the labels of these paths. Given the graph $G = \Delta(\sigma)$ introduced in the previous section, the set of gene models satisfying a given intrinsic constraint $\mathcal{C}$ will be then represented by the paths between the source and the sink of G that start on certain subnodes of the source. These particular subnodes will



Path	Label
$A \xrightarrow{1} B \xrightarrow{0} C \xrightarrow{0} D$	1
$A \xrightarrow{1} B \xrightarrow{0} C \xrightarrow{1} D$	2
$A \xrightarrow{1} B \xrightarrow{1} C \xrightarrow{0} D$	2
$A \xrightarrow{1} B \xrightarrow{1} C \xrightarrow{1} D$	3
$A \xrightarrow{0} B \xrightarrow{0} C \xrightarrow{0} D$	0
$A \xrightarrow{0} B \xrightarrow{0} C \xrightarrow{1} D$	1
$A \xrightarrow{0} B \xrightarrow{1} C \xrightarrow{0} D$	1
$A \xrightarrow{0} B \xrightarrow{1} C \xrightarrow{1} D$	2

Node	Summary
A	$\{0, 1, 2, 3\}$
B	$\{0, 1, 2\}$
C	$\{0, 1\}$
D	$\{0\}$

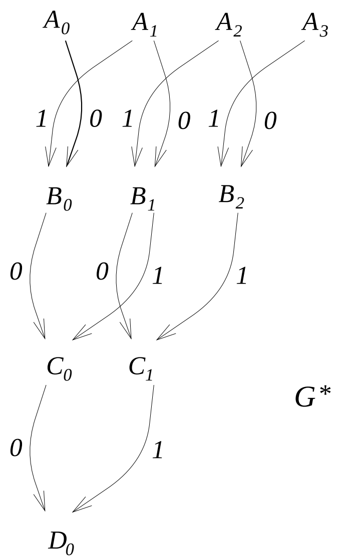


Figure 4.6: Example of decoration and summary for a given graph G. The label of each path in G is defined as the sum of the labels of its edges. For every node i in G, the summary at node i contains the labels of all paths between node i and node D. The graph $G^{\star}$ is obtained from G by subdividing every node in G into subnodes: each subnode of a node i in G corresponds to an element in the summary at node i. For example, node A is subdivided into four subnodes: A_0, A_1, A_2, and A_3 such that every path starting on subnode A_i $(0 \leq i \leq 3)$ and ending on subnode D_0 has i as label.

be specified by means of a set L^0, which will be defined such that there is a one-to-one correspondence between the set of gene models satisfying $\mathcal{C}$ and the set of paths between the source and the sink of G labeled with an element of L^0.

We now provide a more formal description of the various concepts introduced above. We start by providing very general definitions which can be applied to any acyclic directed graph G and then consider the special case of the DAG $\Delta(\sigma)$. We show that given an intrinsic constraint $\mathcal{C}$, it is possible to construct another DAG, denoted $\Delta_{\mathcal{C}}(\sigma)$, which contains only the paths labeled with an element of L^0, i.e., the paths representing the parses of σ satisfying $\mathcal{C}$. To simplify the notations, we renumber the nodes in $\Delta(\sigma)$ between 0 and s: node i represents the i^{th} potential splice site of σ (donor or acceptor), so that node 0 is the only source of $\Delta(\sigma)$ and node s its only sink.

4.3.1 DAG Decoration

Given a DAG G and a non-empty set L, an *L-decoration* Λ of G is a system that labels every edge and path in G with subsets of L. The label of a path p is computed by using a binary operator denoted $\odot$ that "combines" the labels of the edges in p. The labels can be defined to capture various characteristics of the sequences represented by the edges and paths, such as length, type (e.g., lr-sequence or not), existence of stop codons or coding potential.

Formally, let $G = (V, E)$ be *any* directed acyclic graph and L some non-empty set. We denote $\hat{L}$ the powerset of L, i.e., the set whose members are all the subsets of L. The triplet $\Lambda = (\odot, \bar{1}, \lambda)$ is said to be a *L-decoration* of G if it satisfies the following properties:

(H1) $\odot$ is a binary operator on $\hat{L}$, called the *decoration operator*.

(H2) $\bar{1}$ is an identity for $\odot$: $\forall X \in \hat{L}, X \odot \bar{1} = \bar{1} \odot X = X$.

(H3) $\odot$ is associative: $\forall X, Y, Z \in \hat{L}, X \odot (Y \odot Z) = (X \odot Y) \odot Z$.

(H4) λ is a function, called *decoration function*, which maps every path in G into a singleton:

(H4a) For every edge $i \to j$ in G, there exists an element $l \in L$ such that: $\lambda(i \to j) = \{l\}$.

(H4b) For every path $i_0 \to i_1 \to \cdots \to i_q$ passing through the nodes $i_0, i_1, \ldots, i_q$ of G, there exists an element $l \in L$ such that:

$$\lambda(i_0 \to i_1 \to \cdots \to i_q) = \lambda(i_0 \to i_1) \odot \lambda(i_1 \to i_2) \odot \cdots \odot \lambda(i_{q-1} \to i_q) = \{l\}$$

Given a path p (which may simply be an edge), we will say that $\lambda(p)$ is the label of p with respect to λ. The identity $\bar{1}$ can then be thought of as the label of the empty path. We will denote $\langle G, \Lambda \rangle$ the DAG obtained by decorating G with Λ.

Although the decoration operator does not need to be associative in the construction presented below, it is a desirable property as path composition is associative. In other words, if a path p is the concatenation of two subpaths p_1 and p_2, then the label of p should be the same no matter how p_1 and p_2 are chosen, i.e.:

$$\lambda(p) = \lambda(p_1 p_2) = \lambda(p_1) \odot \lambda(p_2)$$

where $p_1 p_2$ represents the path p_2 concatenated to p_1.

For every node i in $\Delta(\sigma)$, let $\pi(i)$ be the position of the corresponding splice site in σ. For example, $\pi(4) = 20$ and $\pi(6) = 28$ for the DAG in Figure 4.2. Then every edge $i \to j$ of $\Delta(\sigma)$ represents the sequence fragment $\sigma_{\pi(i),\pi(j)-1}$ of length $Len(i,j) = \pi(j) - \pi(i)$. In the following, we will denote $i \xrightarrow{l} j$ the edge $i \to j$ when labeled with the singleton $\{l\}$ (with respect to some given decoration). We will say that a path satisfies some constraint $\mathcal{C}$ if the parse represented by this path satisfies $\mathcal{C}$, i.e., we will no longer make the distinction between "path" and "parse".

As an example of decoration, consider the DAG $\Delta(\sigma)$ in Figure 4.2, and assume that every edge representing an exon fragment is labeled with a set containing the length modulo three of that fragment, and every edge representing an intron fragment is labeled with the set $\{0\}$. Let $\odot_{Mod}$ be the operator that takes two sets X and Y as operands and returns the set obtained by adding pairwise every element of X to every element of Y (modulo three). Then the label of a path is the set containing its length modulo three. This can be formulated formally by considering the set $L_{mod} = \{0, 1, 2\}$ and defining the L_{Mod}-decoration $\Lambda_{Mod} = (\odot_{Mod}, \bar{1}_{Mod}, \lambda_{Mod})$ of $\Delta(\sigma)$ satisfying:

- $\forall X, Y \subset L_{Mod} : \ X \odot_{Mod} Y = \{(a + b) \bmod 3 \mid a \in X, b \in Y\}$
- $\bar{1}_{Mod} = \{0\}$
- For every edge $i \to j$ in $\Delta(\sigma)$: $\lambda_{Mod}(i \to j) = \begin{cases} \{Len(i,j) \bmod 3\} & \text{if } j \text{ is a left node} \\ \{0\} & \text{otherwise} \end{cases}$

From the above definition, it results that every path $p = i_0 \xrightarrow{l_1} i_1 \xrightarrow{l_2} \cdots \xrightarrow{l_m} i_m$ in $\Delta(\sigma)$ is labeled with:

$$\lambda_{Mod}(p) = \left\{ \left(\sum_{k=1}^{m} l_k \right) \bmod 3 \right\}$$

i.e., the sum modulo three of the lengths of the exons in the parse represented by p.

We now define a decoration that labels every path of $\Delta(\sigma)$ with the protein it encodes. We will use this decoration in Section 4.3.4 to prove that the set of parses coding for a given protein can be expressed as an intrinsic constraint. This decoration is defined by the triplet $\Lambda_\pi = (\bullet, \{\varepsilon\}, \pi)$, where:

- The "$\bullet$" symbol denotes the sequence concatenation operator but it applies over sets:

  $$\forall X, Y \subset \Sigma^* : \ X \bullet Y = \{a \circ b \mid a \in X, b \in Y\}$$

  where Σ^* denotes the set of all possible strings over the alphabet $\{a, c, g, t\}$ and $\circ$ the usual sequence concatenation operator.
- ε is the empty string.
- π is the decoration function defined as follows:
  - For every edge $i \to j$ in $\Delta(\sigma)$: $\pi(i \to j) = \begin{cases} \{\sigma_{\pi(i),\pi(j)-1}\} & \text{if } j \text{ is a left node} \\ \{\varepsilon\} & \text{otherwise} \end{cases}$

It results that the label of every path p in $\Delta(\sigma)$ is defined as the singleton that contains the sequence resulting from the concatenation of the exons in the parse represented by p.

Given the decoration Λ_π, it is then possible to determine the set of full paths in the DAG $\Delta(\sigma)$ that encode a given protein: one just needs to compare the label of each full path in $\Delta(\sigma)$ with the set containing the desired protein. Similarly, given the decoration Λ_{Mod}, it is possible to determine the set of full paths in the DAG $\Delta(\sigma)$ that satisfy the triplicity constraint, e.g., by comparing the label of each full path in $\Delta(\sigma)$ with the set $\{0\}$. In general, if the DAG $\Delta(\sigma)$ contains only a few paths, then it is possible to list these paths one by one, and compare their label with the desired one. However, this approach is not feasible in practice as the number of paths in $\Delta(\sigma)$ can be arbitrarily large.

In the following, we present a more efficient way to determine the paths in $\Delta(\sigma)$ which have a given label. The proposed algorithm avoids the above-mentioned exhaustive search by grouping the paths with common labels. The formalization of this algorithm leads to the notion of summary and decoration-summary system.

4.3.2 Set Summary

Given a set L, a *summary* θ over L is a function that takes one or more elements of L and "combines" them into an element of L which is their "summary". The definition is extended to the subsets of L by defining, via a binary operator denoted $\oplus$, the summary of two (or more) sets as the summary of their union. Examples of summary are the union ($\cup$) and the minimization (min) operations.

Formally, let L be some non-empty set. The function θ is said to be a *summary over* L if there exists a pair $(\oplus, \bar{0})$ such that:

(H5) $\oplus$ is a binary operator on $\hat{L}$, called the *summary operator*.

(H6) $\forall X \in \hat{L}, X \oplus \bar{0} = \bar{0} \oplus X = \theta(X)$.

(H7) θ is a function which maps every element of $\hat{L}$ into an element of $\hat{L}$: $\forall X \in \hat{L}, \theta(X) \in \hat{L}$.

(H8) $\forall X, Y \in \hat{L}, \theta(X \cup Y) = X \oplus Y$.

Given one or more sets $X_0, \ldots, X_m$, we will say that the set $\theta(X_0 \cup \ldots \cup X_m)$ is their summary with respect to θ. Note that the element $\bar{0}$ (which is not an identity for $\oplus$) can be thought of as the summary of the empty set. Also note that the summary operator is both commutative and associative. These two properties will be used in the next section to summarize the labels of the paths in $\Delta(\sigma)$ efficiently. We will denote $(\oplus, \bar{0})$ the θ function.

A particular summary of interest is the *identity function*, Id, which is defined as follows: $Id = (\cup, \emptyset)$. The Id summary of a set contains every element in that set, i.e., for every set X: $Id(X) = X$. It will be seen that most of the intrinsic constraints are defined in terms of the identity summary. Another summary of interest is the summary $\theta_{Prob} = (\oplus_{Prob}, \bar{0}_{Prob})$, which will be used in Section 4.6.2 to find the most probable path in the DAG $\Delta(\sigma)$ satisfying a given (intrinsic) constraint. The summary θ_{Prob} takes as argument a subset of some strictly ordered set L_{Prob} and returns its smallest element. It is defined as follows:

- $\forall X, Y \subset L_{Prob} :\ X \oplus_{Prob} Y = \left\{ \min \{c \mid c \in X \cup Y\} \right\}$
- $\bar{0}_{Prob} = \emptyset$.

As mentioned above, one can define a summary that summarizes the labels of all paths in a given labeled DAG G. By adding certain restrictions to the definitions of decoration and summary, one can then design an algorithm that determines the labels of every full path in G efficiently, i.e., by traversing every edge of G only once. One can also construct efficiently a new DAG G_0 containing only the full paths in G that have a given label. These restrictions are formulated via the concept of decoration-summary system.

4.3.3 Decoration-Summary System

Given a graph G, a decoration-summary system or *DS-system* allows one to perform the following:

1. Summarize the labels of all full paths in G efficiently (the summary of a set of labels being a subset of that set). Let $W(0)$ be the corresponding summary.

2. Partition the full paths in G according to their label.

3. Choose a subset L^0 of $W(0)$, and construct a new DAG G_0 containing only the full paths labeled with an element of L^0.

The DAG G_0 is obtained from the DAG G by first constructing an intermediate DAG denoted $G^\star$. We start this section by providing a bird's-eye view of the construction of the DAGs $G^\star$ and G_0. We then illustrate this construction on a simple example using the decoration Λ_{Mod} of Section 4.3.1 and the identity summary Id of Section 4.3.2. We finish by giving the definition of a decoration-summary system and the mathematical constructions of $G^\star$ and G_0.

Given a graph G, let Λ be an L-decoration of G and θ a summary over L. The DAGs $G^\star$ and G_0 are constructed as follows:

1. For every node i in $\langle G, \Lambda \rangle$, a set $W(i)$ which contains the summary of the labels of the paths between node i and the sink is constructed (see Figure 4.7). For example, $W(i)$ contains the lengths of all exons and/or the counts of all introns in the parses represented by the paths between node i and the sink.

2. The paths in $\langle G, \Lambda \rangle$ are "duplicated" by subdividing every node into subnodes: a subnode is created for every pair (i, a) such that $a \in W(i)$. This results in a new DAG, denoted $G^\star$, in which every node (i, a) can be thought of as a subnode of i from which the paths ending on the sink have the set $\{a\}$ as label. Every two nodes (i, a) and (j, b) in $G^\star$ are connected whenever i links j in G and finding a path with label $\{a\}$ between i and the sink is equivalent to finding a path with label $\{b\}$ between j and the sink (see Figure 4.8).

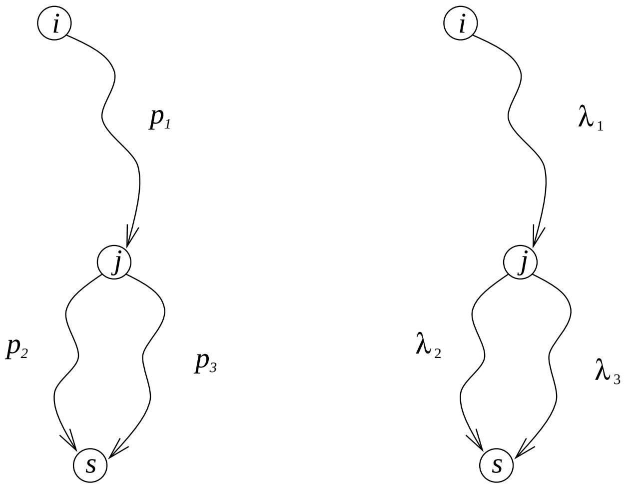


Figure 4.7: Decoration and summary. A decoration specifies the labels of the paths in the DAG: the labels of p_1, p_2 and p_3 are respectively λ_1, λ_2 and λ_3. The label of the path p_1p_2 (obtained by concatenating the paths p_1 and p_2) is $\lambda_{12} = \lambda_1 \odot \lambda_2$; that of path p_1p_3 is $\lambda_{13} = \lambda_1 \odot \lambda_3$. The purpose of defining a decoration-summary system is to compute, for every node i, the summary $W(i)$ of the labels of the paths in the DAG starting on node i and ending on the sink: the summary of the labels λ_2 and λ_3 of the paths p_2 and p_3 is the set $W(j) = \lambda_2 \oplus \lambda_3$. The summary of the labels λ_{12} and λ_{13} of the paths p_1p_2 and p_1p_3 is the set $W(i) = \lambda_{12} \oplus \lambda_{13} = (\lambda_1 \odot \lambda_2) \oplus (\lambda_1 \odot \lambda_3)$. By allowing $\odot$ to distribute over $\oplus$, one can construct the summary $W(i)$ more efficiently, since in this case we have: $W(i) = \lambda_1 \odot (\lambda_2 \oplus \lambda_3) = \lambda_1 \odot W(j)$.

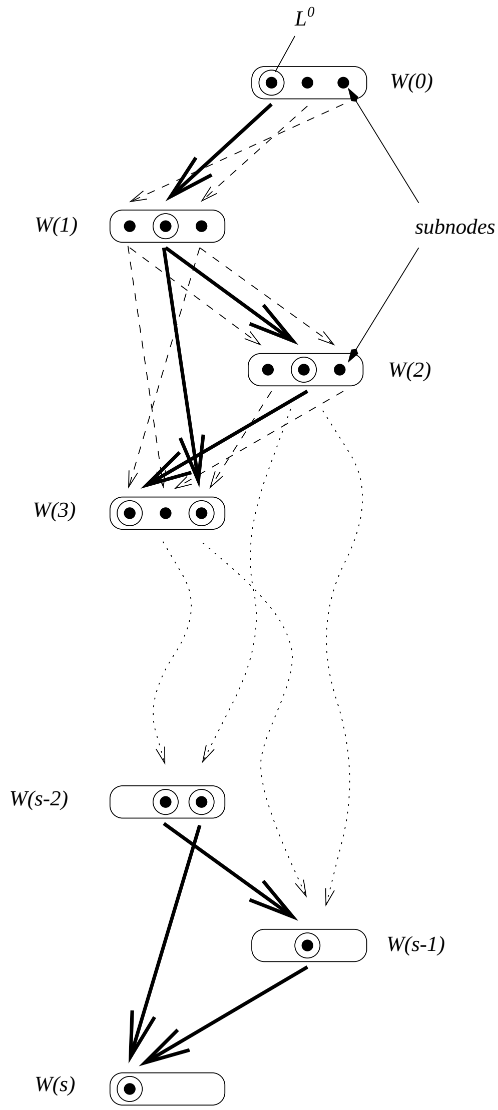


Figure 4.8: Duplication and selection of the valid paths. Every node i is subdivided into a set of subnodes, each of which corresponds to an element in the summary $W(i)$ (in the above figure, every node is divided into three or fewer subnodes). Edges are created between the subnodes to obtain the DAG $G^{\star}$. The subset L^0 of the summary $W(0)$ obtained at the source is used to define the DAG G_0 which contains only the paths in $G^{\star}$ whose label is in L^0. In the above figure, the nodes that are circled and the edges that are thicker represent the nodes and edges of G_0.

3. Given a set L^0, a new DAG G_0 which contains only the full paths of $G^\star$ labeled with an element of L^0 is constructed. This is achieved by constructing a set $V(i)$, for every node i, such that $a \in V(i)$ if and only if $a \in W(i)$ and there exists a full path labeled with an element of L^0 passing through (i, a). In particular, we have: $V(0) = L^0$. In the DAG G_0, every two nodes (i, a) and (j, b) are connected if and only if (i, a) links (j, b) in $G^\star$, and both (i, a) and (j, b) belong to a full path labeled with an element of L^0, i.e., $a \in V(i)$ and $b \in V(j)$.

As an example of the above construction, let us consider the DAG $G = \Delta(\sigma)$ introduced earlier, as well as the decoration Λ_{Mod} of Section 4.3.1 and the identity summary $Id = (\cup, \emptyset)$. Then, for every node i in the DAG $\langle \Delta(\sigma), \Lambda_{Mod} \rangle$, the set $W(i)$ contains the lengths modulo three of the proteins represented by the paths between node i and the sink (see Figure 4.9). Therefore, every node in $\langle \Delta(\sigma), \Lambda_{Mod} \rangle$ can be subdivided into one, two or three subnodes. In the DAG $G^\star$, subnode (i, a) links subnode (j, b) if and only if (1) there exists a path of length a between i and the sink and a path of length b between j and the sink, and (2) finding a path of length a between node i and the sink is equivalent to finding a path of length $b = (a - c) \bmod 3$ between node j and the sink, where c is the length of the sequence fragment (0 if it is non-coding) associated with the edge connecting i and j. For example, in Figure 4.9, node 4 is subdivided into two subnodes $(4, 0)$ and $(4, 1)$ because two out of the three paths going from this node to the sink have the set $\{1\}$ as label ($4 \xrightarrow{2} 6 \xrightarrow{2} 8$ and $4 \xrightarrow{0} 7 \xrightarrow{1} 8$) and the third path has the set $\{0\}$ as label ($4 \xrightarrow{2} 6 \xrightarrow{0} 7 \xrightarrow{1} 8$). The labels for edges $4 \xrightarrow{2} 6$ and $4 \xrightarrow{0} 7$ are respectively $\{2\}$ and $\{0\}$, therefore node $(4, 0)$ links node $(6, 1)$ and node $(4, 1)$ links both the nodes $(6, 2)$ and $(7, 1)$.

By taking $L^0 = \{0\}$, one obtains a new DAG G_0 in which every path between the source and the sink represents a protein whose length is divisible by three. The nodes in G_0 are the nodes in $G^\star$ that are passed through by a full path labeled with $\{0\}$. For every node i in $\langle \Delta(\sigma), \Lambda_{Mod} \rangle$, the subnodes of node i that are passed through by a full path labeled with $\{0\}$ are as specified by the set $V(i)$. For example, $0 \in V(4)$ because there exists a full path labeled with $\{0\}$ passing through node $(4, 0)$. On the

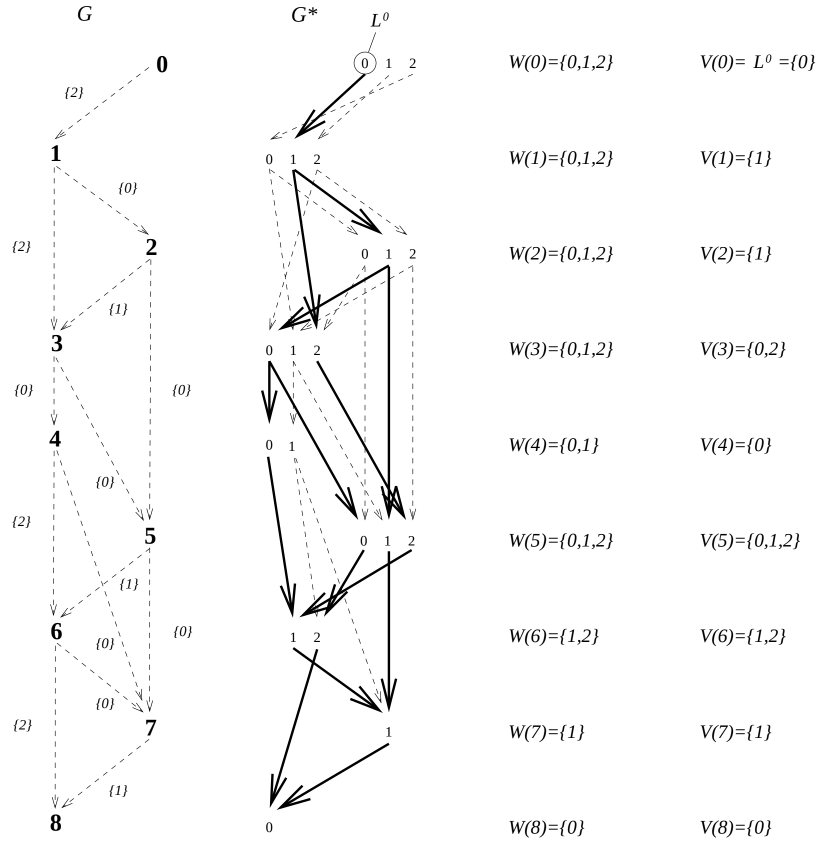


Figure 4.9: The DAGs $G^{\star}$ and G_0 obtained from the DAG G in Figure 4.2 when applying the triplicity constraint. The DAG G_0 is obtained from $G^{\star}$ by excerpting every path starting on subnode $(0, 0)$, i.e., by taking $L^0 = \{0\}$.

contrary, $1 \notin V(4)$ because every full path passing through node $(4,1)$ is labeled with $\{1\}$ or $\{2\}$.

We are now ready to provide a formal definition for a DS-system. Let $G = (V, E)$ be *any* directed acyclic graph. Let $\Lambda = (\odot, \bar{1}, \lambda)$ be a L-decoration of G for some non-empty set L and $\theta = (\oplus, \bar{0})$ a summary over L. The triplet $\langle G, \Lambda, \theta \rangle$ is said to be a *decoration-summary system* of G if and only if:

(H9) $\odot$ distributes over $\oplus$:

$$\forall X, Y, Z \in \hat{L}, X \odot (Y \oplus Z) = (X \odot Y) \oplus (X \odot Z) \text{ and } (Y \oplus Z) \odot X = (Y \odot X) \oplus (Z \odot X)$$

(H10) Every set contains its own summary: $\forall X \in \hat{L}, \theta(X) \subset X$.

Condition (H9) implies that the labels and the summaries are of the same nature, i.e., sets of labels. Condition (H10) requires that the summary of a set of labels be a label belonging to that set. These two properties allow one to perform efficiently the following operations:

1. Obtain the summary $W(0)$ of the labels of all full paths in G.

2. Consider a subset L^0 of $W(0)$.

3. Determine all full paths in G labeled with an element of L^0.

In what follows, we provide a construction to achieve the above goals. We will assume that G contains $s+1$ vertices ($s > 0$) numbered between 0 and s, and that node 0 is the only source in G and node s its only sink (the case in which the DAG G contains more than one source or sink will be discussed at the end of this section).

For some non-empty set L, let us fix some L-decoration $\Lambda = (\odot, \bar{1}, \lambda)$ of G and some summary $\theta = (\oplus, \bar{0})$ over L such that the triplet $\langle G, \Lambda, \theta \rangle$ is a DS-system. As mentioned above, our goal is to construct the DAG G_0 whose only paths are the full paths in G labeled with an element of L^0.

The construction of G_0 from G is performed in two steps. First, the intermediate DAG $G^\star$ is constructed by subdividing every node in $\langle G, \Lambda \rangle$ into a set of subnodes such that the paths in $G^\star$ starting on a given subnode and ending on its sink have the same label, e.g., the set $\{l\}$. As a consequence, the summary of the labels of the paths in $G^\star$ starting on this given subnode is the set $\{l\}$. In the second step, the desired DAG G_0 is obtained by excerpting all full paths in $G^\star$ whose label is in L^0.

For every node i in G, let us denote $P(i)$ the set of paths between the i^{th} node and the sink s of G. The summary operator $\oplus$ is used to compute the summary $W(i)$ of the labels of the paths between i and s:

$$W(i) = \theta\Big(\bigcup_{p \in P(i)} \lambda(p)\Big) = \bigoplus_{p \in P(i)} \lambda(p) \tag{4.1}$$

Note that, in the above formula, the order in which the labels of the paths are computed does not matter since $\oplus$ is both commutative and associative. Let $Succ(i)$ be the set of successors of node i, for $0 \leq i < s$. Using the associativity of $\oplus$ and the distributivity of $\odot$ over $\oplus$, the summary $W(i)$ at node i can be expressed as a function of the summaries of its successors:

$$W(i) = \bigoplus_{j \in Succ(i)} \lambda(i \to j) \odot W(j) \tag{4.2}$$

For the sink s, we have: $W(s) = \bar{1}$. Since every set contains its own summary, we have, for any $a \in L$:

$$a \in W(i) \Leftrightarrow \exists j \in Succ(i) : \ a \in \lambda(i \to j) \odot W(j) \tag{4.3}$$

Let $N(i) = \{(i,a) \mid a \in W(i)\}$. The elements in $N(i)$ correspond to the subnodes of node i, i.e., every element in $N(i)$ corresponds to a node in $G^\star$. Using the $N(i)$ set, equation (4.3) can be rewritten as follows:

$$(i,a) \in N(i) \Leftrightarrow \exists j \in Succ(i), \exists b \in L : \ (j,b) \in N(j) \ \mathit{and} \ \{a\} = \lambda(i \to j) \odot \{b\} \tag{4.4}$$

This suggests the following definition for the edges of $G^\star$:

$$(i,a) \text{ links } (j,b) \text{ in } G^\star \text{ if and only if } \begin{cases} i \text{ links } j \text{ in } G \\ (i,a) \in N(i), \ (j,b) \in N(j) \\ \{a\} = \lambda(i \to j) \odot \{b\} \end{cases} \tag{4.5}$$

which can be read as follows: for every $a \in W(i)$ and $b \in W(j)$, if the $(i,a) \rightarrow (j,b)$ edge exists in $G^\star$, then finding a path of label $\{a\}$ between i and s is equivalent to finding a path of label $\{b\}$ between j and s.

Let us define the set of source nodes of $G^\star$ as the set $\{(0, l_0) \mid l_0 \in L^0\}$, and the set of sink nodes as the set $\{(s, l_s) \mid l_s \in \bar{1}\}$. By construction, every path in $G^\star$ starting on one of the subnodes in $N(i)$ ends on one of the sink (s, l_s) of $G^\star$: the label of every path starting on subnode $(i,a) \in N(i)$ and ending on one of the sinks is $\{a\}$, as is the summary of the labels of the paths between (i,a) and any of the sinks (s, l_s). It follows that every (non empty) path in $G^\star$ starting on one of its source nodes $(0, l_0)$ ends on one of its sink nodes (s, l_s) and has the set $\{l_0\}$ as label; i.e., the DAG obtained by considering every path in $G^\star$ between its source nodes $(0, l_0)$ and its sink nodes (s, l_s) is the desired DAG G_0.

Formally, let $M(i)$ be the set of subnodes of node i in the final DAG G_0, and $V(i) = \{a \mid (i,a) \in M(i)\}$. We then have: $V(0) = L^0$, and for every node j, $0 < j \leq s$:

$$V(j) = \{b \mid a \in V(i) \ \mathit{and} \ (i,a) \rightarrow (j,b) \in G^\star\} \tag{4.6}$$

In other words, the edges in G_0 must satisfy the following property:

$$(i,a) \text{ links } (j,b) \text{ in } G_0 \text{ if and only if } \begin{cases} (i,a) \text{ links } (j,b) \text{ in } G^\star \\ \\ (i,a) \in M(i), \ \ (j,b) \in M(j) \end{cases} \tag{4.7}$$

The similarity between equation (4.7) and equation (4.5) suggests that the algorithm that constructs G_0 from $G^\star$ is similar to the algorithm that constructs $G^\star$ from G. We will describe and analyze the complexity of the algorithm that constructs $G^\star$ then G_0 from G in Section 4.3.5 (see Figure 4.10 on page 123).

It should be noted that the case in which the DAG G contains one or more source nodes or sink nodes can be reduced to the above case by creating a virtual source node and a virtual sink node, and by adding edges labeled with the set $\bar{1}$ (the identity for $\odot$) between the virtual source node and the actual source nodes, and between the virtual sink node and the actual sink nodes.

In the following, we say that a given node (i, a) in the DAG $G^\star$ or G_0 is a left node if i is a left node, and we say that it is a right node if i is a right node. We extend in the same way the definitions of lr-edges, rr-edges, rl-edges and ll-edges.

We have seen that it is possible to represent by a DAG G_0 the set of all gene models satisfying the triplicity constraint. The DAG G_0 is obtained from G by first defining a decoration-summary system, and then considering a particular subset, L^0, of the set of labels. The idea of using a DS-system and a set L^0 to define a constraint that operates on gene structure leads to the notion of intrinsic constraints.

4.3.4 Definition of an Intrinsic Constraint

In this section, we define the notion of intrinsic constraint formally, and provide two examples of such constraints: the first screens out all gene models that do not satisfy the triplicity constraint. The second screens out all gene models that do not code for a given sequence of amino acids.

Let, as before, $\Delta(\sigma)$ be the DAG that represents the parses of a DNA sequence σ satisfying the *a priori* constraints. A constraint $\mathcal{C}$ is *intrinsic* if for some non-empty set L, there exist a L-decoration $\Lambda = (\odot, \bar{1}, \lambda)$ of $\Delta(\sigma)$, a summary $\theta = (\oplus, \bar{0})$ over L, and a subset L^0 of L such that:

(H11) $\langle \Delta(\sigma), \Lambda, \theta \rangle$ is a DS-system.

(H12) For every full path labeled with an element l of L^0, l belongs to the summary $W(0)$ of the labels of all full paths in $\Delta(\sigma)$.

(H13) There is a one-to-one correspondence between the parses of σ satisfying $\mathcal{C}$ and the full paths in $\langle \Delta(\sigma), \lambda \rangle$ labeled with an element of L^0 (with respect to Λ).

We will write: $\mathcal{C} = \langle L, \Lambda, \theta, L^0 \rangle$.

For example, let us consider the DS-system $\langle \Delta(\sigma), \Lambda_{Mod}, Id \rangle$ introduced in the previous section. By taking $L^0 = \{0\}$, one obtains all parses of σ satisfying the triplicity constraint.

Given a DNA sequence σ and an intrinsic constraint $\mathcal{C}$, the DS-system of (H11) allows one to determine efficiently the parses of σ satisfying $\mathcal{C}$. Condition (H12) prevents one from defining a set L^0 whose elements cannot be path labels. Finally, condition (H13) defines the relationship between the paths in $\Delta(\sigma)$ and the parses satisfying $\mathcal{C}$.

From the above definition, it follows that every intrinsic constraint $\mathcal{C}$ can be described formally by the specification of:

- a non-empty set L,
- an L-decoration Λ, i.e.:
  - a decoration operator $\odot$,
  - an identity $\bar{1}$ for $\odot$,
  - a decoration function λ,
- a summary θ over L, i.e.:
  - a summary operator $\oplus$,
  - an identity $\bar{0}$ for $\oplus$,
- a subset L_0 of L.

The decoration Λ and the summary θ must satisfy the axioms (H1 thru H13). The decoration function only needs to be specified for the edges, since the labels of the paths can be computed from the labels of the edges by using the decoration operator. We will denote $\Delta_{\mathcal{C}}(\sigma)$ the DAG that represents the parses of σ satisfying $\mathcal{C}$.

We finish this section by providing an example of intrinsic constraint which can be used to retrieve all gene models of a sequence σ that match a given sequence μ of amino acids [2]. This constraint is defined by the system $\mathcal{C}_\pi = \langle \Sigma^*, \Lambda_\pi, Id_{\Sigma^*}, L_\pi^0(\mu) \rangle$ where:

[2] More examples of intrinsic constraints will be given in Section 4.4.

- Σ^* is the set of all possible strings over the alphabet $\{a, c, g, t\}$.
- Λ_π is as defined in Section 4.3.1, i.e., $\Lambda_\pi = (\bullet, \{\varepsilon\}, \pi)$ where:
  - $\forall X, Y \subset \Sigma^* : \; X \bullet Y = \{a \circ b \mid a \in X, b \in Y\}$
  - ε is the empty string
  - For every edge $i \to j$ in $\Delta(\sigma)$: $\pi(i \to j) = \begin{cases} \{\sigma_{\pi(i),\pi(j)-1}\} & \text{if } j \text{ is a left node} \\ \{\varepsilon\} & \text{otherwise} \end{cases}$
- $Id_{\Sigma^*} = (\cup, \emptyset)$, i.e., $\forall X \subset \Sigma^* : \; Id_{\Sigma^*}(X) = X$.
- $L^0_\pi(\mu) = \{\mu\}$.

Since the decoration Λ_π labels every path of $\Delta(\sigma)$ with the sequence of amino acids it encodes, and the summary Id_{Σ^*} returns the set of strings it takes as argument, the summary $W(0)$ obtained at the source of $\Delta(\sigma)$ contains all protein sequences encoded by the paths of $\Delta(\sigma)$. In contrast, the final DAG $\Delta_{\mathcal{C}_\pi}(\sigma)$ obtained by selecting all full paths of $\Delta(\sigma)$ labeled with $\{\mu\}$ contains only the paths of $\Delta(\sigma)$ encoding the protein μ.

Note that the purpose of the above example is only to illustrate the power of our formalism. In practice, to make the intrinsic constraint $\mathcal{C}_\pi$ computationally tractable, one would have to add to the label of every edge a score reflecting the coding potential of the associated sequence fragment. One would obtain, then, a linear version of the quadratic algorithm proposed in (Gelfand et al., 1996a). This result will be discussed further in Section 4.6. We now describe and analyze the complexity of the algorithm for constructing the DAG $\Delta_{\mathcal{C}}(\sigma)$ for some given intrinsic constraint $\mathcal{C}$.

4.3.5 Complexity of the Proposed Construction

In this section, we provide an algorithm for constructing the DAG G_0 from the DAG G. We first analyze the complexity of this algorithm in the general case and then consider the particular case of the

DAG $\Delta(\sigma)$.

The algorithm for constructing G_0 from G proceeds in three steps which respectively correspond to the construction of $\langle G, \Lambda \rangle$, $G^\star$, and G_0. The first step constructs the DAG $\langle G, \Lambda \rangle$ by labeling the edges in G with some decoration function λ. The second step constructs the DAG $G^\star$ by subdividing every node in $\langle G, \Lambda \rangle$ into a number of subnodes such that every node in $G^\star$ corresponds to a subnode of $\langle G, \Lambda \rangle$. The nodes and edges of $G^\star$ are created as the summaries are computed. This computation is performed bottom-up, i.e., from the sink to the source (see Figure 4.10). The third and last step constructs the DAG G_0 by marking the nodes and edges in $G^\star$ that belong to a path labeled with an element of L^0. This marking is performed top-down, i.e., from the source to the sink.

We now proceed with the complexity analysis of the above steps. The time complexity of the first step is $T_\lambda e$, where T_λ is the time taken for labeling a single edge and e is the number of edges in G. We have seen in Section 4.2.3 that the DAG $\Delta(\sigma)$ contains at most $2(n-1)$ edges; therefore, the time taken for decorating the DAG $\Delta(\sigma)$ is $(2n-1)T_\lambda$ in the worst case.

Let $T_\odot$ be the time taken to apply the $\odot$ operator to two singleton sets and $T_\oplus$ the time taken to summarize a set containing a single element. It is reasonable to assume that the time taken to summarize a set X of size x is $xT_\oplus$. We will also assume that the time taken to apply $\odot$ to two set X and Y of respective size $x+1$ and y is $(x+y)T_\odot$.

Let $T(i)$ be the time taken to compute the set $W(i)$ as given in Equation (4.2). The worst case occurs when, for every successor j of i, both the sets $W(j)$ and $\lambda(i \to j) \odot W(j)$ contain $|L|$ elements. We have then: $T(i) = |Succ(i)||L|(T_\odot + T_\oplus) + T(j)$. Therefore, the time taken to compute all summaries is in the worst case: $T(0) = O(e|L|(T_\odot + T_\oplus))$. As noted earlier, the DAG $G^\star$ contains at most $e|L|$ edges. It results that the marking of the edges that belong to G^0 can be performed in $O(e|L|)$ time in the worst case, i.e, when all paths in $G^\star$ are selected. Finally, the time taken to construct the DAG G_0 from the

begin

for $i \leftarrow 0$ **to** s **do** $\{$*Initialization Phase*$\}$

  $V(i) \leftarrow \emptyset$

  for a **in** L **do**

    $B(i,a) \leftarrow \emptyset$; $C(i,a) \leftarrow \emptyset$

$W(s) \leftarrow \bar{1}$ $\{$*Identity for* $\odot\}$ $\{$*Bottom-up Computation of the Summaries*$\}$

for $i \leftarrow s-1$ **downto** 0 **do** $\{$*See Equation (4.4)*$\}$

  $W(i) \leftarrow \bar{0}$ $\{$*Identity for* $\oplus\}$

  for j **in** $Succ(i)$ **do**

    for b **in** $W(j)$ **do**

      $\{a\} \leftarrow \lambda(i \rightarrow j) \odot \{b\}$; $B(i,a) \leftarrow B(i,a) \cup \{(j,b)\}$; $W(i) \leftarrow W(i) \oplus \{a\}$

$V(0) \leftarrow W(0) \cap L(0)$ $\{$*Top-down Extraction of the Valid Paths*$\}$

for $i \leftarrow 0$ **to** s **do** $\{$*See Equation (4.6)*$\}$

  for a **in** $V(i)$ **do**

    for (j,b) **in** $B(i,a)$ **do**

      $C(i,a) \leftarrow C(i,a) \cup \{(j,b)\}$; $V(j) \leftarrow V(j) \cup \{b\}$

end

Figure 4.10: Algorithm for constructing G_0 from G. For every i such that $0 \le i \le s$ and every a such that $a \in L$, $Succ(i)$ is the set of successors of i in G, $B(i,a)$ the set of successors of (i,a) in $G^\star$, and $C(i,a)$ the set of successors of (i,a) in G_0.

DAG G is in the worst case:

$$T(n) = O(e(T_\lambda + |L|(T_\odot + T_\oplus))) \tag{4.8}$$

In the particular case of the DAG $\Delta(\sigma)$, the time taken to compute the DAG $\Delta_\mathcal{C}$ can be expressed as the quantity:

$$T(n) = O(n(T_\lambda + |L|(T_\odot + T_\oplus))) \tag{4.9}$$

since there are no more than $2(n-1)$ edges in $\Delta(\sigma)$.

Let $K = T_\lambda + |L|(T_\odot + T_\oplus)$. Then we have: $T(n) = O(Kn)$. As mentioned in Section 3.5, the only parameter that varies when performing gene structure prediction is n, i.e., the length of the sequence being analyzed. Assuming that K is a constant, the above algorithm then takes linear time with respect to gene structure prediction: $T(n) = O(n)$. Note however that it is important to take the value of K into consideration when designing an intrinsic constraint as a very large value of K may result in a time-consuming algorithm.

To finish, recall from Section 4.2.3 that the construction of $\Delta(\sigma)$ from σ takes $O(T + n)$ time, where T is the time taken to predict the splice sites of σ. Therefore, the DAG $\Delta_\mathcal{C}$ can be constructed from σ in $O(Kn + T)$ time. Since it is often simpler to first define two intrinsic constraints separately, and then combine them to form a more complex constraint, we now provide a scheme for composing constraints.

4.3.6 Composing Constraints

Let $\mathcal{C}_1 = \langle L_{\mathcal{C}_1}, \Lambda_{\mathcal{C}_1}, \theta_{\mathcal{C}_1}, L^0_{\mathcal{C}_1} \rangle$ and $\mathcal{C}_2 = \langle L_{\mathcal{C}_2}, \Lambda_{\mathcal{C}_2}, \theta_{\mathcal{C}_2}, L^0_{\mathcal{C}_2} \rangle$ be two intrinsic constraints, and $\Delta_{\mathcal{C}_1}(\sigma)$ and $\Delta_{\mathcal{C}_2}(\sigma)$ the corresponding DAGs for a given sequence σ. Every path in $\Delta_{\mathcal{C}_1}(\sigma)$ and $\Delta_{\mathcal{C}_2}(\sigma)$ represents a gene candidate satisfying $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. In this section, we show that we can construct a new DAG, denoted $\Delta_{\mathcal{C}_1 \otimes \mathcal{C}_2}(\sigma)$, whose paths are the paths of $\Delta_{\mathcal{C}_1}(\sigma)$ that satisfy $\mathcal{C}_2$.

Note that there are two ways to obtain the DAG $\Delta_{\mathcal{C}_1 \otimes \mathcal{C}_2}(\sigma)$: the first is to construct the DAG $\Delta_{\mathcal{C}_1}(\sigma)$, and then apply $\mathcal{C}_2$ to the paths of $\Delta_{\mathcal{C}_1}(\sigma)$; the second, described below, is to construct directly the DAG $\Delta_{\mathcal{C}_1 \otimes \mathcal{C}_2}(\sigma)$ from $\Delta(\sigma)$, i.e., by composing the constraints $\mathcal{C}_1$ and $\mathcal{C}_2$. In what follows, we show that $\mathcal{C}_1$ and $\mathcal{C}_2$ can be composed into a new constraint, denoted $\mathcal{C}_1 \otimes \mathcal{C}_2$, which can be thought of as a refinement of $\mathcal{C}_1$ by $\mathcal{C}_2$, and which is intrinsic: $\mathcal{C}_1 \otimes \mathcal{C}_2 = \langle L_{\mathcal{C}_1 \otimes \mathcal{C}_2}, \Lambda_{\mathcal{C}_1 \otimes \mathcal{C}_2}, \theta_{\mathcal{C}_1 \otimes \mathcal{C}_2}, L^0_{\mathcal{C}_1 \otimes \mathcal{C}_2} \rangle$. The aim of this section is to define the parameters $\mathcal{C}_1 \otimes \mathcal{C}_2$, $\Lambda_{\mathcal{C}_1 \otimes \mathcal{C}_2}$, $\theta_{\mathcal{C}_1 \otimes \mathcal{C}_2}$ and $L^0_{\mathcal{C}_1 \otimes \mathcal{C}_2}$. The proposed construction can be easily generalized to compose an arbitrary number of constraints.

Intuitively, the label $\lambda_{\mathcal{C}_1 \otimes \mathcal{C}_2}(p)$ of every path p in $\Delta(\sigma)$ should reflect the label of p with respect to $\Lambda_{\mathcal{C}_1}$ as well as its label with respect to $\Lambda_{\mathcal{C}_2}$. Since both the sets $\lambda_{\mathcal{C}_1}(p)$ and $\lambda_{\mathcal{C}_2}(p)$ contain a single element, e.g., a_1 and a_2 respectively, we define $\lambda_{\mathcal{C}_1 \otimes \mathcal{C}_2}(p)$ as the set containing the pair (a_1, a_2), i.e., the Cartesian product of $\lambda_{\mathcal{C}_1}$ and $\lambda_{\mathcal{C}_2}$. By extension, we define the set $L_{\mathcal{C}_1 \otimes \mathcal{C}_2}$ as the Cartesian product of $L_{\mathcal{C}_1}$ and $L_{\mathcal{C}_2}$:

$$L_{\mathcal{C}_1 \otimes \mathcal{C}_2} = L_{\mathcal{C}_1} \times L_{\mathcal{C}_2} = \{(a_1, a_2) \mid a_1 \in L_{\mathcal{C}_1}, a_2 \in L_{\mathcal{C}_2}\}$$

It follows that the set $L^0_{\mathcal{C}_1 \otimes \mathcal{C}_2}$ is the Cartesian product of $L^0_{\mathcal{C}_1}$ and $L^0_{\mathcal{C}_2}$: $L^0_{\mathcal{C}_1 \otimes \mathcal{C}_2} = L^0_{\mathcal{C}_1} \times L^0_{\mathcal{C}_2}$.

Let $\Lambda_{\mathcal{C}_1} = (\odot_{\mathcal{C}_1}, \bar{1}_{\mathcal{C}_1}, \lambda_{\mathcal{C}_1})$ and $\Lambda_{\mathcal{C}_2} = (\odot_{\mathcal{C}_2}, \bar{1}_{\mathcal{C}_2}, \lambda_{\mathcal{C}_2})$. The $L_{\mathcal{C}_1 \otimes \mathcal{C}_2}$-decoration of $\Delta(\sigma)$ is then defined by the triplet $(\odot_{\mathcal{C}_1 \otimes \mathcal{C}_2}, \bar{1}_{\mathcal{C}_1 \otimes \mathcal{C}_2}, \lambda_{\mathcal{C}_1 \otimes \mathcal{C}_2})$ such that:

- $\forall X, Y \subset L_{\mathcal{C}_1 \otimes \mathcal{C}_2}$:

$$X \odot_{\mathcal{C}_1 \otimes \mathcal{C}_2} Y = \bigcup_{\substack{(a_1,a_2) \in X \\ (b_1,b_2) \in Y}} (\{a_1\} \odot_{\mathcal{C}_1} \{b_1\}) \times (\{a_2\} \odot_{\mathcal{C}_2} \{b_2\})$$

- $\bar{1}_{\mathcal{C}_1 \otimes \mathcal{C}_2} = \bar{1}_{\mathcal{C}_1} \times \bar{1}_{\mathcal{C}_2}$.

- For every edge $i \to j$ in $\Delta(\sigma)$: $\lambda_{\mathcal{C}_1 \otimes \mathcal{C}_2}(i \to j) = \lambda_{\mathcal{C}_1}(i \to j) \times \lambda_{\mathcal{C}_2}(i \to j)$.

Using the above definition, the label of every path $p = i_0 \xrightarrow{(l_1,l'_1)} i_1 \xrightarrow{(l_2,l'_2)} \cdots \xrightarrow{(l_m,l'_m)} i_m$ in $\Delta(\sigma)$

is (with respect to $\Lambda_{\mathcal{C}_1 \otimes \mathcal{C}_2}$):

$$\lambda_{\mathcal{C}_1 \otimes \mathcal{C}_2}(p) = \{(x, x')\} \Leftrightarrow \begin{cases} \{x\} = \{l_1\} \odot_{\mathcal{C}_1} \{l_2\} \odot_{\mathcal{C}_1} \cdots \odot_{\mathcal{C}_1} \{l_m\} \\ \\ \{x'\} = \{l'_1\} \odot_{\mathcal{C}_2} \{l'_2\} \odot_{\mathcal{C}_2} \cdots \odot_{\mathcal{C}_2} \{l'_m\} \end{cases}$$

For every set Z, the summary $\theta_{\mathcal{C}_1 \otimes \mathcal{C}_2}(Z)$ of Z is then obtained as follows (see Figure 4.11). A first set denoted $S(Z)$ is introduced with the purpose of summarizing the elements of Z with respect to $\theta_{\mathcal{C}_1}$, i.e., by ignoring the second component in the path labels:

$$S(Z) = \theta_{\mathcal{C}_1}(\{x \mid (x, y) \in Z\}) \tag{4.10}$$

Then a set $C(Z)$ containing every element of Z whose first component is in $S(Z)$ is constructed:

$$C(Z) = \{(x, y) \in Z \mid x \in S(Z)\} \tag{4.11}$$

This set is used to compute the summaries, with respect to $\theta_{\mathcal{C}_2}$, of the elements of $C(Z)$ that have the same first component:

$$\forall x \in S(Z): \; T_x(Z) = \theta_{\mathcal{C}_2}(\{y \mid (x, y) \in C(Z)\}) \tag{4.12}$$

The summary $\theta_{\mathcal{C}_1 \otimes \mathcal{C}_2}(Z)$ of Z is then:

$$\theta_{\mathcal{C}_1 \otimes \mathcal{C}_2}(Z) = \{(x, y) \in Z \mid x \in S(Z), y \in T_x(Z)\} \tag{4.13}$$

Therefore, the summary $\theta_{\mathcal{C}_1 \otimes \mathcal{C}_2}$ is defined as the couple $(\oplus_{\mathcal{C}_1 \otimes \mathcal{C}_2}, \bar{0}_{\mathcal{C}_1 \otimes \mathcal{C}_2})$ such that:

- $\forall X, Y \subset L_{\mathcal{C}_1 \otimes \mathcal{C}_2}$: $X \oplus_{\mathcal{C}_1 \otimes \mathcal{C}_2} Y = \theta_{\mathcal{C}_1 \otimes \mathcal{C}_2}(X \cup Y)$ where $\theta_{\mathcal{C}_1 \otimes \mathcal{C}_2}$ is as defined in Equation (4.13).
- $\bar{0}_{\mathcal{C}_1 \otimes \mathcal{C}_2} = \bar{0}_{\mathcal{C}_1} \times \bar{0}_{\mathcal{C}_2}$.

It is easily shown that the constraint $\mathcal{C}_1 \otimes \mathcal{C}_2$ satisfies all of the axioms (H1 thru H13) and is therefore intrinsic. We now proceed with the complexity analysis of the above construction. The set $L_{\mathcal{C}_1 \otimes \mathcal{C}_2}$ is the Cartesian product of the sets $L_{\mathcal{C}_1}$ and $L_{\mathcal{C}_2}$; therefore, it contains $|L_{\mathcal{C}_1}||L_{\mathcal{C}_2}|$ elements. The

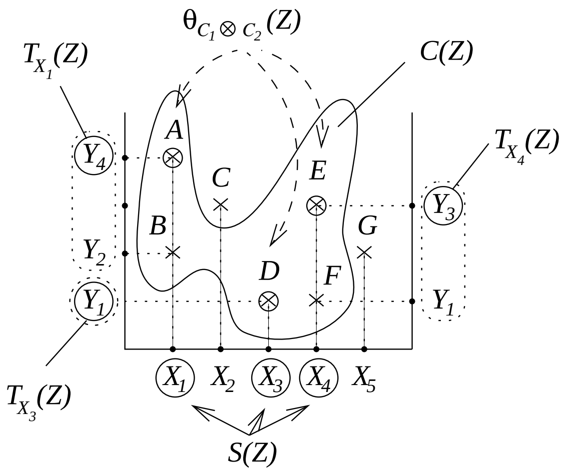


Figure 4.11: Composition of the summaries: every element in the set $Z = \{A, B, C, D, E, F, G\}$ to be summarized is a pair represented by a point whose abscissa and ordinate correspond to the first and second component of that pair. The summary of Z with respect to $\theta_{\mathcal{C}_1 \otimes \mathcal{C}_2}$ is computed in five steps: (1) The summary with respect to $\theta_{\mathcal{C}_1}$ of the abscissas of all pairs X_1, X_2, X_3, X_4, X_5 is computed; in the above figure, this summary is $S(Z) = \{X_1, X_3, X_4\}$. (2) The points whose abscissa is in $S(Z)$ are selected to form the set $C(Z)$; here, this set contains the points A, B, D, E, F. (3) A number of clusters containing the ordinates of the points in $C(Z)$ are constructed such that any two points with the same abscissa have their ordinate in the same cluster and vice-versa; we obtain three clusters: $\{Y_4, Y_2\}$ which contains the ordinates of A and B, $\{Y_1\}$ (ordinate of D) and $\{Y_3, Y_1\}$ (ordinates of E and F). (4) The summary with respect to $\theta_{\mathcal{C}_2}$ of the elements in every cluster is computed: for instance, $T_{X_1}(Z) = \{Y_4\}$, $T_{X_3}(Z) = \{Y_1\}$, $T_{X_4}(Z) = \{Y_3\}$. (5) The summary of Z with respect to $\theta_{\mathcal{C}_1 \otimes \mathcal{C}_2}$ is then the set containing the points whose abscissa x is in $S(Z)$ and ordinate is in $T_x(Z)$. In the above figure, there are three such points: A, D and E.

label of every edge $i \to j$ with respect to $\lambda_{\mathcal{C}_1 \otimes \mathcal{C}_2}$ is the Cartesian product of two singletons: the label of $i \to j$ with respect to $\lambda_{\mathcal{C}_1}$ and the label of $i \to j$ with respect to $\lambda_{\mathcal{C}_2}$, therefore: $T_{\lambda_{\mathcal{C}_1 \otimes \mathcal{C}_2}} = T_{\lambda_{\mathcal{C}_1}} + T_{\lambda_{\mathcal{C}_2}}$. The decoration operator $\odot_{\lambda_{\mathcal{C}_1 \otimes \mathcal{C}_2}}$ takes two sets of labels as operands and computes their combined labels by applying the operators $\odot_{\lambda_{\mathcal{C}_1}}$ and $\odot_{\lambda_{\mathcal{C}_2}}$ and performing a Cartesian product. It is easily seen that the time taken to apply $\odot_{\lambda_{\mathcal{C}_1 \otimes \mathcal{C}_2}}$ to two singleton sets is $T_{\odot_{\mathcal{C}_1 \otimes \mathcal{C}_2}} = T_{\odot_{\mathcal{C}_1}} + T_{\odot_{\mathcal{C}_2}}$. Finally, the summary $\theta_{\mathcal{C}_1 \otimes \mathcal{C}_2}(Z)$ of a set Z is obtained by first summarizing Z with respect to $\theta_{\mathcal{C}_1}$ and then summarizing the subset $C(Z)$ of Z with respect to $\theta_{\mathcal{C}_2}$. Therefore, the time taken to summarize a set with a single element is: $T_{\oplus_{\mathcal{C}_1 \otimes \mathcal{C}_2}} = T_{\oplus_{\mathcal{C}_1}} + T_{\oplus_{\mathcal{C}_2}}$. It results that the total time complexity for constructing $\Delta_{\mathcal{C}_1 \otimes \mathcal{C}_2}(\sigma)$ from $\Delta(\sigma)$ is, using equation (4.9):

$$T_{\mathcal{C}_1 \otimes \mathcal{C}_2}(n) = O(n(T_{\lambda_{\mathcal{C}_1}} + T_{\lambda_{\mathcal{C}_2}} + |L_{\mathcal{C}_1}||L_{\mathcal{C}_2}|(T_{\odot_{\mathcal{C}_1}} + T_{\odot_{\mathcal{C}_2}} + T_{\oplus_{\mathcal{C}_1}} + T_{\oplus_{\mathcal{C}_2}}))) \tag{4.14}$$

Defining a constraint as the composition of simpler ones has a number of advantages: first, it is in general easier to specify two or more constraints separately than altogether. Secondly, by adding constraints one at a time, we can measure the effect of each constraint on prediction accuracy, and hopefully determine which constraints or features are lacking in the description of gene models. Finally, decomposing a constraint into simpler ones allows us to better structure our problem and have a better characterization (and therefore understanding) of the structural properties of genes. In the next section, we provide practical examples of intrinsic constraints and composition.

4.4 Applications

In this section, we provide four important examples of intrinsic constraints and use these constraints to show that the new DAG representation is more compact than the NDFSA-based one. First we summarize the results obtained in the previous section for the triplicity constraint. Then we show that the triplicity constraint is a particular case of the more general open reading frame constraint. Next we define an intrinsic constraint, which we name *cardinality constraint*, that places an upper limit on the number of exons and/or introns that can occur in a gene model. Finally, we use the triplicity and cardinality constraints to provide an example of constraint composition.

4.4.1 Triplicity Constraint

We have seen in Section 4.3 that the triplicity constraint is intrinsic, i.e., we can construct from the DAG $G = \Delta(\sigma)$ a new DAG $G^\star$ — denoted $\Delta_{Mod}(\sigma)$ in the following — that contains only the paths of $\Delta(\sigma)$ representing the parses of σ satisfying the triplicity constraint (see Figure 4.9 on page 115). Indeed, finding the parses of σ that satisfy the triplicity constraint is equivalent to finding the paths in $\Delta(\sigma)$ that satisfy the intrinsic constraint $\mathcal{C}_{Mod}$ defined by the system $\langle L_{Mod}, \Lambda_{Mod}, \theta_{Mod}, L^0_{Mod}\rangle$, where:

- $L_{Mod} = \{0, 1, 2\}$.
- $\Lambda_{Mod} = (\odot_{Mod}, \bar{1}_{Mod}, \lambda_{Mod})$ is the L_{Mod}-decoration of $\Delta(\sigma)$ satisfying:
  - $\forall X, Y \subset L_{Mod} :\ X \odot_{Mod} Y = \{(a + b) \bmod 3 \mid a \in X, b \in Y\}$
  - $\bar{1}_{Mod} = \{0\}$
  - For every edge $i \rightarrow j$ in $\Delta(\sigma)$: $\lambda_{Mod}(i \rightarrow j) = \begin{cases} \{Len(i,j) \bmod 3\} & \text{if } j \text{ is a left node} \\ \{0\} & \text{otherwise} \end{cases}$
- $\theta_{Mod} = Id_{L_{Mod}} = (\cup, \emptyset)$.
- $L^0_{Mod} = \{0\}$.

The DAG $\Delta_{Mod}(\sigma)$ is constructed from the DAG $\Delta(\sigma)$ as follows. First, a DAG $\langle \Delta(\sigma), \Lambda_{Mod}\rangle$ is obtained by labeling every edge in the DAG $\Delta(\sigma)$ with the decoration function λ_{Mod}. Every *ll*-edge and *rl*-edge is labeled with a singleton containing the length modulo three of the exon fragment it represents. Every *lr*-edge and *rr*-edge, i.e., every edge representing an intron fragment, is labeled with the set $\{0\}$ to reflect the fact that only the lengths of the exons are relevant. The decoration operator $\odot_{Mod}$ is used to define the label of every path p in $\langle \Delta(\sigma), \Lambda_{Mod}\rangle$ as the singleton containing the total length (modulo three) of the exons in the gene model represented by p. For every path $i_0 \xrightarrow{l_1} i_1 \xrightarrow{l_2} \cdots \xrightarrow{l_m} i_m$ in $\Delta(\sigma)$, we have:

$$\lambda_{Mod}(i_0 \xrightarrow{l_1} i_1 \xrightarrow{l_2} \cdots \xrightarrow{l_m} i_m) = \left\{ \Big(l_1 + l_2 + \cdots + l_m \Big) \bmod 3 \right\}$$

For every node i in the DAG $\langle \Delta(\sigma), \Lambda_{Mod} \rangle$, the summary operator $\oplus_{Mod}$ is used to collect in the set $W_{Mod}(i)$ the labels of the paths starting on node i and ending on the sink s of $\langle \Delta(\sigma), \Lambda_{Mod} \rangle$. Therefore, an element w of $L_{Mod} = \{0, 1, 2\}$ belongs to the set $W_{Mod}(i)$ if and only if there exists a path between i and s that contains exactly w (modulo three) coding bases. As described earlier, the key idea for constructing $\Delta_{Mod}(\sigma)$ is to consider an intermediate DAG $\Delta^{\star}_{Mod}(\sigma)$. The latter is obtained by subdividing every node i in $\langle \Delta(\sigma), \Lambda_{Mod} \rangle$ into the set of subnodes $N_{Mod}(i) = \{(i, a) \mid a \in W_{Mod}(i)\}$. For every edge $i \to j$ in $\Delta(\sigma)$, the subnodes of i are linked to the subnodes of j according to Equation (4.5) on page 117. Note that, since $L_{Mod} = \{0, 1, 2\}$, there cannot be more than three edges between the subnodes of i and the subnodes of j. The paths in $\Delta^{\star}_{Mod}(\sigma)$ that correspond to the parses of σ satisfying the triplicity constraint are those starting on node $(0, 0)$, since by definition, these paths are labeled with the set $L^0_{Mod} = \{0\}$ and they end on the sink node $(s, 0)$.

We now proceed with the complexity analysis of the above construction. The label of every edge $i \to j$ is the singleton $\{0\}$ or the set $\{Len(i, j)\}$, where $Len(i, j) = \pi(j) - \pi(i) + 1$. Therefore the time taken to label a single edge is $T_{\lambda_{Mod}} = O(1)$. The decoration operator $\odot_{Mod}$ takes two subsets of L_{mod} and performs an operation similar to the Cartesian product of these sets. Therefore it takes constant time to apply $\odot_{Mod}$ to two singleton sets: $T_{\odot_{Mod}} = O(1)$. The summary θ_{Mod} is the summary identity, therefore it also takes constant time to summarize a set with a single element: $T_{\oplus_{Mod}} = O(1)$. It follows that the time complexity of the algorithm that constructs $\Delta_{Mod}(\sigma)$ from $\Delta(\sigma)$ is, using equation (4.9) on page 124, $T_{Mod}(n) = O(n|L_{Mod}|)$. Since the number of elements in L_{Mod} is a constant ($|L_{Mod}| = 2$), we have: $T_{Mod}(n) = O(n)$.

We finish this section with an important property that will be used in the next section to define the open reading frame constraint formally. Recall that, for every node i in $\Delta(\sigma)$, $\pi(i)$ denotes the position of the i^{th} splice site in σ. It can be easily shown that every ll-edge or rl-edge $(i, a) \rightarrow (j, b)$ in the DAG $\Delta_{Mod}(\sigma)$ represents the sequence $\sigma_{\pi(i),\pi(j)-1}$ in reading frame $(-a \bmod 3)$. For example, in the DAG $G^{\star} = \Delta_{Mod}(\sigma)$ of Figure 4.9, edge $(4, 0) \xrightarrow{2} (6, 1)$ connecting nodes $(4, 0)$ and $(6, 1)$ represents the sequence $\sigma_{20,27}$ in reading frame 0 whereas edge $(4, 1) \xrightarrow{2} (6, 2)$ represents the same sequence in reading frame 2.

4.4.2 Open Reading Frame Constraint

Let $\Delta_{Stop}(\sigma)$ be the DAG whose paths represent the parses of σ satisfying the open reading frame constraint, which we denote $\mathcal{C}_{Stop}$. In this section, we provide two definitions for $\mathcal{C}_{Stop}$: the first uses a set L_{Stop} containing a large number of elements whereas the second uses a smaller set. We show that the DAG $\Delta_{Stop}(\sigma)$ is significantly smaller than the DAG generated by the NDFSA F_3 introduced in Section 3.3, and that the set of paths satisfying the triplicity constraint is a subset of the set of paths in $\Delta_{Stop}(\sigma)$.

To construct $\Delta_{Stop}(\sigma)$, some bookkeeping is required to keep track of the possible stop codons occurring in (the parses represented by) the paths of $\Delta(\sigma)$. In particular, the label of every path must indicate the reading frame of every stop codon occurring in that path, if there is any. Since the reading frame of a given codon depends on its position in the sequence, the lengths of the exons in the paths of $\Delta(\sigma)$ must also be known. In addition, since a stop codon may be formed whenever two exon fragments (adjacent or not) are concatenated, the prefixes and suffixes of all exons represented by the paths of $\Delta(\sigma)$ must also be stored in the path labels. Furthermore, since each stop codon consists of three nucleotides, a splice can only separate the first or last two bases of a stop codon from the rest; therefore, the prefix and suffix of a sequence only need to contain the first and last two nucleotides of that sequence, respectively.

Formally, let D be the set whose members are the empty sequence (denoted ε) and all possible nucleotide sequences of length 1 or 2. The set D contains 21 elements: 16 sequences of length 2, 4 sequences of length 1, and the empty string. Given a sequence μ, we denote $First(\mu)$ and $Last(\mu)$ the two sequences containing respectively the first two symbols and the last two symbols of μ. If μ contains less than or exactly two symbols, then $First(\mu)$ is the empty sequence and $Last(\mu)$ is the sequence μ itself. We also define $STOP(\mu)$ as the set containing the positions of the stop codons in μ:

$$STOP(\mu) = \{k \bmod 3 \mid \text{a stop codon occurs at position } k \text{ in } \mu\}$$

For any two sequences μ_1 and μ_2, we denote $\mu_1 \circ \mu_2$ the sequence resulting from the concatenation of μ_1 and μ_2. As mentioned above, every edge or path between two nodes i and j in the DAG $\Delta(\sigma)$ is labeled with a 4-tuple $(z, Z_{Stop}, \mu_{First}, \mu_{Last})$ in which:

- z is the length modulo 3 of $\sigma'_{i,j}$, where $\sigma'_{i,j} = \sigma_{\pi(i),\pi(j)-1}$.
- Z_{Stop} is the set containing the reading frames of the stop codons occurring in $\sigma'_{i,j}$.
- μ_{First} is the sequence consisting of the first two symbols of $\sigma'_{i,j}$, or the empty sequence: $\mu_{First} = First(\sigma'_{i,j})$.
- μ_{Last} is the sequence consisting of the last two symbols in $\sigma'_{i,j}$, or the last symbol, or yet the empty sequence: $\mu_{Last} = Last(\sigma'_{i,j})$.

The DAG $\Delta_{Stop}(\sigma)$ is obtained by selecting the full paths of $\Delta(\sigma)$ that have no in-frame stop codons, i.e., those satisfying the constraint $\mathcal{C}_{Stop} = \langle L_{Stop}, \Lambda_{Stop}, \theta_{Stop}, L^0_{Stop}\rangle$, where:

- $L_{Stop} = \{(z, Z_{Stop}, \mu_{First}, \mu_{Last}) \mid z \in \{0,1,2\}, Z_{Stop} \subset \{0,1,2\}, \mu_{First} \in D, \mu_{Last} \in D\}$
- $\Lambda_{Stop} = (\odot_{Stop}, \bar{1}_{Stop}, \lambda_{Stop})$ is the L_{Stop}-decoration of $\Delta(\sigma)$ satisfying:
  - $\forall X, Y \subset L_{Stop}$: $(c, C_{Stop}, \xi_{First}, \xi_{Last}) \in X \odot_{Stop} Y$ if and only if there exists $(a, A_{Stop}, \alpha_{First}, \alpha_{Last}) \in X$ and $(b, B_{Stop}, \beta_{First}, \beta_{Last}) \in Y$ such that:

* $c = (a + b) \bmod 3$.
* $\xi_{First} = First(\alpha_{First} \circ \alpha_{Last} \circ \beta_{First} \circ \beta_{Last})$, $\xi_{Last} = Last(\alpha_{First} \circ \alpha_{Last} \circ \beta_{First} \circ \beta_{Last})$.
* $C_{Stop} = \{(a + b') \bmod 3 \mid b' \in B_{Stop}\} \cup \{(a - Len(\alpha_{Last}) + k) \bmod 3 \mid k \in STOP(\alpha_{Last} \circ \beta_{First})\}$

– $\bar{1}_{Stop} = \{(0, \emptyset, \varepsilon, \varepsilon)\}$

– For every edge $i \to j$ in $\Delta(\sigma)$:

$$\lambda_{Stop}(i \to j) = \begin{cases} \{(Len(i,j) \bmod 3, STOP(\sigma'_{i,j}), First(\sigma'_{i,j}), Last(\sigma'_{i,j}))\} & \text{if } j \text{ is a left node} \\ \{(0, \emptyset, \varepsilon, \varepsilon)\} & \text{otherwise} \end{cases}$$

- $\theta_{Stop} = Id_{L_{Stop}} = (\cup, \emptyset)$.
- $L^0_{Stop} = \{(z, Z_{Stop}, \mu_{First}, \mu_{Last}) \mid z \in \{0, 1, 2\}, 0 \notin Z_{Stop}, \mu_{First} \in D, \mu_{Last} \in D\}$.

The careful reader may have noticed that the elements of L^0_{Stop} whose first component is null can be used to select the parses satisfying both the open reading frame codon and triplicity constraints, i.e., $\mathcal{C}_{Mod \otimes Stop} = \langle L_{Stop}, \Lambda_{Stop}, \theta_{Stop}, L^0_{Mod \otimes Stop} \rangle$, where:

$$L^0_{Mod \otimes Stop} = \{(0, Z_{Stop}, \mu_{First}, \mu_{Last}) \mid 0 \notin Z_{Stop}, \mu_{First} \in D, \mu_{Last} \in D\}$$

Figures 4.12 and 4.13 provide the summaries $W_{Stop}(i)$ obtained for the DAG $\Delta(\sigma)$ in Figure 4.2 (see page 98) when applying equation (4.2) on page 117. For every node i, the set $W_{Stop}(i)$ corresponds to the summary of the labels of the paths between node i and the sink s of $\langle \Delta(\sigma), \Lambda_{Stop} \rangle$. In other words, the summary $W_{Stop}(i)$ contains some 4-tuple $(z, Z_{Stop}, \mu_{First}, \mu_{Last})$ if and only if there exists a path p between node i and the sink of $\langle \Delta(\sigma), \Lambda_{Stop} \rangle$ such that:

- z is the number of coding bases in p.
- Z_{Stop} contains the reading frames of all stop codons occurring in the exons of p.

i	$\lambda_{Stop}(i \to l) \odot W_{Stop}(l)$	$\alpha_{Last} \circ \beta_{First}$	$\lambda_{Stop}(i \to r) \odot W_{Stop}(r)$	$\alpha_{Last} \circ \beta_{First}$
8	$\{(0, \emptyset, \varepsilon, \varepsilon)\}$	ε		
7	$\{(1, \emptyset, at, aa)\}$	$aa \circ \varepsilon = aa$		
6	$\{(2, \emptyset, gt, aa)\}$	$aa \circ \varepsilon = aa$	$\{(1, \emptyset, at, aa)\}$	$\varepsilon \circ at = at$
5	$\{(0, \{2\}, tc, aa)\}$	$cc \circ gt = ccgt$	$\{(1, \emptyset, at, aa)\}$	$\varepsilon \circ at = at$
	$\{(2, \{2\}, tc, aa)\}$	$cc \circ at = ccat$		
4	$\{(0, \{0\}, gt, aa)\}$	$cc \circ at = ccat$	$\{(1, \emptyset, at, aa)\}$	$\varepsilon \circ at = at$
	$\{(1, \{0\}, gt, aa)\}$	$cc \circ gt = ccgt$		
3	$\{(0, \{0\}, gt, aa)\}$	$ca \circ gt = cagt$	$\{(0, \{2\}, tc, aa)\}$	$\varepsilon \circ tc = tc$
	$\{(1, \emptyset, gt, aa)\}$	$ca \circ at = caat$	$\{(1, \emptyset, at, aa)\}$	$\varepsilon \circ at = at$
	$\{(1, \{0\}, gt, aa)\}$	$ca \circ gt = cagt$	$\{(2, \{2\}, tc, aa)\}$	$\varepsilon \circ tc = tc$
2	$\{(0, \{0\}, gt, aa)\}$	$g \circ tc = gtc$	$\{(0, \{2\}, tc, aa)\}$	$\varepsilon \circ tc = tc$
	$\{(1, \{0\}, gt, aa)\}$	$g \circ tc = gtc$	$\{(1, \emptyset, at, aa)\}$	$\varepsilon \circ at = at$
	$\{(1, \{1\}, gg, aa)\}$	$g \circ gt = ggt$	$\{(2, \{2\}, tc, aa)\}$	$\varepsilon \circ tc = tc$
	$\{(2, \emptyset, ga, aa)\}$	$g \circ at = gat$		
	$\{(2, \emptyset, gg, aa)\}$	$g \circ gt = ggt$		
	$\{(2, \{1\}, gg, aa)\}$	$g \circ gt = ggt$		
1	$\{(0, \emptyset, gt, aa)\}$	$gg \circ at = gcat$	$\{(0, \{0\}, gt, aa)\}$	$\varepsilon \circ gt = gt$
	$\{(0, \emptyset, gt, aa)\}$	$gg \circ gt = gcgt$	$\{(0, \{2\}, tc, aa)\}$	$\varepsilon \circ tc = tc$
	$\{(0, \{2\}, gt, aa)\}$	$gg \circ gt = gcgt$	$\{(1, \emptyset, at, aa)\}$	$\varepsilon \circ at = at$
	$\{(1, \{1\}, gt, aa)\}$	$gg \circ tc = gctc$	$\{(1, \{0\}, gt, aa)\}$	$\varepsilon \circ gt = gt$
	$\{(2, \{1\}, gt, aa)\}$	$gg \circ tc = gctc$	$\{(1, \{1\}, gg, aa)\}$	$\varepsilon \circ gg = gg$
	$\{(2, \{2\}, gt, aa)\}$	$gg \circ gt = gcgt$	$\{(2, \emptyset, ga, aa)\}$	$\varepsilon \circ ga = ga$
			$\{(2, \emptyset, gg, aa)\}$	$\varepsilon \circ gg = gg$
			$\{(2, \{1\}, gg, aa)\}$	$\varepsilon \circ gg = gg$
			$\{(2, \{2\}, tc, aa)\}$	$\varepsilon \circ tc = tc$

Figure 4.12: The sets $W_{Stop}(i)$ obtained for the DAG in Figure 4.2. All 4-tuples in row i correspond to the elements of $W_{Stop}(i)$, and are computed from the elements in $W_{Stop}(l)$ and $W_{Stop}(r)$, where l and r are respectively the left and right successors of i. For each element of $W(i)$, the corresponding string $\alpha_{Last} \circ \beta_{First}$ is also indicated.

$W_{Stop}(0)$	$\alpha_{Last} \circ \beta_{First}$	Corresponding Paths
$\{(0, \emptyset, at, aa)\}$	$ct \circ at = ctat$	$0 \to 1 \to 2 \to 5 \to 7 \to 8$
$\{(0, \{0\}, at, aa)\}$	$ct \circ gg = ctgg$	$0 \to 1 \to 2 \to 3 \to 4 \to 6 \to 7 \to 8$
$\{(0, \{0\}, at, aa)\}$	$ct \circ gt = ctgt$	$0 \to 1 \to 3 \to 5 \to 6 \to 7 \to 8$
$\{(0, \{2\}, at, aa)\}$	$ct \circ gt = ctgt$	$0 \to 1 \to 2 \to 3 \to 5 \to 6 \to 8$
$\{(1, \emptyset, at, aa)\}$	$ct \circ gg = ctgg$	$0 \to 1 \to 2 \to 3 \to 4 \to 7 \to 8$
$\{(1, \{0\}, at, aa)\}$	$ct \circ gg = ctgg$	$0 \to 1 \to 2 \to 3 \to 4 \to 6 \to 8$
$\{(1, \{0\}, at, aa)\}$	$ct \circ gt = ctgt$	$0 \to 1 \to 3 \to 5 \to 6 \to 8$
$\{(1, \{1\}, at, aa)\}$	$ct \circ ga = c\underline{tga}$	$0 \to 1 \to 2 \to 3 \to 5 \to 7 \to 8$
$\{(1, \{1\}, at, aa)\}$	$ct \circ gt = ctgt$	$0 \to 1 \to 3 \to 4 \to 6 \to 7 \to 8$
$\{(1, \{1\}, at, aa)\}$	$ct \circ tc = cttc$	$0 \to 1 \to 2 \to 5 \to 6 \to 7 \to 8$
$\{(2, \emptyset, at, aa)\}$	$ct \circ gt = ctgt$	$0 \to 1 \to 3 \to \begin{matrix} 4 \\ 5 \end{matrix} \to 7 \to 8$
$\{(2, \{1\}, at, aa)\}$	$ct \circ tc = cttc$	$0 \to 1 \to 2 \to 5 \to 6 \to 8$
$\{(2, \{1\}, at, aa)\}$	$ct \circ gt = ctgt$	$0 \to 1 \to 3 \to 4 \to 6 \to 8$
$\{(2, \{2\}, at, aa)\}$	$ct \circ gt = ctgt$	$0 \to 1 \to 2 \to 3 \to 5 \to 6 \to 7 \to 8$

Figure 4.13: The set $W_{Stop}(0)$ obtained for the DAG in Figure 4.2.

- μ_{First} consists of the first two symbols of the first exon occurring in p.

- μ_{Last} consists of the last two symbols of the last exon occurring in p.

Note that, because the computation of the sets $W_{Stop}(i)$ is performed bottom-up, the last component μ_{Last} is the same in every 4-tuple, and it is the string containing the last two symbols of σ.

Figure 4.13 provides the set $W_{Stop}(0)$ which contains the summary of the labels of the full paths of $\langle \Delta(\sigma), \Lambda_{Stop} \rangle$. The elements of interest in this set are the 4-tuples whose first component is null as they represent the labels of the paths satisfying the triplicity constraint. In particular, the two sets $\{(0, \emptyset, at, aa)\}$ and $\{(0, \{2\}, at, aa)\}$ of the set $W_{Stop}(0)$ of Figure 4.13 correspond to the labels of two paths with no in-frame stop codons. The first label corresponds to a parse with no (in-frame or out-of-frame) stop codons whereas the second corresponds to a gene with (at least) one stop codon in reading frame 2. In Figure 4.13, the two other paths satisfying the triplicity constraint are labeled with the sets $\{(0, \{0\}, at, aa)\}$ and $\{(0, \{0\}, at, aa)\}$: these paths contain an in-frame stop codon (tga) between positions 23 and 25 (see Figure 4.2).

As an example of a stop codon interrupted by one or more introns, consider the path $0 \to 1 \to 2 \to 3 \to 5 \to 7 \to 8$ in Figure 4.2. This path represents a sequence interrupted by two introns: the first intron starts at position 6 and ends at position 12; it interrupts a stop codon (tga) between its first and second nucleotides. The second intron starts at position 14 and ends at position 34; it interrupts this same stop codon but between its second and third nucleotides. In Figure 4.13, this stop codon has been detected and its reading frame, which is 1, inferred by concatenating the corresponding values of α_{Last} and β_{First} (see the definition of C_{Stop} above). This resulted in the path label $\{(1, \{1\}, at, aa)\}$.

We now show that the construction of DAG $\Delta_{Stop}(\sigma)$ from $\Delta(\sigma)$ is linear in the length of σ. Every edge in $\Delta(\sigma)$ ending on a right node is labeled with the set $\{(0, \emptyset, \varepsilon, \varepsilon)\}$, therefore it takes constant time to label every lr-edge and rr-edge of $\Delta(\sigma)$. In contrast, for every edge ending on a left node, the four following quantities: z , Z_{Stop}, μ_{First} and μ_{Last} need to be known. It should be clear that these four quantities can be computed by simply scanning the *ll*-sequences and *rl*-sequences from left to right. Note that the total length of the sequence obtained by concatenating every *ll*-sequence and *rl*-sequence of σ cannot be greater than $2n$. Also note that $\Delta(\sigma)$ contains exactly s edges of type ll or rl (see Section 4.2.2). Therefore, we will assume that the average length of an *ll*-sequence and *rl*-sequence is $2n/s$, and that every edge of $\Delta(\sigma)$ can be labeled in $T_{\lambda_{Stop}} = 2n/s$ time.

We will also assume that it takes constant time to apply $\odot_{Stop}$ to two singleton sets: $T_{\odot_{Stop}} = O(1)$. The summary θ_{Stop} is the summary identity, therefore it also takes constant time to summarize a set with a single element: $T_{\oplus_{Stop}} = O(1)$. It follows that the time complexity of the algorithm that constructs $\Delta_{Mod}(\sigma)$ from $\Delta(\sigma)$ is, using equation (4.8), $T_{Stop}(n) = O(e(2n/s + |L_{Stop}|))$. In addition, we have seen in Section 4.2.2 that the number of edges e in $\Delta(\sigma)$ is at most $2(s-1)$, therefore we have: $T_{Stop}(n) = O(n|L_{Stop}|) = O(n)$. Note however that the set L_{Stop} contains a large number of 4-tuples: $|L_{Stop}| = 3 \times 2^3 \times 21 \times 21 = 10,584$, since there are 21 elements in D. In the rest of this section, we show how the number of elements in L_{Stop} can be reduced.

First remark that a stop codon cannot be created when appending two exon fragments μ_1 and μ_2 unless the following conditions hold:

- μ_1 ends with t, ta or tg, and,
- μ_2 starts with a or g.

If μ_1 ends with t, then we need to know whether μ_2 starts with one of the dinucleotides: aa, ag or ga. If μ_1 ends with ta or tg, then we need to know whether μ_2 starts with a or g. This means that in the definition of Λ_{Stop}, we can replace the sets $First(\mu)$ and $Last(\mu)$ with the sets $First'(\mu)$ and $Last'(\mu)$ defined as follows:

$$First'(\mu) = \begin{cases} First(\mu) & \text{if } First(\mu) \in \{aa, ag, ga, a, g, \varepsilon\} \\ ax & \text{if } First(\mu) \in \{ac, at, ax\} \\ gx & \text{if } First(\mu) \in \{gc, gg, gt, gx\} \\ xx & \text{otherwise} \end{cases}$$

$$Last'(\mu) = \begin{cases} Last(\mu) & \text{if } Last(\mu) \in \{ta, tg, a, g, \varepsilon\} \\ xt & \text{if } Last(\mu) \in \{at, ct, gt, tt, xt, t\} \\ xx & \text{otherwise} \end{cases}$$

where x is a special "don't care" symbol. The definition of L_{Stop} is the same as the one previously given except that μ_{First} belongs to the set: $\{\varepsilon, a, g, aa, ag, ax, ga, gx, xx\}$, and μ_{Last} to the set: $\{\varepsilon, a, g, ta, tg, xt, xx\}$, so that the number of 4-tuples in L_{Stop} is now $3 \times 2^3 \times 9 \times 7 = 1512$.

If the nodes of the DAG $\Delta(\sigma)$ represent the potential splice sites of σ as defined by the sets $\mathcal{D}$ and $\mathcal{A}$, i.e., respectively the sets of donor and acceptor consensus, then the DAG $\Delta_{Stop}(\sigma)$ represents the same parses of σ as the DAG obtained by unfolding the NDFSA F_3 on σ. Recall that the number of edges in the DAG generated from F_3 is at most ne', where e' is the number of transitions of F_3, i.e., $e' = 21373$. In contrast, the number of edges in the DAG $\Delta_{Stop}(\sigma)$ is at most $2k|L_{Stop}|$, where $|L_{Stop}| = 1512$ and k is the number of occurrences in σ of the patterns from $\mathcal{D} \cup \mathcal{A}$ ($k \leq n$). Besides the significant decrease in the size of the DAGs obtained, the approach described in this chapter exhibits a number of features that are partially missing in the representation by NDFSA. These will be discussed in Section 4.7.

4.4.3 Cardinality Constraint

Let $\Delta_{Card}(\sigma)$ be the DAG that contains only the paths of $\Delta(\sigma)$ satisfying the cardinality constraint, i.e., the paths corresponding to the parses of σ that contain less than or exactly I introns, for some fixed positive integer I. For example, for $I = 2$, this constraint can be used for analyzing yeast genes, as these usually contain only one or two introns.

Note that it is not difficult to design an NDFSA that recognizes all genes containing I introns: this can be done by first designing an NDFSA that recognizes all genes containing a single intron, and then making I copies of this NDFSA. The main disadvantage of this approach is that the number of states and transitions of the resulting NDFSA may be too large to be of practical use. The intrinsic constraint provided below allows one to construct efficiently all parses of a gene containing I or fewer introns. This intrinsic constraint, denoted $\mathcal{C}_{Card}$, is defined by the system $\langle L_{Card}, \Lambda_{Card}, \theta_{Card}, L^0_{Card}\rangle$, where:

- $L_{Card} = \{0, 1, \ldots, I+1\}$.
- $\Lambda_{Card} = (\odot_{Card}, \bar{1}_{Card}, \lambda_{Card})$ is the L_{Card}-decoration of $\Delta(\sigma)$ satisfying:
  - $\forall X, Y \subset L_{Card}: X \odot_{Card} Y = \{S_I(a+b) \mid a \in X, b \in Y\}$ where: $\forall c,\ S_I(c) = \begin{cases} c & \text{if } 0 \leq c \leq I \\ I+1 & \text{otherwise} \end{cases}$
  - $\bar{1}_{Card} = \{0\}$
  - For every edge $i \to j$ in $\Delta(\sigma)$: $\lambda_{Card}(i \to j) = \begin{cases} \{1\} & \text{if } i \to j \text{ is a } lr\text{-edge} \\ \{0\} & \text{otherwise} \end{cases}$
- $\theta_{Card} = Id_{L_{Card}} = (\cup, \emptyset)$.
- $L^0_{Card} = \{0, 1, \ldots, I\}$

Therefore, the construction of $\Delta_{Card}(\sigma)$ is similar to that of $\Delta_{Mod}(\sigma)$. The first step consists of labeling every lr-edge in $\Delta(\sigma)$ with the set $\{1\}$, and every other edge with the set $\{0\}$. The decoration operator $\odot_{Card}$ is used to define the label of every path p in $\langle\Delta(\sigma), \Lambda_{Card}\rangle$ as the set containing the number of introns in (the parse represented by) p. If p contains more than I introns, then this label is the set $\{I+1\}$. The second step then consists of computing the summaries of the labels of the paths ending on the sink s of $\langle \Delta(\sigma), \Lambda_{Card} \rangle$. For every element w of L_{Card} such that $w \leq I$, w belongs to the set $W_{Card}(i)$ if and only if there exists a path between i and s that contains exactly w introns. If $I+1 \in W_{Card}(i)$ then there exists a path between i and s that contains more than I introns (see Figure 4.14). The last step consists of constructing the DAG $\Delta^{\star}_{Card}(\sigma)$, i.e., creating the subnodes corresponding to the elements in $W_{Card}(i)$ and the edges linking these subnodes: if the edge $(i,a) \rightarrow (j,b)$ exists in $\Delta^{\star}_{Card}(\sigma)$, then finding a path with a introns between i and s is equivalent to finding a path with b introns between j and s. Thus, the paths in $\Delta^{\star}_{Card}(\sigma)$ starting on node $(0,i)$ correspond to the parses of σ that have exactly i introns ($i \leq I$) whereas those starting on node $(0, I+1)$ correspond to the parses of σ that have I or more introns.

The complexity analysis of the above construction is similar to that given in Section 4.4.1 for the construction of $\Delta_{Mod}(\sigma)$. The labeling of the edges simply consists of distinguishing the *lr*-edges from the other edges, therefore: $T_{\lambda_{Card}} = O(1)$. The decoration operator $\odot_{Card}$ takes two subsets X and Y of L_{Card} and performs an operation similar to the Cartesian product of these sets. Therefore it takes constant time to apply $\odot_{Card}$ to two singleton sets: $T_{\odot_{Card}} = O(1)$. Finally, the summary operator $\oplus_{Card}$ is the summary identity, therefore: $T_{\oplus_{Mod}} = O(1)$. It follows that the time complexity of the algorithm that constructs $\Delta_{Card}(\sigma)$ from $\Delta(\sigma)$ is, using equation (4.9), $T_{Card}(n) = O(nI)$.

It is reasonable to assume that the simplest NDFSA that describes all gene models containing I introns consists of I copies of the NDFSA in Figure 3.2, and therefore contains $69I$ transitions. The DAG obtained by unfolding this NDFSA on a sequence σ of length n then contains at most $69nI$ edges. In contrast, the DAG $\Delta_{Card}(\sigma)$ contains $2kI$ edges in the worst case, where k is the number of putative splice sites of σ. Since $k \leq n$, this corresponds to a more than 30-fold decrease in the number of edges.

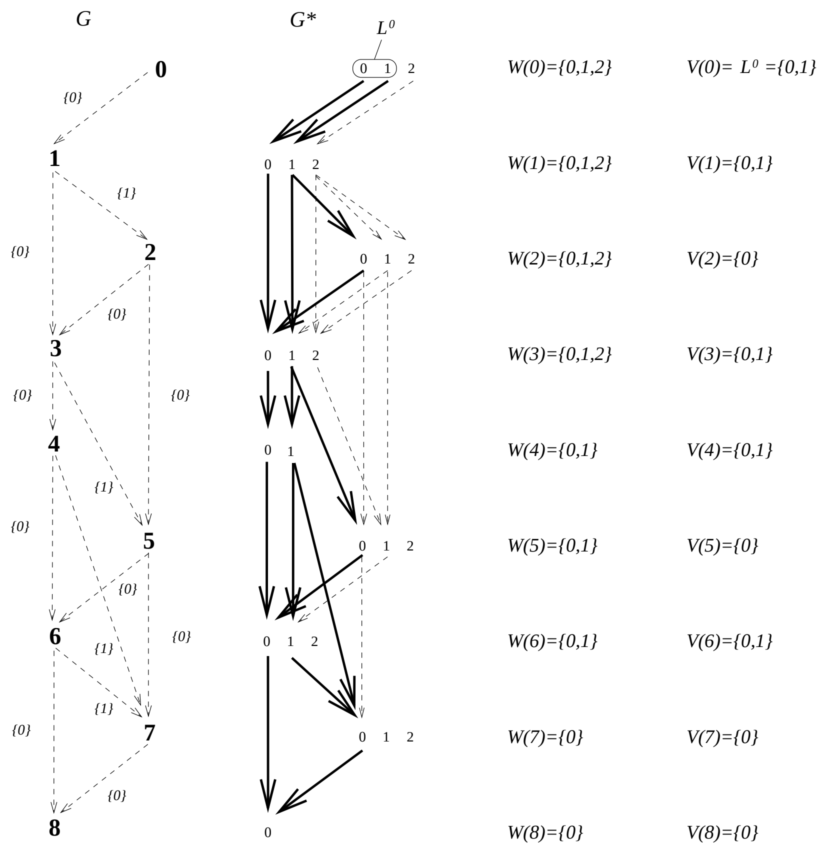


Figure 4.14: The DAGs $G^{\star}$ and G_0 obtained from the DAG G in Figure 4.2 when applying the cardinality constraint (with $I = 1$). G_0 is obtained from $G^{\star}$ by excerpting every path starting on subnode $(0, 0)$ or $(0, 1)$, i.e., by taking $L^0 = \{(0, 0), (0, 1)\}$.

4.4.4 A Simple Example of Composition using the Triplicity and Cardinality Constraints

We now apply the technique described in Section 4.3.6 for composing constraints to define a new intrinsic constraint, denoted $\mathcal{C}_{Mod\otimes Card}$, that selects the paths in $\Delta(\sigma)$ that both satisfy $\mathcal{C}_{Mod}$ and $\mathcal{C}_{Card}$. The constraint $\mathcal{C}_{Mod\otimes Card}$ is defined by the system $\langle L_{Mod\otimes Card}, \Lambda_{Mod\otimes Card}, \theta_{Mod\otimes Card}, L^0_{Mod\otimes Card}\rangle$, where:

- $L_{Mod\otimes Card} = L_{Mod} \times L_{Card} = \{(a_{Mod}, a_{Card}) \mid a_{Mod} \in \{0,1,2\}, a_{Card} \in \{0,1,\ldots,I+1\}\}$
- $\Lambda_{Mod\otimes Card} = (\odot_{Mod\otimes Card}, \bar{1}_{Mod\otimes Card}, \lambda_{Mod\otimes Card})$ is the $L_{Mod\otimes Card}$-decoration of $\Delta(\sigma)$ satisfying:
  - $\forall X, Y \subset L_{Mod\otimes Card}$:

$$X \odot_{Mod\otimes Card} Y = \bigcup_{\substack{(a_{Mod},a_{Card})\in X\\(b_{Mod},b_{Card})\in Y}} (\{a_{Mod}\} \odot_{Mod} \{b_{Mod}\}) \times (\{a_{Card}\} \odot_{Card} \{b_{Card}\})$$

$$= \{((a_{Mod}+b_{Mod}) \bmod 3, S_I(a_{Card}+b_{Card})) \mid (a_{Mod}, a_{Card}) \in X, (b_{Mod}, b_{Card}) \in Y\}$$

    where S_I is as defined in Section 4.4.3.
  - $\bar{1}_{Mod\otimes Card} = \bar{1}_{Mod} \times \bar{1}_{Card} = \{(0,0)\}$
  - For every edge $i \to j$ in $\Delta(\sigma)$:

$$\lambda_{Mod\otimes Card}(i \to j) = \lambda_{Mod}(i \to j) \times \lambda_{Card}(i \to j) = \begin{cases} \{(Len(i,j), 0)\} & \text{if } j \text{ is a } ll\text{- or } rl\text{-edge} \\ \{(0,1)\} & \text{if } i \to j \text{ is a } lr\text{-edge} \\ \{(0,0)\} & \text{if } i \to j \text{ is a } rr\text{-edge} \end{cases}$$

- $\theta_{Mod\otimes Card} = Id_{Mod\times Card} = (\cup, \emptyset)$
- $L^0_{Mod\otimes Card} = L^0_{Mod} \times L^0_{Card} = \{(0,0), (0,1), \ldots, (0,I)\}$

It follows that the label of every path p in $\Delta(\sigma)$ contains both the total length (modulo three) of the exons and the number of introns in p. For every path $p = i_0 \xrightarrow{(l_1,l'_1)} i_1 \xrightarrow{(l_2,l'_2)} \cdots \xrightarrow{(l_m,l'_m)} i_m$ in

$\Delta(\sigma)$, we have:

$$\lambda_{Mod\otimes Card}(p) = \{(x_{Mod}, x_{Card})\} \Leftrightarrow \begin{cases} x_{Mod} = (\sum_{k=1}^{m} l_k) \bmod 3 \\ \\ x_{Card} = S_I(\sum_{k=1}^{m} l'_k) \end{cases}$$

For every node i in $\Delta(\sigma)$, the set $W_{Mod\otimes Card}(i)$ is constructed such that $(a_{Mod}, a_{Card}) \in W_{Mod\otimes Card}(i)$ if and only if there exists a path between node i and the sink that contains both a_{Mod} coding bases (modulo 3) and a_{Card} introns ($a_{Card} \leq I$). Recall that the time taken for constructing $\Delta_{Mod}(\sigma)$ and $\Delta_{Card}(\sigma)$ are respectively $T_{Mod}(n) = O(n|L_{Mod}|)$ and $T_{Card}(n) = O(nI)$. It follows from Equation (4.14) that the algorithm for constructing $\Delta_{Mod\otimes Card}(\sigma)$ has $T_{Mod\otimes Card}(n) = O(n|L_{Mod}|I)$ complexity.

At this point we have shown that every constraint encoded by the NDFSA in Figure 3.5 can be reformulated as an *a priori* or intrinsic constraint with a significant decrease in the size of the DAGs representing gene models. We now turn our attention to *a posteriori* constraints.

4.5 *A posteriori* Constraints

Given a sequence σ, it may be the case that the number of parses of σ satisfying both the *a priori* and intrinsic constraints is still too large to infer its exon-intron structure. The purpose of the last screening set of constraints, i.e., the *a posteriori* constraints, is to select the one or few predicted genes that represent the most likely alternation(s) of exons and introns in σ.

As mentioned in Section 4.1, an example of *a posteriori* constraint is the criterion introduced in the previous chapter for distinguishing exons from introns. Recall that, according to this criterion, a base in non-coding if and only if its probability of being coding is null. In other words, for every edge $i \rightarrow j$ in $\Delta(\sigma)$, the sequence fragment represented by edge $i \rightarrow j$ is non-coding if and only if (a) j is the only successor of i and (b) j is a right node. Therefore, to find the coding and non-coding regions of σ, we simply need to determine, for each node i in $\Delta(\sigma)$, if there exists an *ll*-edge or *rl*-edge starting on i. Interestingly, this can be achieved by defining the intrinsic constraint $\mathcal{C}_{coding} = \langle L_{Coding}, \Lambda_{Coding}, \theta_{Coding}, L^0_{Coding}\rangle$ such that:

- $L_{Coding} = \{ex, in, id\}$.
- $\Lambda_{Coding} = (\odot_{Coding}, \bar{1}_{Coding}, \lambda_{Coding})$ is the L_{Coding}-decoration of $\Delta(\sigma)$ satisfying:
  - $\forall X, Y \subset L_{Coding}: \ X \odot_{Coding} Y = \begin{cases} X & \text{if } X \neq \{id\} \\ Y & \text{otherwise} \end{cases}$
  - $\bar{1}_{Coding} = \{id\}$
  - For every edge $i \to j$ in $\Delta(\sigma)$: $\lambda_{Coding}(i \to j) = \begin{cases} \{ex\} & \text{if } j \text{ is a left node} \\ \{in\} & \text{otherwise} \end{cases}$
- $\theta_{Coding} = Id_{L_{Coding}} = (\cup, \emptyset)$.
- $L^0_{Coding} = \{ex\}$.

Note that the only reason for adding the element id to the set L_{Coding} is to define an identity for $\odot_{Coding}$ as required by axiom (H2).

The interpretation of the constraint $\mathcal{C}_{coding}$ is as follows. Every edge of $\Delta(\sigma)$ is labeled with ex if it represents an exon fragment; otherwise it is labeled with in. The label of a path p starting on a given edge is defined as the label of that edge. In other words, the label of p indicates if p represents a sequence starting with an exon or intron fragment. For every node i in $\Delta(\sigma)$, $ex \in W_{Coding}(i)$ if and only if there exists a path between i and the sink that represents a parse starting with an exon fragment. Similarly, $in \in W_{Coding}(i)$ if and only if there exists a path between i and the sink that represents a parse starting with an intron fragment. In particular, every sequence represented by an edge starting on a node i such that $W(i) = \{in\}$ is non-coding. Since a gene always start with an exon, L^0_{Coding} is defined as the singleton containing ex, so that the DAG $\Delta_{Coding}(\sigma)$ obtained after applying C_{Coding} is identical to $\Delta(\sigma)$. This means that the intrinsic constraint C_{Coding} is only useful when it is composed with other constraints. Figure 4.15 provides the algorithm for detecting the exon and intron fragments

begin

$i \leftarrow 0$ {*source node*}

while $i \neq s$ **do** {*while sink node is not reached*}

  if ex **in** $W(i)$ **then**

    $j \leftarrow Nextleft[i]$

    $\sigma_{\pi(i),\pi(j)-1}$ *is an exon fragment*

  else

    $j \leftarrow Nextright[i]$

    $\sigma_{\pi(i),\pi(j)-1}$ *is an intron fragment*

  $i \leftarrow j$

end

Figure 4.15: Algorithm for applying the criterion.

of σ based on the values of the summaries. It should be clear that this algorithm as well as the algorithm for constructing the DAG $\Delta_{Coding}(\sigma)$ from $\Delta(\sigma)$ have both $O(n)$ time complexity.

4.6 Relationship with other Approaches

We now present a few approaches which on a cursory reading appear similar to the one advocated in this thesis but actually differ in the way gene structures are represented. We also show that, like other existing approaches, we can incorporate statistical information such as coding potential into the DAG, and find the most probable gene model satisfying a given (intrinsic) constraint.

4.6.1 Other Approaches using a DAG

All approaches described in this section but the first one use a DAG, either explicitly or implicitly, to represent gene models. The first approach uses a tree instead of a DAG but is very similar to the other approaches. These trees or DAGs mainly differ by the objects, i.e., splice sites, exons or introns, represented by their nodes or edges.

The first approach is the one proposed in (Guigo et al., 1992). Like most other gene identification programs, it consists of the three following steps:

1. Detection of all potential donor and acceptor sites. This step is performed by using position weight matrices.

2. Determination of all putative exons from the set of predicted splice sites. The large number of putative exons is reduced by calculating (via a number of numerical measures) the coding potential of each predicted exon. Exons that have a coding potential below a given threshold are discarded.

3. Construction of all possible gene models from the set of putative exons. The predicted genes are partitioned according to their initial exon: two predicted genes are in the same subset if and only

if they start with the same initial exon. Each subset S is represented as a rooted tree in which the root node represents the common initial exon of the genes in S, and the internal and terminal nodes represent the internal and final exons of these genes, respectively. There exists an edge from a given node to another if the exon represented by the first node can occur before the exon represented by the second node, e.g., the two exons do not overlap. Each path from the root to a terminal node corresponds to a predicted gene.

A depth-first strategy is used to search the trees for the most probable gene model (with respect to some scoring function). The combinatorial explosion involved in the search of the space of the predicted genes is reduced, but not avoided, by defining the concept of exon equivalence: two exons are defined as being equivalent if they are the roots of the same subtree. In other words, two exons are equivalent if they are the prefixes of two identical parses. Note that the nodes in the trees actually represent exon classes rather than exons.

The approach described in this chapter differs from the above in the following important points:

1. Given a sequence σ, the nodes in the DAG $\Delta(\sigma)$ correspond to the putative splice sites of σ whereas the nodes in the trees represent exons.

2. The edges in the DAG $\Delta(\sigma)$ represent exon and intron fragments whereas the nodes in the trees represent entire exons.

3. In our approach, the set of parses of σ is not represented by a series of trees but by a single DAG. The DAG does not only represent the common prefixes of the parses of σ, but also their common suffixes.

4. The various DAGs introduced in this chapter can be constructed in linear time whereas the trees are obtained in exponential time. This makes the analysis of large sequences impossible.

In (Gelfand and Roytberg, 1993; Gelfand et al., 1996a; Rogozin et al., 1996), the authors also use a DAG to construct gene models. The strategy that they employ is similar to that described above except that the set of all possible gene structures is represented by a DAG instead of a set of trees. The nodes in this DAG represent the putative exons of the sequence being analyzed. Two nodes are linked by an edge if the exons that they represent can occur in the same gene model. The number of edges in this DAG is proportional to the square of the number of nodes. However, the number of paths grows exponentially with the number of nodes. The search of the best gene structure is performed by using a quadratic dynamic programming algorithm. In contrast, the nodes in the DAG $\Delta(\sigma)$ correspond to the putative splice sites of σ and the number of edges grows only linearly with the number of nodes.

In (Snyder and Stormo, 1993; Stormo and Haussler, 1994; Snyder and Stormo, 1995; Wu, 1996), no DAG is explicitly defined but the proposed algorithms can be thought of as constructing a DAG which differs only from the DAG $\Delta(\sigma)$ in that the edges represent entire exons and introns rather than exon and intron fragments. Like the other approaches, every putative exon and intron is assigned a score and the most probable gene structure is determined. These steps are achieved by using a dynamic programming algorithm that runs in quadratic time.

In our view, the most important difference between the approach described in this chapter and the other approaches is the use of intron and exon fragments as the basic processing elements instead of entire introns and exons. By representing splice sites as nodes and fragments between neighboring splice sites as edges, we can describe the set of all gene assemblies by a significantly smaller graph. However, it should be noted that in our approach the edges of the DAG are not labeled with probabilities. We now show that the formalism developed in this chapter can also be used to express statistical properties and thus determine the most probable gene model satisfying a given constraint.

4.6.2 Using Scoring Functions

In Section 2.2, we have seen a number of scoring schemes for measuring how likely a given sequence fragment is coding or non-coding. These scoring schemes are often used in gene structure prediction programs to construct the combination of exons and introns that yield an optimal score. In other words, the problem of determining gene structure is reduced to an optimization problem that can be solved in most cases by dynamic programming techniques. A major drawback of the algorithms using dynamic programming is their quadratic worst-time complexity.

In this section, we show that by assigning probabilities to the edges of the DAG $\Delta(\sigma)$, we can find the most probable combination of exons and introns in some sequence σ in linear time. Note that although in general coding measures are applied to sequences representing complete exons and introns, it is also possible to apply these to exon and intron fragments. This can be done because the majority of the existing coding measures can be computed recursively on their domain, i.e., the score of an exon (or intron) can be derived from the score of its constitutive parts.

To determine the most probable combination of exons and introns of a sequence σ, we can use the intrinsic constraint $\mathcal{C}_{Prob} = \langle L_{Prob}, \Lambda_{Prob}, \theta_{Prob}, L^0_{Prob} \rangle$ defined as follows:

- $L_{Prob} = [0, 1]$
- $\Lambda_{Prob} = (\odot_{Prob}, \bar{1}_{Prob}, \lambda_{Prob})$ is the L_{Prob}-decoration of $\Delta(\sigma)$ satisfying:
  - $\forall X, Y \subset L_{Prob} : \; X \odot_{Prob} Y = \{a.b \mid a \in X, b \in Y\}$
  - $\bar{1}_{Prob} = \{1\}$
  - For every edge $i \rightarrow j$ in $\Delta(\sigma)$:

$$\lambda_{Prob}(i \rightarrow j) = \begin{cases} \{Pr(i \rightarrow j)\} & \text{if } j \text{ is a left node} \\ \{1 - Pr(i \rightarrow j)\} & \text{otherwise} \end{cases}$$

    where $Pr(i \rightarrow j)$ is the probability that sequence $\sigma_{\pi(i),\pi(j)-1}$ is an exon fragment.

- $\theta_{Prob} = (\oplus_{Prob}, \bar{0}_{Prob})$ is the summary over L_{Prob} satisfying:
  - $\forall X, Y \subset L_{Prob} : \; X \oplus_{Prob} Y = \Big\{ \max \{c \mid c \in X \cup Y\} \Big\}$

- $\bar{0}_{Prob} = \emptyset$ (or alternatively: $\bar{0}_{Prob} = \{0\}$).

- $L^0_{Prob} = L_{Prob}$.

Therefore, the most probable path in the DAG $\Delta(\sigma)$ can be determined by first labeling every edge $i \to j$ with $\lambda_{Prob}(i \to j)$, then computing the summary $W_{Prob}(i)$ for every node i. Note that each summary $W_{Prob}(i)$ contains a unique value: a, which represents the probability of the most probable path between i and the sink. If follows that every node i in $\Delta(\sigma)$ has only one corresponding subnode (i, a). In particular, the subnode $(0, a_0)$ obtained at the source indicates the probability of the most probable gene model(s), i.e., a_0. Since a_0 can be any value between and including 0 and 1, the set L^0_{Prob} is chosen to be the interval $[0, 1]$. The most probable gene model(s) are then obtained by constructing the set $V_{Prob}(i)$ for every node i. Note that the definition of $\mathcal{C}_{Prob}$ can be easily modified to obtain the B most probable gene models ($B \geq 1$): one simply needs to store in each summary $W(i)$ the probabilities of the B most probable paths. It can be easily shown that the resulting algorithm has $T_{Prob}(n) = O(nB)$ time complexity.

Note that, in general, scoring functions are not used on their own but in conjunction with other constraints. We now provide an example of constraint that combines $\mathcal{C}_{Mod}$ and $\mathcal{C}_{Prob}$.

4.6.3 Determining the Most Probable Path Satisfying a given Constraint

In this section, we use constraint composition to define an intrinsic constraint that selects the most probable paths in $\Delta(\sigma)$ satisfying the triplicity constraint. The definitions given below can be easily modified to select the most probable paths satisfying any other intrinsic constraint.

The composition of $\mathcal{C}_{Mod}$ and $\mathcal{C}_{Prob}$ results in a new intrinsic constraint denoted $\mathcal{C}_{Mod \otimes Prob}$ which is defined by the 4-tuple $\langle L_{Mod \otimes Prob}, \Lambda_{Mod \otimes Prob}, \theta_{Mod \otimes Prob}, L_{Mod \otimes Prob} \rangle$ such that:

- $L_{Mod \otimes Prob} = L_{Mod} \times L_{Prob} = \{(a_{Mod}, a_{Prob}) \mid a_{Mod} \in \{0, 1, 2\}, a_{Prob} \in [0, 1]\}$

- $\Lambda_{Mod \otimes Prob} = (\odot_{Mod \otimes Prob}, \bar{1}_{Mod \otimes Prob}, \lambda_{Mod \otimes Prob})$ is the $L_{Mod \otimes Prob}$-decoration of $\Delta(\sigma)$ satisfying:

– $\forall X, Y \subset L_{Mod \otimes Prob}$:

$$X \odot_{Mod \otimes Prob} Y = \bigcup_{\substack{(a_{Mod}, a_{Prob}) \in X \\ (b_{Mod}, b_{Prob}) \in Y}} (\{a_{Mod}\} \odot_{Mod} \{b_{Mod}\}) \times (\{a_{Prob}\} \odot_{Prob} \{b_{Prob}\})$$

$$= \{((a_{Mod} + b_{Mod}) \bmod 3, a_{Prob}.b_{Prob}) \mid (a_{Mod}, a_{Prob}) \in X, (b_{Mod}, b_{Prob}) \in Y\}$$

– $\bar{1}_{Mod \otimes Prob} = \bar{1}_{Mod} \times \bar{1}_{Prob} = \{(0, 1)\}$

– For every edge $i \to j$ in $\Delta(\sigma)$:

$$\lambda_{Mod \otimes Prob}(i \to j) = \lambda_{Mod}(i \to j) \times \lambda_{Prob}(i \to j) = \begin{cases} \{(Len(i,j) \bmod 3, Pr(i \to j))\} & \text{if } j \text{ is a left node} \\ \{(0, 1 - Pr(i \to j))\} & \text{otherwise} \end{cases}$$

- $\theta_{Mod \otimes Prob} = (\oplus_{Mod \otimes Prob}, \bar{0}_{Mod \otimes Prob})$ is the summary over $L_{Mod \otimes Prob}$ defined as follows:

  – $\forall X, Y \subset L_{Mod \otimes Prob}$: $\theta_{Mod \otimes Prob}(Z) = \{(x, y) \in Z \mid \forall (x, y') \in Z : y \geq y'\}$

  – $\bar{0}_{Mod \otimes Prob} = \bar{0}_{Mod} \times \bar{0}_{Prob} = \emptyset$.

- $L^0_{Mod \otimes Prob} = L^0_{Mod} \times L^0_{Prob} = \{(0, z) \mid z \in [0, 1]\}$.

The summary function $\theta_{Mod \otimes Prob}$ was obtained by using equations (4.10) through (4.13) as follows:

$$S(Z) = \theta_{Mod}(\{x \mid (x, y) \in Z\}) = \{x \mid (x, y) \in Z\}$$

$$C(Z) = \{(x, y) \in Z \mid x \in S(Z)\} = Z$$

$$\forall x \in S(Z) : \ T_x(Z) = \theta_{Prob}(\{y \mid (x, y) \in C(Z)\}) = \left\{ \max \{y \mid (x, y) \in Z\} \right\}$$

$$\theta_{Mod \otimes Prob}(Z) = \{(x, y) \in Z \mid x \in S(Z), y \in T_x(Z)\} = \{(x, y) \in Z \mid \forall (x, y') \in Z : y \geq y'\}$$

Note again that the above definition can be easily extended to determine the B most probable gene models satisfying the triplicity constraint ($B \geq 1$). In this case, one can also apply the *a posteriori* constraint of Section 4.5 to select a particular combination of exons and introns.

4.7 Discussion and Comparison with the NDFSA-based Approach

In this chapter, we have proposed an alternate representation for describing gene models, and we have shown the effectiveness of the steps undertaken to reduce the potentially large number of edges in the DAG generated by the NDFSA F_3 in Figure 3.5 (see page 64).

Besides this algorithmic improvement, we have shown that the DAG-based representation has more expressive power than the NDFSA-based one. Indeed, the representation introduced in this chapter allows us to formulate constraints that cannot be described by the sole use of NDFSA. For example, we can incorporate coding potential information into the edges of the DAG and determine the most probable path satisfying one or more given constraints. We can also determine which combinations of exons and introns in a given sequence match one or more given proteins. At this point it is important to distinguish between the constraints that can be encoded by an NDFSA but result in an explosion in the number of states and edges, and those that cannot. For example, we have seen in Section 4.4.3 that we can express the cardinality constraint using both representations but a significant reduction in the total number of the edges in the DAG is obtained when using the DAG-based representation.

An important feature of the DAG-based representation is that it allows the user to add new constraints without having to worry about the possible side-effects of this addition. This is because each intrinsic constraint is defined independently from the others. On the contrary, the NDFSA representation combines all constraints at once. As a consequence, one has to make sure that the correctness of the entire NDFSA is preserved when adding new constraints. For example, if a new splice site consensus needs to be added into the definition of F_3, then one has to make sure that the addition of this consensus will not create in-frame stop codons. In other words, with respect to constraint implementation, the DAG representation can be thought of as being incremental. This means that the work required to add an extra constraint to an existing set of constraints is proportional to the complexity of this addition, and not related to the size of the initial set (Jampel, 1995). In contrast, adding an extra constraint when using the NDFSA representation may require as much work as adding all constraints from scratch.

It is important to note that the NDFSA and DAG representations operate at a different level of resolution: the former allows one to describe constraints at the nucleotide level, i.e., by defining transitions on individual symbols, whereas the latter allows one to describe constraints at the exon and intron fragment level, i.e., by defining edges representing sequences of symbols. Thus, a constraint may be easy to express in one representation, but difficult to express in the other. An example of such constraint is the open reading frame constraint. Unlike the triplicity and cardinality constraints, which can be easily stated at the exon-intron level:

"the length of the appended *exons* is divisible by three",

"the number of *introns* does not exceed a given value",

the open reading frame constraint does not only involve the dichotomy between exons and introns but also the specification of information at the nucleotide level:

"no in-frame *stop codons*, i.e., TAA, TAG, TGA, may occur in the concatenated exons".

This difference of resolution in the description of the constraints explains why all constraint but the open reading frame constraint are easier to define at the exon-intron level, i.e., by an intrinsic constraint, than at the nucleotide level, i.e., by an NDFSA. Therefore, a possible line of research would be to find an intermediate representation between NDFSA and DAG. This will be discussed further in the next section.

Aside from the computational aspects, the separation of the constraints into three different classes allows a better distinction between the constraints that are *active* and the constraints that are *passive*. Indeed, an *a priori* constraint such as the $GT - AG$ rule can be thought of as *active* as it describes a property that is (in the vast majority of cases) necessary, but *not* sufficient, for the splicing mechanism to occur, whereas an intrinsic constraint such as the triplicity constraint can be thought of as *passive* as it describes a relation that must be satisfied after intron splicing takes place. In this context, an *a* *posteriori* constraint describes then a heuristic whose biological significance is not completely established. As mentioned in the beginning of this thesis, a major area of study in molecular biology is to understand the mechanism behind splice site recognition. Therefore, an open question is whether we can find a set of *a priori* constraints that are both *necessary* and *sufficient* to characterize the splice sites of a given sequence, i.e., whether we can reformulate both the intrinsic and *a posteriori* constraints as *a priori* constraints.

Although the representation introduced in this chapter is sufficiently powerful for describing the various constraints operating on gene models, we believe that it would greatly benefit from the two following extensions:

- The first extension allows to prune some of the nodes and edges in the DAG $\Delta(\sigma)$ before the summary at the source is computed. It consists of assigning a constraint to every node and edge of $\Delta(\sigma)$. When the summaries are computed, if a constraint for a given node or edge is satisfied, then this node or edge is preserved, otherwise it is deleted. An example of intrinsic constraint that would greatly benefit from this improvement is the constraint $\mathcal{C}_\pi$, which selects all gene models matching a given protein μ (see Section 4.3.4): this extension would then allow to delete every edge that cannot represent a subsequence of μ.
- The second extension consists of defining an operator $\odot'$ and $\oplus'$ for the computation of the sets $V(i)$, and computing summary-like quantities as the paths with a label in L^0 are determined. This extension allows to calculate the probability that each edge is passed though by a path from the source to the sink, and obtain efficiently the coding distribution of the sequence being analyzed.

Finally note that, in addition to constraint composition, one could easily define many other simple operations, such as constraint negation, conjunction and disjunction. One could also define an identity, which selects all paths in the DAG, and an annihilator, which selects none. Although such extensions may be of theoretical interest, at the present time they are not necessary to perform gene structure prediction.

Chapter 5

Conclusion and Future Work

We have proposed a novel, simple and elegant way of predicting gene structure and described two linear time algorithms to detect exons and introns in genes: one using formal languages and the other using graph theory.

The main contribution of this thesis is the innovative use of simple and well-known constraints to describe several important properties of genes. These properties have been first formulated using non deterministic finite-state automata (NDFSA), and then reformulated through three distinct classes of constraints. The NDFSA-based representation is very general: it can be applied to detect features other than exons and introns in genes, and analyze DNA fragments containing multiple genes, as long as those are describable by finite-state automata. In contrast, the DAG-based representation is specifically designed for predicting exons and introns in genes. It is not only more compact than the NDFSA-based representation but it can also express a wider range of constraints. Furthermore, it is both a generalization and an improvement of current algorithms using dynamic programming.

An important original contribution is that unlike other gene structure prediction programs, the algorithms described in this thesis do not use learning methods, sequence comparison, or statistical information to infer gene structure. To quote (Gelfand et al., 1996a), "it is unlikely that algorithms relying on statistical information can be significantly improved". It is also unlikely that learning methods or sequence comparison provide us with some insight about understanding the structure or function of DNA. Instead, our approach consists of using NDFSA or DAGs to define a number of constraints that operate on genes. With more than 65% of exons correctly identified, our approach compares favorably with other gene structure prediction programs that do not use sequence comparison. More importantly, the approach can be refined to achieve higher rates of success.

Not only do the NDFSA and DAG-based approaches provide a simple framework for describing the structural characteristics of protein-coding genes, but also they allow the user to express and evaluate each of the constraints that he or she wants to incorporate in their description of gene models. The main advantage of using constraints is that they can provide the basis for a theoretical explanation of the underlying biochemistry. The results obtained with the set of constraints used in this thesis suggest that introns are confined in the regions where non-determinism is minimal. In other words, we posit that the protein encoded by a given gene can be obtained by considering the longest sequence of concatenated exons that satisfy the triplicity, open reading frame, and other simple constraints set forth by the cellular mechanism or evolution. Although more constraints and more sequences need to be tested to further validate the above hypotheses, we are confident that by progressively fitting the model (NDFSA or DAG) to the data, we can reduce the gap between description and explanation.

The NDFSA-based approach resulted in the development of a program named *DAG-O-Bert* which allows the user to construct and profile NDFSA. Given a high-level, structured description of a non-deterministic finite-state automaton $\mathcal{M}$ and an input sequence σ, the user can obtain and visualize graphically the set of all possible parses of σ. He or she can also assign one or more types (e.g., *coding*, $splice-site$, $branch-point$, and so forth) to every transition of $\mathcal{M}$, and visualize the frequency distribution of each given type (as in Figure 3.4 on page 61). In addition, the user can define one or more selection criteria based on these frequencies and determine which subsequences in the input sequence are of a given type. Finally, he or she can build a database of sequences to be analyzed, and use it to test the correctness of the NDFSA. The program *DAG-O-Bert* and its user manual are both available on the World Wide Web at `http://www.cs.brandeis.edu/~obaby`.

We now describe some possible extensions of this work in the respective areas of gene structure prediction and computational linguistics. First, we propose a number of improvements for increasing the prediction accuracy of the current approach, and address the problem of analyzing sequences containing errors. Then, we discuss the problem of constructing the most compact DAG (i.e., the DAG with the smallest number of edges), given an NDFSA and an input sequence accepted by this NDFSA. Finally, we provide some examples of problems that could be solved using the approach described in this thesis. More particularly, we focus on the problem of automatically restoring accents in French texts.

5.1 Improving Prediction Accuracy

To increase the prediction accuracy of the current approach, we can either attempt to reduce the average number of parses generated by the (NDFSA or DAG-based) algorithm on the input sequences, i.e., define more stringent *a priori* and intrinsic constraints, and/or further process the gene structures predicted after applying the criterion, i.e., define more complex *a posteriori* constraints. In this section, we outline some of the possible future developments in both these directions.

As mentioned in Section 3.5.3, one way to improve prediction accuracy would be to use longer consensus for describing the patterns at the splice sites. However, the main problem of this approach is the risk of overfitting the data. Probably a better way to locate splice sites would be to use hybrid methods combining existing approaches with the one proposed in this work. For example, one could construct position weight matrices and/or calculate via some coding measure the coding potentials of the exon and intron fragments. The main advantages of using coding potentials are (1) the nodes and edges that are unlikely to be passed through by a full path can be removed from the DAG, and (2) the paths in the DAG can be ordered from the most to the least probable one. The latter can be useful for predicting alternative splice sites.

Note that, so far, we have assumed that the splice site consensus are identical for all three reading frames. Another possible extension would be to specify different splice site consensus or weight matrices for each reading frame. Indeed, there is statistical evidence that the reading frame in which splicing is more likely to occur differs from one codon to another (Tomita et al., 1996). Therefore, it would be certainly advantageous to enhance the description of gene models such that the definition of an intron does not depend only on the codon it interrupts but also on the reading frame in which this codon is interrupted.

The specification of branch sites could also be very useful for detecting acceptor splice sites (recall that branch sites are signals that are located upstream of the acceptor splice sites). As mentioned in Section 3.5.3, we expect that adding branch site information, when it becomes available, will result in a significant increase in accuracy. Therefore, future work should include the analysis of the regions that are known to contain the branch sites, i.e., within the range $10 - 50$ bases upstream of the acceptor sites. In the meantime, it may be worth specifying a general (and weak) consensus for the branch site, such as $YYRAY$, where $Y \in \{C, T\}$ and $R \in \{A, G\}$ (Harris and Senapathy, 1990) . Since this consensus occurs in less than 30% of human sequences (Burge, 1997), a reasonable approach would be to define two NDFSA: one that includes the branch site in the description of introns, and the other that does not. One could then refine the consensus by inspecting the sequences that have non-conforming branch sites.

Note that all of the above-mentioned improvements operate only at the level of *a priori* or *intrinsic* constraints. Yet another possibility to increase prediction accuracy would be to define more complex *a posteriori* constraints. In addition to the obvious goal of disambiguating the possibly large number of parses of a given sequence, *a posteriori* constraints should be designed to maximize the ratio of predicted genes that are generated by the NDFSA. Indeed, we have seen in Section 3.3 that 39% of the predicted genes are not generated by the NDFSA, and therefore cannot be (syntactically) correct. Although this ratio is expected to decrease as we consider more (*a priori* and intrinsic) constraints, it is tempting to adopt the following strategy. For some given sequence, let ϕ be the parse obtained after applying the criterion. One could define the final result of gene structure prediction as the parse generated by the NDFSA which is the most similar to ϕ. This strategy, which can be implemented by a simple edit-distance calculation or a more complex Hidden Markov Model, guarantees that most predicted gene structures satisfy the user-defined constraints, and therefore it removes the upper limit on the number of genes than can be correctly predicted (note that, rather than considering the parses generated by the NDFSA, an alternative would be to determine the sequence in the gene database which is the most similar to ϕ).

5.2 Dealing with Sequencing Errors

We have implicitly assumed that the sequence being analyzed is free of errors. The approach proposed in this thesis is likely to be sensitive to sequencing errors, and more particularly indels (nucleotide insertions and deletions), as it relies on constraints that are frame-sensitive (i.e., triplicity and open reading frame constraints). Robustness to sequencing errors is a property that is desired in programs that attempt to maximize prediction accuracy without a special concern about understanding the biochemistry involved. In contrast, since our main guideline is to understand the underlying biology, we feel that it is reasonable to assume that the sequences being analyzed are error-free, i.e., are exactly like those processed by the cell machinery. In other words, dealing with errors may not be desirable if the goal is to understand gene structure, because it complicates the description of sequences, and it obscures the biochemical processes involved. However, since there is no guarantee that a given input sequence is error-free, we suggest that the sequences be corrected before they are submitted for analysis (e.g., by using the programs mentioned in Section 2.2.6).

$$
\begin{aligned}
\langle gene \rangle &::= \langle start\_codon \rangle\ \langle gene\_body \rangle\ \langle stop\_codon \rangle \\
\langle start\_codon \rangle &::= \mathbf{atg} \\
\langle stop\_codon \rangle &::= \mathbf{taa}\ \mid\ \mathbf{tag}\ \mid\ \mathbf{tga} \\
\langle gene\_body \rangle &::= \langle exon\_body \rangle\ \langle left \rangle\ \mid\ \langle exon\_body \rangle \\
\langle right \rangle &::= \mathbf{ag}\ \langle intron\_body \rangle\ \langle right \rangle\ \mid\ \mathbf{ag}\ \langle exon\_body \rangle\ \langle left \rangle\ \mid\ \mathbf{ag}\ \langle exon\_body \rangle \\
\langle left \rangle &::= \mathbf{gt}\ \langle intron\_body \rangle\ \langle right \rangle\ \mid\ \mathbf{gt}\ \langle exon\_body \rangle\ \langle left \rangle\ \mid\ \mathbf{gt}\ \langle exon\_body \rangle \\
\langle intron\_body \rangle &::= \langle base \rangle^* \\
\langle exon\_body \rangle &::= \langle base \rangle^* \\
\langle base \rangle &::= \mathbf{a}\ \mid\ \mathbf{c}\ \mid\ \mathbf{g}\ \mid\ \mathbf{t}
\end{aligned}
$$

Figure 5.1: Regular grammar describing a gene as an alternation of exons and introns.

5.3 Using a more General Representation

It appears that there should be a continuum between the two approaches described in this thesis. The purpose of this section is to provide an initial line of research for future study of the relationship between the NDFSA and DAG-based representations.

Recall that the DAG $\Delta(\sigma)$ introduced in Section 4.2 contains two kinds of nodes corresponding to two distinctive signals: all left nodes but the sink represent potential donor splice sites, and all right nodes but the source represent potential acceptor splice sites. The DAG also contains two kinds of edges: those ending on a left node represent exons whereas those ending on a right node represent introns. Now consider the grammar G in Figure 5.1, which represents the set of all genes satisfying the $GT-AG$ rule. The left nodes and the right nodes of the DAG $\Delta(\sigma)$ in Figure 4.2 can be thought of as representing, respectively, the non-terminals $left$ and $right$ in G. Similarly, the edges representing exon and intron fragments can be thought of as representing the non-terminals $exon\_body$ and $intron\_body$. In addition, the source and the sink of the DAG can be thought of as representing the non-terminals $start\_codon$ and $stop\_codon$.

Yet another possible extension of this work would be to consider a grammar in which not two but three or even more distinctive signals are represented. Each type of signal would then be represented by a different type of node in the DAG. For example, one could devise a DAG with three kinds of nodes: two representing the intron boundaries, and the third representing the branch sites in the sequence being analyzed. A possible generalization of the DAG representation introduced in this thesis could be as follows.

Using the grammar G, let us represent by a tree $T(\sigma)$ the set of all parses of a given input sequence σ (see Figure 5.2). It is easily seen that the DAG $\Delta(\sigma)$ can be obtained from $T(\sigma)$ by repeatedly applying the two following transformation rules (see Figure 5.3):

- Remove duplicate subtrees: if two or more non-terminal nodes are the roots of identical subtrees, then eliminate all but one subtree and redirect all edges pointing to the roots of the eliminated subtrees to the root of the remaining one.
- Remove useless nodes: eliminate every node with a single successor and redirect all edges pointing to the eliminated nodes to their successors.

The above rules are well known in CAD (i.e., computer-aided design) where they are often used to reduce the size of circuits represented by binary decision diagrams (Bryant, 1992). Although it is true in theory that, given any NDFSA, these rules can be used to represent by a compact DAG the set of all parses of an input sequence σ, the time taken to construct the tree $T(\sigma)$ and transform it into the DAG $\Delta(\sigma)$ are both prohibitive in practice since it may be exponential. The algorithm that we have proposed to construct $\Delta(\sigma)$ from σ avoids the construction of the tree and runs in linear time. Therefore, an open problem is whether it is possible to design a linear time algorithm which, given an NDFSA (or any more complex linguistic representation) and an input sequence σ, constructs a DAG representing all parses of σ such that the number of nodes and edges in this DAG are both minimal.

The question also arises as to whether it is possible to express every constraint encoded by NDFSA as an *a priori* or intrinsic constraint. Although this problem is of theoretical interest, it is important

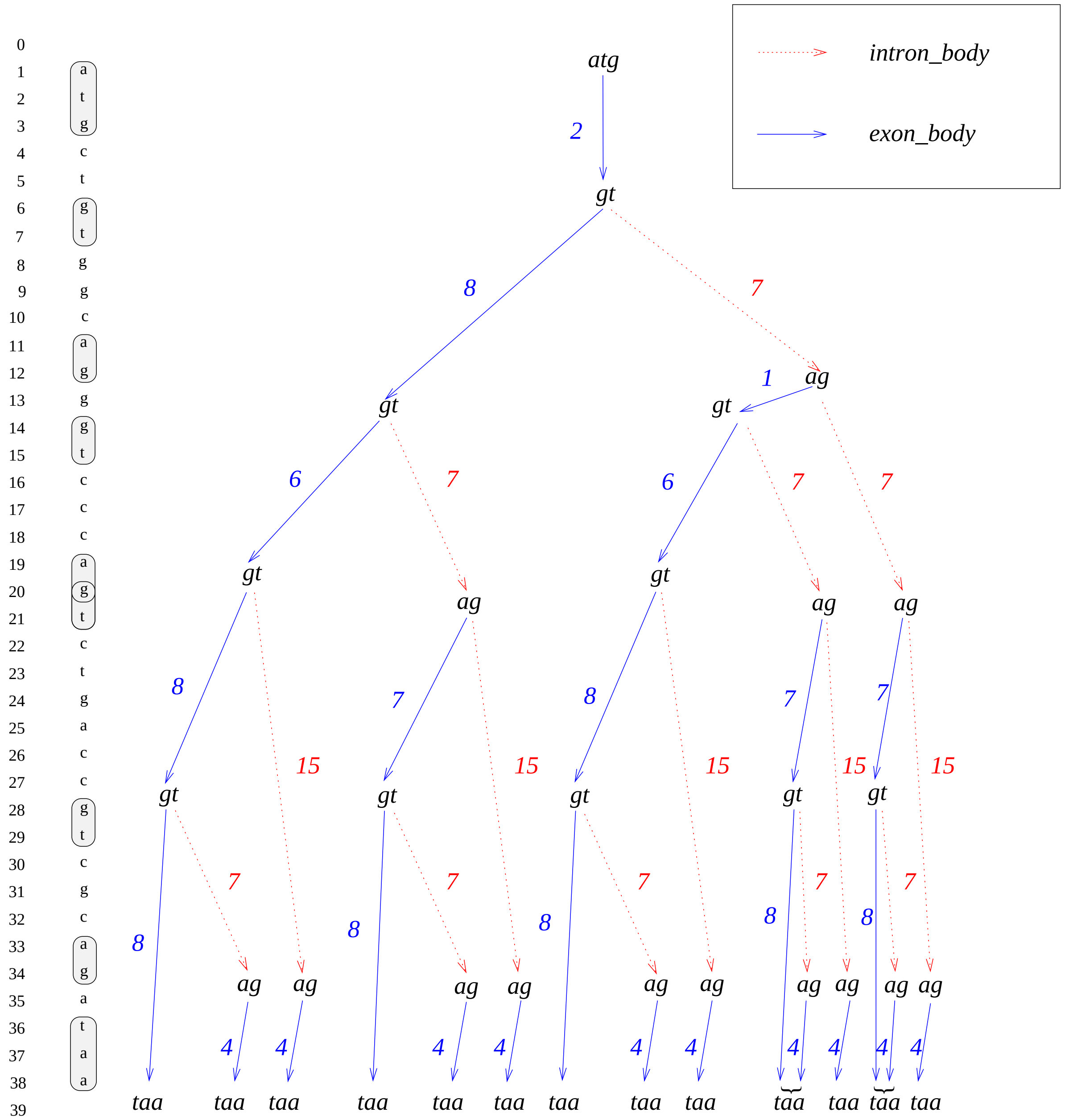


Figure 5.2: Tree representing all parses of the sequence σ in Figure 4.1 when using the grammar in Figure 5.1. Every path between the root of the tree and its leaves represents a parse of σ. For clarity, every edge in the tree is labeled with the length of the sequence fragment it represents.

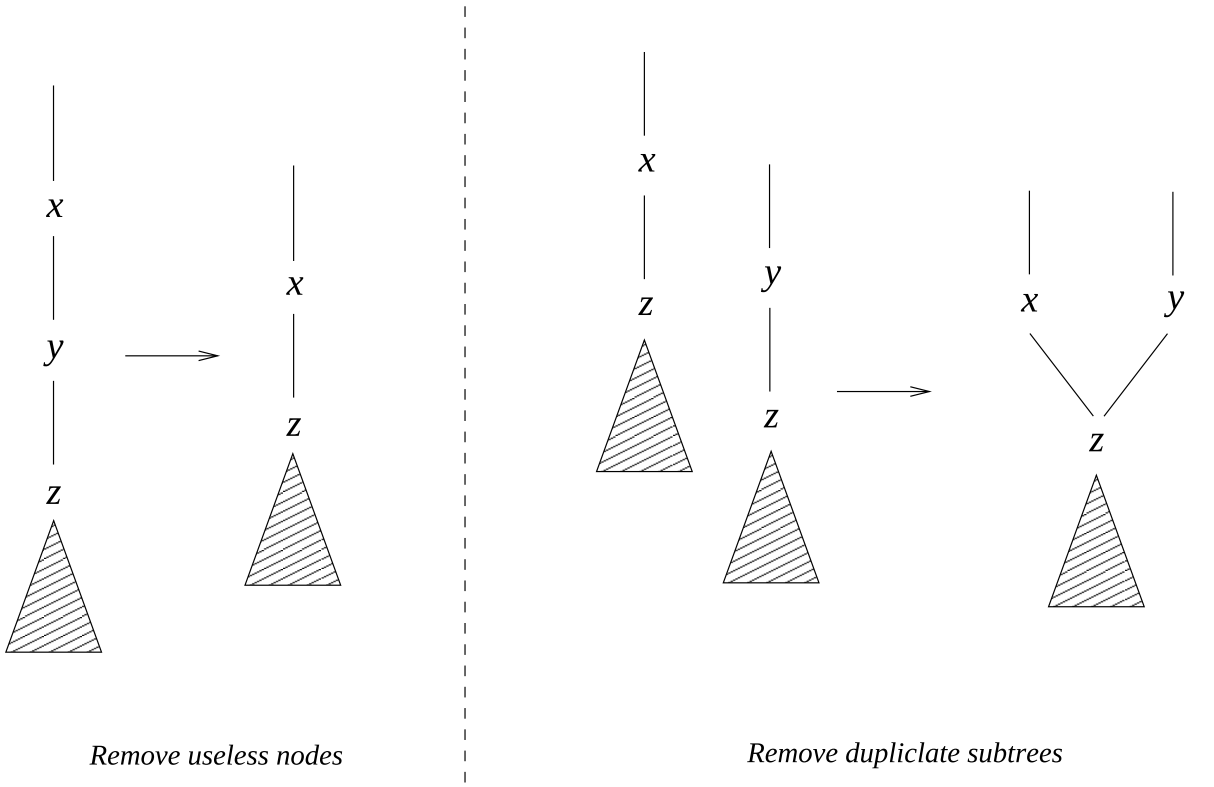


Figure 5.3: The two transformation rules used to obtain the DAG in Figure 4.2 from the tree in Figure 5.2.

to keep in mind that our goal is not to solve a general theoretical problem, but a particular problem involving a finite number of constraints. As noted by Searls in (Searls, 1995b), this "minimalist" approach is motivated not only by the algorithmic difficulties associated with more complex representations (such as context-free grammars or DAGs with more than two types of nodes) but also by "a hope that the search for a formalism with just necessary and sufficient power" will help to elucidate the underlying biochemistry.

It should be also noted that the degree of importance of non-determinism in the proposed algorithms remains to be assessed. In particular, there seems to exist a relationship between the problem of finding the regions of a sequence that have a unique interpretation and that of finding the longest common subsequence of a set of sequences. For example, assume that the sequence being analyzed has two parses. Each intron of the first parse may or may not overlap with an intron of the second parse. In the parse obtained after applying the criterion, the regions of the sequence that are non-coding correspond to the overlaps. More generally, the sequence obtained by concatenating the introns in the predicted gene structure can be thought of as the longest non-coding subsequence that occurs in every parse. Therefore, it might be well possible to predict gene structure without generating all existing parses, by applying an algorithm similar to that used to find the longest common subsequence of a set of sequences.

5.4 Some Other Possible Applications

Aside from the specific goal of gene structure prediction, this work may be found useful in other areas. We conclude this thesis by providing an example of application in the area of natural language processing: the automatic restoration of accents in French texts (Simard, 1996).

The problem of unaccented French texts typically arises in the context of electronic mail (email) messages. People who exchange email in French are often confronted with texts containing characters that should be accented but are not. The reason is that typing accents in an email message is not always an easy task: not only is it often difficult to remember the combination of keys that yield the desired accent mark, but also hitting the right keys simultaneously requires dexterity. The absence of accents in a French text may increase the relative degree of ambiguity inherent to the language. For example, the sentence:

Le salaire diminue a cause des greves

may be interpreted as:

Le salaire diminué a causé des grèves [The decrease in salary has caused strikes]

or:

Le salaire diminue à cause des grèves [The salary is decreasing because of the strikes]

The problem of restoring accents in a text is similar to the problem of determining exons and introns in genes. In particular, it is easily seen that:

1. The process of removing the accents in a given text is deterministic, as each accented letter can correspond to only one unaccented letter. Similarly, the translation of a gene in which the exons and introns are known into a protein is also deterministic, as every codon corresponds to a particular amino acid.

2. As mentioned above, the meaning of an unaccented text is highly ambiguous, as is the interpretation (in terms of protein product) of a gene in which the exons and introns are to be predicted.

Therefore, a possible strategy for disambiguating unaccented French texts would be to define an NDFSA describing all possible ways to restore accents in a given sentence. The transitions of this NDFSA would be labeled by all unaccented letters of the French alphabet, and their type would be any of the accents encountered in French. The set of all parses of a given sentence would then represent all accentuation possibilities for that sentence. A criterion could then be used to generate the most probable meaning(s) of the sentence being analyzed.

Note that the algorithm proposed in (Simard, 1996) to restore the accents in French texts relies on a hidden Markov model. From a broader point of view, it would be worthwhile to investigate the relationship between problems that can be (partially or totally) solved by hidden Markov models, such as part-of-speech tagging or speech processing, and those that can be solved by applying the methodology advocated in this thesis. In our view, it is quite likely that problems that involve a highly ambiguous mapping between one representation to another, such as those mentioned above, could be solved by the approach developed in this thesis.

Appendix A

Glossary

Most of the definitions in this glossary were adapted from (Gusfield, 1997; Griffiths et al., 1993; U.S. Gov., 1988).

Acceptor splice site - 3′ splice site The junction between an intron and an exon at the 3′ end of the intron. Compare donor splice site.

Amino acid The basic building block of proteins. There are 20 common amino acids found in proteins, each of which is coded in DNA by a codon.

Branch point A site located within the intron, generally $20 - 50$ nucleotides upstream from the 3′ splice site, that is necessary for proper splicing of the intron.

Codon A sequence of three nucleotides that codes for a single amino acid.

Coding measure A function that measures the "codingness" of a DNA sequence.

Coding potential A scalar that measures the "codingness" of a DNA sequence.

Coding region A segment of DNA containing the codons that specify a particular protein. Same as Exon.

Consensus sequence The sequence that occurs the most frequently for a given signal (e.g., donor or acceptor splice site).

DNA (deoxyribonucleic acid) The molecule that encodes genetic information. DNA is a double-stranded molecule held together by weak bonds between base pairs of nucleotides. The four nucleotides in DNA contain the bases: adenine (A), guanine (G), cytosine (C), and thymine (T). In nature, base pairs form only between A and T and between G and C; thus the base sequence of each single strand can be deduced from that of its partner. See Nucleotide.

DNA sequence The relative order of base pairs, whether in a fragment of DNA, a gene, or an entire genome.

Donor splice site - $5'$ splice site The junction between an exon and an intron at the $5'$ end of the intron. Compare acceptor splice site.

Exon A segment of DNA containing the codons that specify a particular protein. Compare introns.

Gene A contiguous segment of DNA that codes for a particular protein.

Gene Structure The alternation of exons and introns in a gene.

Genetic code The correspondence between nucleotide triplets (codons) and amino acids.

Genome All the genetic material of an organism.

Introns The DNA base sequences interrupting the protein-coding sequences of a gene; these sequences are transcribed into RNA but are not used to produce proteins. Compare exons.

Nucleic acid A large molecule composed of nucleotide subunits.

Nucleotide A subunit of DNA or RNA consisting of a base (adenine, guanine, thymine, or cytosine in DNA; adenine, guanine, uracil, or cytosine in RNA), a phosphate molecule, and a sugar molecule (deoxyribose in DNA and ribose in RNA). Thousands of nucleotides are linked to form a DNA or RNA molecule. See DNA, RNA.

Open reading frame A sequence of DNA between the start codon (AUG) and the first stop codon (UAA, UAG or UGA) found in the same reading frame as the start codon.

Protein A large molecule composed of one or more chains of amino acids in a specific order; the order is determined by the base sequence of nucleotides in the gene coding for the protein. Proteins are required for the structure, function, and regulation of the body's cells, tissues, and organs, and each protein has unique functions. Examples are hormones, enzymes, and antibodies.

Reading frame One of the three possible ways of reading a nucleotide sequence, since codons are triplets.

RNA (ribonucleic acid) A single stranded nucleic acid similar to DNA but having ribose sugar rather than deoxyribose sugar and uracil rather than thymine as one of the bases. There are several classes of RNA molecules, including messenger RNA, transfer RNA, ribosomal RNA, each of which serves a different purpose. See DNA, Nucleotide.

Sequencing The process of determining the nucleotide sequence of DNA or RNA.

Splice site The junction between an exon and an intron. Compare $5'$ splice site and $3'$ splice site.

Start codon A codon (usually AUG) that signals the start of a sequence to be translated into protein.

Stop codon A codon (usually UAA, UAG or UGA) that signals the end of a sequence to be translated into protein.

Bibliography

Adleman, L. M. 1994. Molecular computation of solutions to combinatorial problems. *Science* 266:1021–1024.

Altschul, S., Gish, W., Miller, W., and Lipman, E. M. D. 1990. Basic Local Alignment Search Tool. *Journal of Molecular Biology* 215:403–410.

Altschul, S. F. 1997. Sequence comparison and alignment. *In* M. J. Bishop and C. J. Rawlings (eds.), DNA and protein sequence analysis: a practical approach, chapter 7. Oxford University Press.

Bach, E., Condon, A., Glaser, E., and Tanguay, C. 1996. DNA Models and Algorithms for NP-complete Problems, chapter Proceedings of the 11th Conference on Computational Complexity. IEEE Computer Society Press.

Barrett, R., Berry, M., Chan, T. F., Demmel, J., nad J. J. Dongarra, J. D., Eijkhout, V., Pozo, R., Romine, C., and van der Vorst, H. A. 1994. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. SIAM.

Barton, G. J. and Sternberg, M. J. 1990. Flexible protein sequence patterns: a sensitive method to detect weak structural similarities. *Journal of Molecular Biology* 212:389–402.

Beckmann, J. S., Brendel, V., and Trifonov, E. N. 1986. Intervening sequences exhibit distinct vocabularies. *Journal of Biomolecular Structure and Dynamics* 4:391–400.

Bentolita, S. 1996. A grammar describing 'biological binding operators' to model gene regulation.

Biochimie 78:335–350.

Berg, O. G. and von Hippel, P. H. 1987. Selection of DNA binding sites by regulatory proteins: I. statistical-mechanical theory and application to operators and promoters. *Journal of Molecular Biology* 193:723–750.

Bertsch, E. and Nederhof, M.-J. 1996. An innovative finite-state concept for recognition and parsing of context-free languages. *In* A. Kornai (ed.), Proceedings of the ECAI '96 Workshop: Extended Finite State Models of Language, Budapest, Hungary.

Blelloch, G. E. 1993. Prefix sums and their applications, pp. 35–60. *In* J. H. Reif (ed.), Synthesis of Parallel Algorithms. Morgan Kaufmann Publishers, San Mateo, CA. Chap. 1.

Borodovsky, M., Koonin, E., and Rudd, K. 1994. New genes in old sequences: A strategy for finding genes in a bacterial genome. *Trends in Biosciences* 19:309–313.

Brave, S. 1996. Evolving deterministic finite automata using cellular encoding. *In* Genetic Programming 1996: Proceedings of the First Annual Conference, pp. 39–44, Stanford University. The MIT Press.

Brāzma, A. and Čerāns, K. 1997. Noise-tolerant efficient inductive synthesis of regular expressions from good examples. *New Generation Computing* 15:105–140.

Brendel, V., Beckmann, J. S., and Trifonov, E. N. 1986. Linguistics of nucleotide sequences: Morphology and comparison of vocabularies. *Journal of Biomolecular Structure and Dynamics* 4:11–21.

Brendel, V. and Busse, H. G. 1984. Genome structure described by formal languages. *Nucleic Acids Research* 12:2561–2568.

Brunak, S., Engelbrecht, J., and Knudsen, S. 1991. Prediction of human mRNA donor and acceptor sites from the DNA sequence. *Journal of Molecular Biology* 220:49–65.

Bryant, R. E. 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys* 24:293–318.

Bucher, P. and Bairoch, A. 1994. A generalized profile syntax for biomolecular sequences motifs and its function in automatic sequence interpretation. pp. 53–61.

Bucher, P., Karplus, K., Moeri, N., and Hofmann, K. 1996. A flexible motif search technique based on generalized profiles. *Computers & Chemistry* 20:2–23.

Buckler, A. J., Chang, D. D., Graw, S. L., Brook, J. D., Haber, D. A., Sharp, P. A., and Housman, D. E. 1991. Exon amplification: a strategy to isolate mammalian genes based on RNA splicing. *Proceedings of the National Academy of Sciences of the USA* 88:4005–4009.

Burge, C. 1997. Identification of Genes in Human Genomic DNA. PhD thesis, Stanford University.

Burset, M. and Guigo, R. 1996. Evaluation of gene structure prediction programs. *Genomics* 34:353–367.

Califano, A. and Rigoutsos, I. 1993. FLASH: a fast look-up algorithm for string homology. *In* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 353–359, New York City, NY.

Cavalier-Smith, T. 1985. Selfish DNA and the origin of introns. *Nature* 135:283–284.

Chambon, P. 1981. Split genes. *Scientific American* 244:60–71.

Cheeseman, P. 1990. On finding the most probable model, pp. 73–95. *In* J. Shrager and P. Langley (eds.), Computational Models of Scientific Discovery and Theory Formation., chapter 3. Morgan Kauffman.

Chow, E. T., Hunkapiller, T., Peterson, J. C., Zimmerman, B. A., and Waterman, M. S. 1991. A systolic array processor for biological information signal processing. *In* Conference Proceedings, 1991 International Conference on Supercomputing, pp. 216–223, Cologne, Germany. ACM SIGARCH.

Claverie, J. M. 1992. Sequence “signals”: Artifact or reality? *Computers and Chemistry* 16:89–91.

Claverie, J. M. and Bougueleret, L. 1986. Heuristic informational analysis of sequences. *Nucleic*

Acids Research 14:179–196.

Claverie, J. M., Sauvaget, I., and Bougueleret, L. 1990. k-tuple frequency analysis: From intron/exon discrimination to T-cell epitope mapping. *Methods in Enzymology* 183:237–252.

Collado-Vides, J. 1989. A transformational grammar approach to the study of the regulation of gene expression. *Journal of Theoretical Biology* 136:403–425.

Collado-Vides, J. 1991a. The search for a grammatical theory of gene regulation is formally justified by showing the inadequacy of context-free grammars. *Computer Application in Biosciences* 7:321–326.

Collado-Vides, J. 1991b. A syntactic representation of units of genetic information. *Journal of Theoretical Biology* 148:401–429.

Collado-Vides, J. 1992. Grammatical model of the regulation of gene expression. *Proceedings of the National Academy of Sciences of the USA* 89:9405–9409.

Collado-Vides, J. 1993. A linguistic representation of the regulation of transcription initiation. *BioSystems* 29:87–128.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. 1990. Introduction to Algorithms, chapter A General Framework for Solving Path Problems in Directed Graphs. The MIT Press - McGraw-Hill Book Company.

Darnell Jr., J. E. 1983. The processing of RNA. *Scientific American* 249:90–100.

Darnell Jr., J. E. 1985. RNA. *Scientific American* 253:68–78.

DOEa 1990. Understanding our genetic inheritance: The U.S. human genome project - the first five years: Fiscal years 1991-1995. DOE/ER-0452P NIH Publication No. 90-1590.

DOEb 1992. Primer on molecular genetics.

Doolittle, R. F. 1985. Proteins. *Scientific American* 253:88–99.

Duret, L., Mouchiroud, D., and Gautier, C. 1995. Statistical analysis of vertebrate sequences reveals that long genes are scarce in CG-rich isochores. *Journal of Molecular Evolution* 40:308–317.

Felsenfeld, G. 1985. DNA. *Scientific American* 253:58–67.

Fichant, G. A. and Quentin, Y. 1995. A frameshift error detection algorithm for DNA sequencing projects. *Nucleic Acids Research* 23:2900–2908.

Fickett, J. W. 1982. Recognition of protein coding regions in DNA sequences. *Nucleic Acids Research* 10:5303–5318.

Fickett, J. W. 1996a. Finding genes by computer: the state of the art. *Trends in Genetics* 12:316–320.

Fickett, J. W. 1996b. The gene identification problem: an overview for developers. *Computers & Chemistry* 20:103–118.

Fickett, J. W. and Tung, C. S. 1992. Assessment of protein coding measures. *Nucleic Acids Research* 20:6441–6450.

Fields, C. A., Pfeiffer, H. D., and Eskridge, T. C. 1991. Knowledge representation and control in "gm1", an automated DNA sequence analysis system based on the MGR architecture. *International Journal of Man-Machine Studies* 34:549–573.

Fields, C. A. and Soderlund, C. A. 1990. gm: a practical tool for automating DNA sequence analysis. *Computer Application in Biosciences* 6:263–270.

Frech, K., Quandt, K., and Werner, T. 1997. Finding protein-binding sites in DNA sequences: the next generation. *Trends in Biosciences* 22:103–104.

Gelfand, M. S. 1990. Computer prediction of exon-intron structure of mammalian pre-mRNAs. *Nucleic Acids Research* 18:5865–5869.

Gelfand, M. S. 1995. Prediction of function in DNA sequence analysis. *Journal of Computational Biology* 2:87–115.

Gelfand, M. S., Mironov, A. A., and Pevzner, P. A. 1996a. Gene recognition via spliced sequence alignment. *Proceedings of the National Academy of Sciences of the USA* 93:9061–9066.

Gelfand, M. S., Podolsky, L. I., Astakhova, T. V., and Roytberg, M. A. 1996b. Recognition of genes in human DNA sequences. *Journal of Computational Biology* 3:223–234.

GELFAND, M. S. AND ROYTBERG, M. A. 1993. Prediction of the exon-intron structure by a dynamic programming approach. *BioSystems* 30:173–182.

GILBERT, W., MARCHIONNI, M., AND MCKNIGHT, G. 1986. On the antiquity of introns. *Cell* 46:151–153.

GRIBSKOV, M. 1992. The language metaphor in sequence analysis. *Computers & Chemistry* 16:85–88.

GRIBSKOV, M., LUTHY, R., AND EISENBERG, D. 1990. Profile analysis. *Methods in Enzymology* 183:146–159.

GRIBSKOV, M., MCLACHLAN, A. D., AND EISENBERG, D. 1987. Profile analysis: Detection of distantly related proteins. *Proceedings of the National Academy of Sciences of the USA* 84:4355–4358.

GRIFFITHS, A. J., MILLER, J. H., SUZUKI, D. T., LEWONTIN, R. C., AND GELBART, W. M. 1993. An Introduction to Genetic Analysis, chapter 13. W. H. Freeman and Company.

GUIGO, R. AND FICKETT, J. W. 1995. Distinctive sequence features in protein coding, genic noncoding, and intergenic human DNA. *Journal of Molecular Biology* 253:51–60.

GUIGO, R., KNUDSEN, S., DRAKE, N., AND SMITH, T. 1992. Prediction of gene structure. *Journal of Molecular Biology* 226:141–157.

GUSFIELD, D. 1997. Algorithms on Strings, Trees, and Sequences, chapter 3. Cambridge University Press.

HARRIS, N. AND SENAPATHY, P. 1990. Distribution and consensus of branch point signals in eukaryotic genes: A computerized statistical analysis. *Nucleic Acids Research* 18:3015–3019.

HARRISON, M. A. 1978. Introduction to Formal Language Theory. Addison Wesley.

HENDERSON, J., SALZBERG, S., AND FASMAN, K. H. 1997. Finding genes in DNA with a hidden markov model. *Journal of Computational Biology* 4:127–142.

HILLIS, W. D. AND STEELE, G. L. 1986. Data parallel algorithms. *Communications of the ACM*

29:1170–1183.

Hopcroft, J. and Ullman, J. D. 1979. Introduction to Automata Theory, Languages, and Computation. Addison Wesley.

Hunter, L. 1993. Molecular biology for computer scientists. *In* L. Hunter (ed.), Artificial Intelligence and Molecular Biology, chapter 1. The MIT Press.

Jampel, M. 1995. A compositional theory of constraint hierarchies (operational semantics). *In* M. Jampel, E. Freduer, and M. Maher (eds.), OCS'95: Workshop on Over-Constrained Systems at CP'95, Cassis, Marseilles.

Kolchanov, N. A., Rogozin, I. B., Kel, A. E., Ponomarenko, M. P., Likchachev, J., and Milanesi, L. 1991. Computer analysis and recognition of human splice sites. *In* S. Mitaku, O. Goto, A. Konagaya, and A. Yonezawa (eds.), Proceedings of Genome Informatics Workshop II, Japan.

Konopka, A. K. and Owens, J. 1990. Complexity charts can be used to map functional domains in DNA. *Gene Analysis Technical Application* 7:35–38.

Krogh, A., Mian, I., and Haussler, D. 1994. A hidden markov model that finds genes in E. *coli* DNA. *Nucleic Acids Research* 22:4768–4778.

Kudo, M., Iida, Y., and Shimbo, M. 1987. Syntactic pattern analysis of 5′-splice site sequences of mRNA precursors in higher eukaryote genes. *Computer Application in Biosciences* 3:319–324.

Kudo, M., Kitamura-Abe, S., Shimbo, M., and Iida, Y. 1992. Analysis of context of 5′-splice site sequences in mammalian mRNA precursors by subclass method. *Computer Application in Biosciences* 8:367–376.

Kulp, D., Haussler, D., Reese, M. G., and Eeckman, F. H. 1996. A generalized hidden Markov model for the recognition of human genes in DNA. *In* D. J. States, P. Agarwal, T. Gaasterland, L. Hunter, and R. F. Smith (eds.), Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology, pp. 134–142. AAAI Press.

LONG, M., SOUZA, S. J., AND GILBERT, W. 1995. Evolution of the intron-exon structure of eukaryotic genes. *Current Opinion in Genetics & Development* 5:774–778.

MARSHALL, E. 1995. Emphasis turns from mapping to large-scale sequencing. *Science* 268:1270–1271.

MCCALDON, P. AND ARGOS, P. 1988. Oligopeptide biases in protein sequences and their use in predicting protein coding regions in nucleotide sequences. *Proteins: Structure, Function, and Genetics* 4:99–122.

MILANESI, L., KOLCHANOV, N., ROGOZIN, I., KEL, A., AND TITOV, I. 1994. Sequence functional inference. *In* M. J. Bishop (ed.), Guide to Human Genome Computing, chapter 8. Academic Press.

MILANESI, L., KOLCHANOV, N. A., ROGOZIN, I. B., ISCHENKO, I. N., KEL, A. E., ORLOV, Y. L., PONOMARENKO, M. P., AND VEZZONI, P. 1993. Genviewer: a computing tool for protein-coding regions prediction in nucleotide sequences. *In* H. A. Lim, J. W. Fickett, C. R. Cantor, and R. J. Robbins (eds.), Proceedings of the Second International Conference on Bioinformatics, Supercomputing, and Complex Genome Analysis, pp. 573–587. World Scientific.

MOUNT, S. M. 1982. A catalogue of splice junction sequences. *Nucleic Acids Research* 10:459–472.

MULLIGAN, M. E., HAWLEY, D. K., ENTRIKEN, R., AND MCCLURE, W. R. 1984. *Escherichia coli* promoter sequence predict in vitro RNA polymerase selectivity. *Nucleic Acids Research* 12:789–800.

Nat.Gen. 1996. To affinity... and beyond. *Nature Genetics* 14:367–370.

OHSHIMA, Y. AND GOTOH, Y. 1987. Signals for the selection of a splice site in pre-mRNA - computer analysis of splice junction sequences and like sequences. *Journal of Molecular Biology* 195:247–259.

PEARSON, W. AND LIPMAN, D. 1988. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the USA* 85:2444–2448.

PEREIRA, F. C. N. AND WRIGHT, R. N. 1997. Finite-state approximation of phrase-structure grammars, pp. 149–173. *In* E. Roche and Y. Schabes (eds.), Finite-state language processing, chapter 5. The MIT Press.

Posfai, J. and Roberts, R. J. 1992. Finding errors in DNA sequences. *Proceedings of the National Academy of Sciences of the USA* 89:4698–4702.

Rabiner, L. R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* 77:257–286.

Rogers, J. H. 1989. How were introns inserted into nuclear genes. *Trends in Genetics* 5:213–216.

Rogozin, I. B., Milanesi, L., and Kolchanov, N. A. 1996. Gene structure prediction using information on homologous protein sequence. *Computer Application in Biosciences* 12:161–170.

Rood, C. M. 1996. Efficient finite-state approximation of context free grammars. *In* A. Kornai (ed.), Proceedings of the ECAI '96 Workshop: Extended Finite State Models of Language, Budapest, Hungary.

Ruby, S. W. and Abelson, J. 1991. Pre-mRNA splicing in yeast. *Trends in Genetics* 7:79–85.

Salomon, P. and Konopka, A. K. 1992. A maximum entropy principle for the distribution of local complexity in naturally occuring nucleotide sequences. *Computers & Chemistry* 16:117–124.

Searls, D. and Dong, S. 1993. A syntactic pattern recognition system for DNA sequences. *In* H. Lim, J. Fickett, C. Cantor, and R. Robbins (eds.), Proceedings of the Second International Conference on Bioinformatics, Supercomputing, and Complex Genome Analysis, pp. 89–101. World Scientific.

Searls, D. B. 1992. The linguistics of DNA. *American Scientist* 80:579–591.

Searls, D. B. 1993. The computational linguistic of biological sequences. *In* L. Hunter (ed.), Artificial Intelligence and Molecular Biology, chapter 2. The MIT Press.

Searls, D. B. 1994. Gene structure prediction by linguistic methods. *Genomics* 23:540–551.

Searls, D. B. 1995a. Formal grammars for intermolecular structure. *In* Proceedings of the First International IEEE Symposium on Intelligence in Neural and Biological Systems, pp. 30–37. IEEE Computer Society Press.

Searls, D. B. 1995b. String variable grammar: A logic grammar formalism for the biological language of DNA. *Journal of Logic Programming* 24:73–102.

Senapathy, P. 1988. Possible evolution of splice-junction signals in eukaryotic genes from stop codons. *Proceedings of the National Academy of Sciences of the USA* 85:1129–1133.

Senapathy, P. 1995. Introns and the origin of protein-coding genes. *Science* 268:1366–1367.

Senapathy, P., Shapiro, M. B., and Harris, N. L. 1990. Splice junctions, branch point sites, and exons: Sequence statistics, identification, and applications to genome project. *Methods in Enzymology* 183:252–278.

Sharp, P. A. 1981. Speculations on RNA splicing. *Cell* 23:643–646.

Sharp, P. A. 1985. On the origin of RNA splicing and introns. *Cell* 42:397–400.

Simard, M. 1996. Automatic restoration of accents in French text. Technical report, Centre d'innovation en technologies de l'information, Laval, Canada.

Singh, R. K., Hoffman, D. L., Tell, S. G., and White, C. T. 1996. BioSCAN: A network-sharable computational resource for searching biosequence databases. *Computer Application in Biosciences* 12:191–196.

Smith, R. 1988. A finite state machine algorithm for finding restriction sites and other pattern matching applications. *Computer Application in Biosciences* 4:459–465.

Snyder, E. E. and Stormo, G. D. 1993. Identification of coding region in genomic DNA sequences: an application of dynamic programming and neural networks. *Nucleic Acids Research* 21:607–613.

Snyder, E. E. and Stormo, G. D. 1995. Identification of protein coding region in genomic DNA. *Journal of Molecular Biology* 248:1–18.

Snyder, E. E. and Stormo, G. D. 1997. Identifying genes in genomic DNA sequences. *In* M. J. Bishop and C. J. Rawlings (eds.), DNA and protein sequence analysis: a practical approach, chapter 11. Oxford University Press.

Solovyev, V. V., Salamov, A. A., and Lawrence, C. B. 1994. Predicting internal exons by oligonucleotide composition and discriminant analysis of spliceable open reading frames. *Nucleic Acids Research* 22:5156–5163.

Souza, S. J., Long, M., and Gilbert, W. 1996. Introns and gene evolution. *Genes to Cells* 1:493–505.

Staden, R. 1984. Computer methods to locate signals in nucleic acid sequences. *Nucleic Acids Research* 12:505–519.

Staden, R. 1990. Finding protein coding regions in genomic sequences. *Methods in Enzymology* 183:163–180.

Staden, R. and McLachlan, A. D. 1982. Codon preference and its use in identifying protein coding regions in long dna sequences. *Nucleic Acids Research* 10:141–156.

States, D. J. and Botstein, S. 1991. Molecular sequence accuracy and the analysis of protein coding regions. *Genetics* 88:5518–5522.

Stormo, G. D. 1990. Consensus patterns in DNA. *Methods in Enzymology* 183:211–221.

Stormo, G. D. and Haussler, D. 1994. Optimally parsing a sequence into different classes based on multiple types of evidence. *In* R. Altman, D. Brutlag, P. Karp, R. Lathrop, and D. Searls (eds.), Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology, pp. 369–375. AAAI Press.

Tomita, M., Shimizu, N., and Brutlag, D. 1996. Introns and reading frames: correlation between splicing sites and their codon positions. *Molecular Biology and Evolution* 13.

Uberbacher, E. C., Einstein, J. R., Guan, X., and Mural, R. J. 1993. Gene recognition and assembly in the GRAIL system: Progress and challenges. *In* H. A. Lim, J. W. Fickett, C. R. Cantor, and R. J. Robbins (eds.), Proceedings of the Second International Conference on Bioinformatics, Supercomputing, and Complex Genome Analysis, pp. 465–476. World Scientific.

U.S. Gov. 1988. Mapping our genes-the genome projects: How big, how fast? OTA-BA-373, Washington, D.C.: U.S. Government Printing Office.

von Hippel, P. H. 1994. Protein-DNA recognition: New perspectives and underlying themes. *Science* 263:769–770.

Weinberg, R. A. 1985. The molecules of life. *Scientific American* 253:48–57.

Wootton, J. C. 1997. Simple sequences of protein and DNA. *In* M. J. Bishop and C. J. Rawlings (eds.), DNA and protein sequence analysis: a practical approach, chapter 8. Oxford University Press.

Wu, T. 1996. A segment-based dynamic programming algorithm for predicting gene structure. *Journal of Computational Biology* 3:375–394.

Xu, Y., Mural, R. J., and Uberbacher, E. C. 1994. Constructing gene models from accurately predicted exons: an application of dynamic programming. *Computer Application in Biosciences* 10:613–623.

Xu, Y., Mural, R. J., and Uberbacher, E. C. 1995. Correcting sequencing errors in dna coding regions using a dynamic programming approach. *Computer Application in Biosciences* 11:117–124.

Yokomori, T. and Kobayashi, S. 1995. DNA evolutionary linguistics and RNA structure modeling: a computational approach. *In* Proceedings of the First International IEEE Symposium on Intelligence in Neural and Biological Systems, pp. 38–45. IEEE Computer Society Press.

Zhang, M. Q. 1997. Identification of protein coding regions in the human genome by quadratic discriminant analysis. *Proceedings of the National Academy of Sciences of the USA* 94:565–568.