An Inductive Logic Programming Approach to Learning which uORFs Regulate Gene Expression

Selpi

PhD 2008

An Inductive Logic Programming Approach to Learning which uORFs Regulate Gene Expression

Selpi

A thesis submitted in partial fulfilment of the
requirements of
The Robert Gordon University
for the degree of Doctor of Philosophy

March 2008

Abstract

Some upstream open reading frames (uORFs) regulate gene expression (i.e., they are functional) and can play key roles in keeping organisms healthy. However, how uORFs are involved in gene regulation is not yet fully understood. In order to get a complete view of how uORFs are involved in gene regulation, it is expected that a large number of functional uORFs are needed. Unfortunately, lab experiments to verify that uORFs are functional are expensive. In this thesis, for the first time, the use of inductive logic programming (ILP) is explored for the task of learning which uORFs regulate gene expression in the yeast *Saccharomyces cerevisiae*. This work is directed to help select sets of candidate functional uORFs for experimental studies.

With limited background knowledge, ILP can generate hypotheses which make the search for novel functional uORFs 17 times more efficient than random sampling. Adding mRNA secondary structure to the background knowledge results in hypotheses with significantly increased performance. This work is the first machine learning work to study both uORFs and mRNA secondary structures in the context of gene regulation. Using a novel combination of knowledge about biological conservation, gene ontology annotations and genes' response to different conditions results in hypotheses that are simple, informative, have an estimated sensitivity of 81% and provide provisional insights into biological characteristics of functional uORFs. The hypotheses predict 299 further genes to have 450 novel functional uORFs. A comparison with a related study suggests that 8 of these predicted functional uORFs (from 8 genes) are strong candidates for experimental studies.

Acknowledgements

I thank God for giving me the chance to study as far as this and for the strength not to give up.

I would like to express my deepest gratitude to my principal supervisor, Dr Chris Bryant, for his advice, constructive feedback, encouragement and patience throughout my PhD studies. I also would like to thank my second supervisor, Dr John McCall, and my advisor until April 2006, Dr Daniel Fredouille, for their advice and valuable discussions.

This thesis has highly benefited from a collaboration with many researchers. Therefore, I extend my thanks to my collaborators: Dr Graham Kemp from Chalmers-Computing Science for initiating the collaboration, as well as for valuable inputs and discussions; Marija Cvijovic from Max Planck Institute and Professor Per Sunnerhagen from Göteborg University for discussions on uORFs; Professor Olle Nerman, for hosting me during a three-month fellowship in Sweden, and Erik Kristiansson, Janeli Sarv and Alexandra Jauhiainen, all from Chalmers-Mathematical Statistics for useful discussions on post-transcriptional regulation and microarray; also to Alex Wilson from RGU-School of Computing for helpful discussions on statistics.

My thanks also goes to: Professor Susan Craw and Dr Ines Arana for their help in their administrative-research roles; all members of CIG Group and to my fellow research students at the CTC for exchange of ideas, support and friendships; IT-support, administrative, library, and janitorial staff, as well as staff at the International Office and Martin Simpson for their assistance on various things over the past few years.

Thanks to Dr Garry Brindley, Dr Bernice West and Professor Maureen Melvin, I have had the pleasure of "working" in a UK University. Thanks also to Dr Charles Juwah for various courses on research skill and personal development.

I am grateful to the following for supporting my PhD studies financially:

RGU-School of Computing for a three-year studentship, my principal supervisor for part of my travel costs, the ILP 2004 Conference for free conference fees, the Integrative Bioinformatics Workshop 2006 for a bursary to present a paper, the EU-EST Marie Curie grant for a three-month research fellowship in Sweden, and the Scottish International Education Trust for a travel grant to attend and present work at ISMB/ECCB 2007 and ISCB-SCS3.

I thank my once neighbours at Cameron House and Indonesian friends for the happy memories in Aberdeen and their friendships.

Most importantly, I am deeply grateful to Ruby, Jim and Graham for always giving me a warm welcome and family comfort, and to my parents and brothers for their love, support, and constant encouragement by any means; all of you make my life more meaningful and cheerful.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the *post-genomics* era, research across engineering and science in the life science interface has grown wider than ever before. Among life sciences, biology is the one which has currently gained the most interest from many computing scientists, mathematicians and statisticians. Many existing and new techniques from computing science, mathematics and statistics have been applied to try to solve, or at least to take steps towards solving, complex biological problems. This area of research is known as *bioinformatics*; this is where the application domain of this thesis lies.

The central information in bioinformatics is *deoxyribonucleic acid* (DNA) sequences. DNA carries a complete set of instructions for making all the proteins a living cell will ever need (see Figure 1.1). A segment of DNA which contains the information for the synthesis of specific *ribonucleic acid* (RNA) and protein is called a *gene*. One of the greatest challenges ever since genes were discovered is to understand the process of *gene expression*, that is to understand how information from the DNA are converted into RNA and protein.

The expression of gene involves two major processes, as shown in Figure 1.1. The first is *transcription*, where information from DNA are copied into RNA. Several kinds of RNA molecules are transcribed, but the ones which carry the specific instructions from DNA for making proteins are called *messenger*

RNA (mRNA). The second process is *translation*, where the information in the mRNA are converted into protein.

Figure 1.1: From DNA to protein.

Different genes are expressed differently in different places, at different times and in different amounts. The mechanisms that control gene expression are known as *gene regulation*. Misregulation of gene expression can cause an abnormality, leading to disease(s) or even cancer (Mata, Marguerat and Bähler, 2005). Therefore, a complete understanding of gene regulation is important; however, this is still far beyond the current knowledge in biology.

This research explores the use of a *machine learning* technique, called *inductive logic programming* (ILP), for a bioinformatics problem in the area of gene regulation, specifically for learning which *upstream open reading frames* (uORFs) regulate gene expression in the yeast *Saccharomyces cerevisiae*.

1.1 Inductive Logic Programming

Inductive logic programming (ILP) is a machine learning technique which deals with the induction of hypothesised predicate definitions of a concept. Machine learning deals with how to construct computer programs which can automatically improve (learn) from experience (Mitchell, 1997). Unlike most machine learning techniques which only learn from *examples* of the concept, ILP is able to bias inference by taking into account *background knowledge*. Background knowledge in ILP is knowledge about the application domain known to the learner before the learning starts. This knowledge is usually extracted from domain experts and/or the literature. Thus, ILP algorithms take examples of

the concept, together with potentially pertinent background knowledge about the concept, and construct a hypothesis which explains the examples in terms of the background knowledge.

The examples, background knowledge and the induced hypotheses in ILP are represented in a declarative manner and so can be easily translated into English. Consequently domain experts can help with the selection and integration of appropriate background knowledge and the final dissemination of discoveries to the wider scientific community.

ILP has been successfully applied to a diverse range of real-world problems. These include problems involving algebra, music, natural language processing, mechanical engineering, as well as biology and chemistry.

1.2 Upstream Open Reading Frame

Upstream open reading frame (uORF) is one of the *regulatory elements* (i.e., elements which can regulate gene expression) that may be present in the $5'$ *untranslated region* (UTR) of mRNA (see Figure 4.5 on page 52). Research has revealed that some transcribed uORFs regulate the translation process (i.e., the uORFs are *functional*) (Vilela and McCarthy, 2003; Vilela, Ramirez, Linz, Rodrigues-Pousada and McCarthy, 1999; Hinnebusch, 1997; Fiaschi, Marzocchini, Raugei, Veggi, Chiarugi and Ramponi, 1997; Iacono, Mignone and Pesole, 2005), while a few others do not (i.e., the uORFs are *non-functional*) (Morris and Geballe, 2000; Krummeck, Gottenöf and Rödel, 1991).

Functional uORFs have been shown to play important roles in keeping organisms healthy, usually by controlling the synthesis of certain proteins which are harmful if over synthesised (Kozak, 1991; Willis, 1999). One example of this is *thrombocythaemia*. Thrombocythaemia is a condition where blood contains too many *platelets*, a type of blood cell involved in blood clotting. People with this condition have a higher risk of developing a blood clot, a stroke or

heart attack. The production of platelets in the blood cells is controlled by the hormone expressed from the gene thrombopoietin. According to Kozak's (1999) review, based on Wiestner, Schlemper, van der Maas and Skoda (1998), under normal conditions, the uORFs of thrombopoietin mRNA act to limit the translation of the thrombopoietin gene and thus limit the production of the platelets in the blood cells. When uORFs are somehow eliminated from the thrombopoietin mRNA, the translation of thrombopoietin gene is increased and thus the amount of the platelets, causing thrombocythaemia.

1.3 Motivation

To date, transcribed uORFs have only been verified in a small number of genes in several organisms. From this data, a partial understanding of how uORFs can regulate protein expression has been achieved. However, as more and more uORFs have been found in the mRNA of genes with critical roles, it has become important to get a complete understanding of how uORFs are involved in the regulatory mechanism of gene expression.

To be able to draw a complete understanding of the mechanism, we would expect that a large number of functional uORFs would be needed. Unfortunately, lab-based experiments to identify functional uORFs are extremely expensive and time-consuming. Therefore, an *in silico* prediction method which can help in selecting sets of candidate functional uORFs for experimental studies is essential.

This PhD project sets out to develop such a method using ILP as the learning technique and the yeast *Saccharomyces cerevisiae* (famously known as baker's and brewer's yeast) as the model organism. Given some data, the method should automatically generate hypotheses which can then be used to predict novel functional uORFs.

Since we want to develop an automated learning method and will deal with

genome data, machine learning is a suitable approach to use. Among many machine learning techniques, we chose ILP for the following reasons. First, ILP provides a richer representation than other machine learning techniques based on attribute-value representation. The latter representation cannot concisely represent the relationships between attributes in the uORF domain. For example, the attribute-value representation cannot concisely represent relationships between uORFs and UTRs because of the arbitrary number of uORFs a UTR may have (see Section 4.2 on page 53). Second, unlike other machine learning techniques, ILP is able to utilise existing knowledge from domain experts and/or the literature; this special feature of ILP is beneficial and will be used in Chapters 5 - 7. Finally, all ILP's input (examples and background knowledge) and output (hypotheses) can be easily translated into English and thus easily understood by human scientists; this is demonstrated, for example, in Table 5.8 on page 76.

The yeast *S. cerevisiae* was chosen as a model organism for several reasons. First is because it is one of the most and best studied biological models. Second, yeast is simpler than other *eukaryotes* (organisms whose cells have nuclei), but it has the characteristics of complex eukaryotes. Third, the size of its genome is relatively small compared to other eukaryotes; this makes yeast more appealing. Last, yeast can be grown very fast making lab experimental studies, to verify whether particular uORFs do indeed regulate translation, feasible.

1.4 Objectives

This goal of this PhD project is to develop an automated learning method, using inductive logic programming (ILP) as the learning method and the yeast *Saccharomyces cerevisiae* as the model organism, to help select sets of candidate functional uORFs for experimental studies. In achieving this goal, the following objectives were set:

1. to investigate whether ILP could automatically generate a model that identifies functional uORFs and whether this model, when used as a filter, could be more efficient than random sampling;

2. to investigate whether adding mRNA secondary structure to ILP background knowledge could increase the performance of the ILP system in recognising known functional uORFs in the yeast *S. cerevisiae*;

3. to investigate whether a combination of knowledge derived from biological sequences of several yeast species, an analysis of several expression data sets and gene ontology annotations could generate hypotheses that identify functional uORFs with good performance;

4. to investigate whether the performance of the hypotheses generated by an ILP system, CProgol4.4, depends on the ordering of input positive training examples.

1.5 Contributions to Knowledge

Unless stated otherwise, the work described in this thesis is solely the author's work. Some ideas used and developed in this thesis may have arisen from discussion with the author's PhD supervisors and collaborators.

The collaborators include several senior researchers: Dr Graham J.L. Kemp[1], Professor Per Sunnerhagen[2] and Professor Olle Nerman[3]; as well as several fellow PhD students: Marija Cvijovic[4], Erik Kristiansson[3], Alexandra Jauhiainen[3] and Janeli Sarv[3].

This thesis makes original contributions to knowledge in the field of machine

[1]Department of Computer Science and Engineering, Chalmers University of Technology, Sweden.

[2]Department of Cell and Molecular Biology, Göteborg University, Sweden.

[3]Department of Mathematical Statistics, Chalmers University of Technology, Sweden.

[4]Max Planck Institute for Molecular Genetics, Berlin, Germany.

learning, particularly ILP, and bioinformatics. The main contribution to ILP is:

- The first empirical study to show that there is a dependency between the performances of the hypotheses constructed by CProgol4.4, an ILP system which uses the covering approach, and the orderings of positive training examples.

The main contributions to bioinformatics are:

- An automated learning method for the task of learning which uORFs regulate gene expression in the yeast *S. cerevisiae*. The method can help with the selection of sets of candidate functional uORFs for experimental studies;
- Putative novel functional uORFs and a set of strong candidate functional uORFs for lab experimental studies;
- Provisional insights into the biological characteristics of functional uORFs in the yeast *S. cerevisiae*.

Part of the work described in this thesis (Chapter 5) has been published in:

> Selpi, Bryant, C.H., Kemp, G.J.L. and Cvijovic, M. (2006). A First Step towards Learning which uORFs Regulate Gene Expression. *Journal of Integrative Bioinformatics* **3**(2):31.

The data and other supplementary materials for this paper are available at `http://www.comp.rgu.ac.uk/staff/chb/research/data_sets/jib2006/uORF/`.

1.6 Thesis Structure

This thesis is written from a bioinformatics point of view, with the following readers in mind: computing scientists with interest in biology and bioinformaticians. The thesis is structured as follows.

Foundation chapters:

- Chapter 2 provides an overview of inductive logic programming (ILP), the chosen machine learning technique. Basic ILP terminology is introduced. Why ILP was chosen and how it can address the challenging task in bioinformatics are explained. This chapter also describes the ILP systems which will be used in this research;

- Chapter 3 briefly introduces the field of bioinformatics and reviews some of machine learning (particularly ILP) work on bioinformatics;

- Chapter 4 introduces a new bioinformatics problem for ILP and machine learning generally, namely learning which uORFs regulate gene expression. The roles of uORFs in gene regulation and in keeping organisms healthy are explained. A challenge facing biologists working in the uORF domain is identified which provides the motivation for the research described in the experimental chapters described next.

Experimental chapters:

- Chapter 5 describes our first exploration of using an ILP system to learn which uORFs regulate gene expression in the yeast *Saccharomyces cerevisiae*. This chapter shows that ILP has the potential to help in selecting sets of candidate functional uORFs for wet-experimental studies;

- Chapter 6 investigates whether adding mRNA secondary structure to background knowledge could increase the performance of an ILP system, CProgol4.4, in recognising known functional uORFs in the yeast *S. cerevisiae*. This is the first machine learning work to study mRNA secondary structures and uORFs together in the context of gene regulation. This chapter also investigates whether the performance of the hypotheses generated by CProgol4.4 depends on the ordering of positive training examples;

- Chapter 7 presents a new and finer approach to learning yeast functional uORFs. The shortcomings from the previous two chapters are addressed in this chapter.

Final chapter:

- Chapter 8 draws conclusions. Original contributions to knowledge and ideas for future research are also presented.

Chapter 2

An Overview of Inductive Logic Programming

This chapter gives an overview of inductive logic programming (ILP), the chosen machine learning technique used in this research. The aims of this chapter are to introduce the basic ILP terminology and framework which will be used throughout this thesis and to show why ILP was chosen and how it can address the challenging tasks in bioinformatics, which is the application domain of this research. Thus, this chapter only covers a small part of ILP, mainly the part needed to carry out this research.

2.1 Basic ILP

Inductive logic programming (ILP), as a scientific discipline, has grown fast since it emerged in the early 1990s. As a research area, ILP lies at the intersection of machine learning and logic programming (Muggleton, 1991).

Machine learning (ML) deals with how to construct computer programs which can automatically improve (learn) from experience (Mitchell, 1997). The learning process may involve deductive, inductive and abductive reasoning (inference). The difference between these three types of reasoning is described

briefly in the following ('$\rightarrow$' denotes implication from left to right).

Deductive

$X \rightarrow Y$	*Example* :	All vertebrates have backbones;
X		Dogs are vertebrates;
Y		Therefore dogs have backbones.

If the premises are true then the conclusion must be correct. A deductive argument is a valid argument.

Inductive

$X_1 \rightarrow Y_1$	*where* :	X_i is an instance of X and
$X_2 \rightarrow Y_2$		Y_i is an instance of Y.
$X_3 \rightarrow Y_3$		
$X_n \rightarrow Y_n$	*Example* :	All cats that we have seen are black;
$X \rightarrow Y$		then we may infer that all cats are black.

Abductive

	Example :	If a patient is affected by a beta thalassemia,
$X \rightarrow Y$		his/her level of hemoglobin A2 is increased;
Y		Ann's level of hemoglobin A2 is increased;
X		We can infer that Ann is affected by a beta thalassemia.

Although induction and abduction are not sound, both are useful in the process of learning.

Logic programming is a declarative, relational style of programming based on first-order logic (Kowalski, 1999). In logic programming, logic is used to represent programs and deduction is used to execute the programs.

Some of these characteristics of both machine learning and logic programming are inherited by ILP, and both deductive and inductive inference are used. As its name suggests, ILP generally uses inductive inference to generate

hypotheses from *observations* (also called *examples*). Its engine uses deductive inference to compute logical consequence, and is often implemented in *Prolog* (Clocksin and Mellish, 1981), which is a standard language used in logic programming.

In order to understand basic ILP terminology, one needs to know the *alphabet* of first-order logic. The alphabet consists of *variables*, *functors* and *predicate symbols* (Lavrač and Džeroski, 1994). Each functor and predicate symbol has an *arity* (number of arguments) ≥ 0. A functor with $arity = 0$ is called a *constant*. A *term* is either a variable, a constant, or an expression $f(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms and f is an n-ary functor. An *atomic formula* (or simply *atom*) is expressed as $p(t_1, \ldots, t_n)$, with $t_1, \ldots, t_n$ are terms and p is an n-ary predicate symbol. An atom or a term with no variable is called *ground*. The alphabet also consists of *logical connectives* and *quantifiers*. Logical connectives are '$\wedge$' (conjunction), '$\vee$' (disjunction), '$\neg$' (negation), '$\leftarrow$' (implication from right to left) and '$\leftrightarrow$' (logical equivalence). Quantifiers are '$\forall$' (universal) and '$\exists$' (existential).

Here, the basic terminology used in logic programming is described briefly. A *clause* is a formula

$$\forall(L_1 \vee \ldots \vee L_n)$$

where each L_i is either a *positive literal* (i.e., an atom) or a *negative literal* (i.e., a negation of an atom). Thus, a clause can also be written as

$$\forall(A_1 \vee \ldots \vee A_m \vee \neg A_{m+1} \vee \ldots \vee \neg A_{m+n})$$

or equivalently

$$\forall((A_1 \vee \ldots \vee A_m) \leftarrow (A_{m+1} \wedge \ldots \wedge A_{m+n}))$$

where each A_i is an atom, $m \geq 0$ and $n \geq 0$. *Horn clauses* are clauses where $0 \leq m \leq 1$ and $n \geq 0$. A Horn clause with $m = 1$ is called a *definite clause* and a Horn clause with $m = 0$ is called a *goal*.

ILP systems mostly deal with Horn clauses. From here, Prolog syntax is used for describing ILP terminology. A definite clause is written in Prolog in the form of

$$A{:-}B_1, \ldots, B_n.$$

where $n \geq 0$. The ':−' sign is pronounced *if* and means implication from right to left. A is called the *head* of the clause. Altogether, the B_i form the *body* of the clause. When $n > 0$, the clause is called a *rule*, and when the body of the clause is empty ($n = 0$), the clause is called a *fact* or a *unit clause* (in this case, the ':−' sign is omitted). A finite set of definite clauses makes up a *definite logic program*.

ILP uses inductive inference to generate a predicate definition of a *target concept* (namely the *hypotheses*). A *predicate definition* is a set of clauses with the same predicate symbol and arity in their heads. ILP commonly learns from classified examples of a target concept. These examples are also called *training examples*. The instances which belong to the concept are called *positive examples*, while instances which do not belong to the concept are called *negative examples*.

Unlike most ML techniques which learn hypotheses from examples only, ILP also takes into account background knowledge (Lavrač and Džeroski, 1994). *Background knowledge* in ILP is knowledge about the application domain known to the learner before the learning starts. This knowledge is usually extracted from domain experts and/or the literature (Srinivasan, 2001). Represented as a set of predicates, background knowledge can be described in an *intensional* and/or *extensional* manner (Lavrač and Džeroski, 1994). Knowledge is described extensionally by listing all the facts (or the unit clauses) of the background predicates. However, when the number of unit clauses are large, an extensional definition can be unsuitable. In that case, an intensional definition, represented as rules, is preferred as it is more compact and concise. In most ILP systems, the input (examples and background knowledge) and the

output (hypotheses, in a form of rules set) are all represented as definite logic programs.

As in general ML, a learning task in ILP can be formulated as a search problem in a predefined space of potential hypotheses for the best hypothesis that fits the training examples (Mitchell, 1997). The way the hypotheses space is defined and searched is controlled by giving *declarative bias* (Lavrač and Džeroski, 1994). There are two different kinds of bias known in ILP; they are *syntactic bias* (also called *language bias*) and *semantic bias* (Muggleton and Raedt, 1994).

Syntactic bias restricts the syntax (the form) of clauses allowed in the hypotheses. Often this is done by giving a set of parameters, such as the maximum number of variables allowed in a clause and the maximum number of literals allowed in a clause. Semantic bias defines the form of the hypotheses themselves and usually given by declaring *types* and *modes* (Muggleton and Raedt, 1994). Some practical information about bias will be discussed in more detail later on. However, the underlying theory of bias will not be covered in this thesis, but is available from Muggleton and Raedt (1994).

2.2 Issues and Developments in ILP

This section briefly describes issues and developments in ILP to provide basic information of what ILP can and cannot do, as part of consideration when choosing this technique. The evaluation of these issues and developments in terms of application to biology and chemistry related problems is detailed in Chapter 3.

2.2.1 Positive-only Learning

ILP was originally designed to deal with binary classification tasks. This inductive learner generalises observed training examples by identifying features

that empirically distinguish positive from negative training examples. Hence, ILP generally requires a number of positive and negative training examples. However, there are some application domains where negative examples do not exist, or are scarce. For example, grammar learning in natural language processing is not concerned with distinguishing positive examples from negative ones (Džeroski, Cussens and Manandhar, 2000). It is only concerned with the positive examples (in this case, the sentences). This kind of situation has driven the development of *positive-only* learning (Muggleton, 1996).

While there have been many applications of the positive-only setting of ILP to natural language processing (see a workshop series Learning Language in Logic (LLL) in 1999, 2000, 2001 and 2005), applications of this setting to real-world biological problems are still rare. In Chapters 5 - 7, the use of positive-only setting of ILP on a bioinformatics problem, namely identifying functional upstream open reading frames (uORFs), is explored.

2.2.2 Multi-class Problems

ILP was designed to deal with genuine binary problems and problems which concerned with discriminating one class from the rest of the classes. However, some real-world problems can involve more than two classes (they are called *multi-class* problems) with each of the classes are equally important. The fact that ILP was designed to induce binary classifiers makes applying ILP to a multi-class problem is not a straightforward task.

A popular ML approach in dealing with multi-class (n-class, $n > 2$) is by reducing the n-class problem into multiple binary problems. At least there are two ways to do this. First is by treating one class as positive and the rest as negative and repeating this process for each class and thus creating n binary problems (Allwein, Schapire and Singer, 2000). Second is by considering a distinct pair of classes each time and ignoring the rest of the classes; this way an n-class problem is broken down into $\binom{n}{2}$ binary problems (Hastie and

Tibshirani, 1998; Fürnkranz, 2001; Allwein et al., 2000). Other approaches can be read from Dietterich and Bakiri (1995) and Allwein et al. (2000). An example of ILP application which uses the class-reduction method is Quiniou, Cordier, Carrault and Wang (2001).

2.2.3 Incorporating Abduction

The accuracy of inductive learning depends on availability of data (background knowledge and training examples). Due to this fact, different lines of research have been carried out among the ILP community to learn from incomplete data. Among those, some use an abductive approach, which then leads to a new research area called Abductive Logic Programming (ALP; details can be read from Kakas, Kowalski and Toni (1998)), and others combine abduction and induction for extended ILP systems (Kakas and Riguzzi, 1997; Mooney, 1997; Tamaddoni-Nezhad, Chaleil, Kakas and Muggleton, 2006).

When combined with induction, abduction is used to supplement incomplete background knowledge by making assumptions about missing facts in the background knowledge (Riguzzi, 1998) and/or to modify the existing background knowledge (in the sense of theory refinement (Mooney, 1997)). Abductive assumptions or explanations can then be generalised by induction. In such a way, induction and abduction strengthen each other to learn from incomplete data.

2.3 Why Choose ILP

Like other techniques, ILP has plus and minus points. However, for the purpose of our research (i.e., to develop an automated learning method which can help in selecting sets of candidate functional uORFs), we believe ILP, with its characteristics, has potential.

ILP provides a richer representation than other machine learning techniques

based on attribute-value representation. Attribute-value learners cannot concisely represent the relationships between attributes in the uORF domain, the specific application domain of this research. For example, the attribute-value learners cannot concisely represent relationships between uORFs and UTRs because of an arbitrary number of uORFs a UTR may have. uORFs, UTRs and their relationships will be discussed in more detail in Section 4.2 on page 53.

This research builds on previous (biological) research. Therefore, ILP's ability to utilise existing knowledge from domain experts and/or the literature is beneficial. The positive-only learning of ILP is certainly an advantage, especially for domain where negatives are difficult to get. As demonstrated, for example, in Table 5.8 on page 76, the declarative representations of ILP's input and output can be easily translated into English and thus can be easily understood by human scientists.

Moreover, ILP has been successfully applied to a diverse range of real-world problems, including those involving algebra, music, natural language processing, mechanical engineering, as well as biology and chemistry. The applications of ILP to biology and chemistry related problems will be reviewed in Chapter 3, as they are more related to bioinformatics, the application domain of this research. Other applications of ILP will not be reviewed here, but are available from Bratko and Muggleton (1995), Muggleton (1999) and Džeroski (2001).

2.4 ILP Systems

Since the early 1990s, many ILP systems have been built. Two of them, which are used in this research, are discussed in this section. A review of the early ILP systems FOIL, GOLEM, and MOBAL is available from Lavrač and Džeroski (1994, Chapter 4) and Muggleton and Raedt (1994). mFOIL, which uses FOIL approach with heuristics search and stopping criteria to provide a better noise-

handling, is described in Lavrač and Džeroski (1994, Chapter 9). Other ILP systems, especially those related with knowledge discovery in databases, are reviewed in the chapters of Džeroski and Lavrač (2001).

2.4.1 CProgol

The main reasons why CProgol (Muggleton, 1995) was chosen are because it is an established ILP system and it has been successfully applied to many different problems, including those in bioinformatics. Although it was written in C (Kernighan and Ritchie, 1988) programming language, CProgol uses Prolog (Clocksin and Mellish, 1981) to represent its input (examples and background knowledge) and output (hypotheses in a form of rules set).

Several versions of CProgol[1] are available; they are CProgol4.1 (Muggleton, 1995), CProgol4.2 (Roberts, 1997), CProgol4.4 (Muggleton and Firth, 2001), CProgol4.5 and CProgol5.0 (Muggleton and Bryant, 2000). The positive-only setting is implemented in version 4.2 and 4.4. Abductive capability is incorporated in version 5.0.

Related with CProgol, there are also other ILP systems using *Progol-like* algorithm, such as PProgol[2], Indlog (Camacho, 1994) and Aleph (Srinivasan, 1999). Aleph is a further development of PProgol. These three systems were written in Prolog. Experiments described in Chapters 5 and 6 use CProgol4.4, and those in Chapter 7 use Aleph. The rest of this section describes CProgol. An overview of Aleph is given in Section 2.4.2.

Representing Input and Defining Hypotheses Space

Like other ILP systems, CProgol learns from examples and background knowledge. The positive examples are represented as definite clauses, but usually

[1]Versions of CProgol are available from `http://www.doc.ic.ac.uk/~shm/Software/`

[2]`http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/PProgol/ppman.html`

as ground unit clauses. And the negative examples are represented as headless clauses. Unlike background knowledge in FOIL and GOLEM which can only consist of ground facts (Lavrač and Džeroski, 1994), background knowledge in CProgol may consist of background facts, which describe the examples (also called extensional definitions), and/or general rules, which are common to examples within the domain (also called intensional definitions).

To let CProgol know where to search for the hypotheses, the hypotheses space has to be defined. This is done by declaring types and modes. *Types* specify the categories of objects under consideration, and usually are declared by listing unary predicates of the form

```
type(object).
```

where `type` is a type name and `object` is an object name. Types are enforced in CProgol. As a result, only part of the hypotheses space is considered by CProgol, that is the part which is type-conform.

Modes specify the relation between objects of given types and the forms that atoms can take place in a hypothesis clause. There are two different kinds of modes declarations, i.e., *modeh* (to be used in the head of a hypothesis) and *modeb* (to be used in the body of a hypothesis). The two modes are usually declared in the form of

```
:- modeh(RecallNumber,HeadTemplate)?
:- modeb(RecallNumber,BodyTemplate)?
```

where `RecallNumber` is either a number > 0 which specifies how many times the `HeadTemplate` or `BodyTemplate` can be called successfully, or a '$*$' which specifies that the `HeadTemplate` or `BodyTemplate` can be called successfully up to 100 times; `HeadTemplate` and `BodyTemplate` are n-ary predicates, with $n \geq 1$ and each of the arguments is a *variable type* preceded by either a '+' (indicates that the argument should be an input), '-' (indicates that the argument should be an output), or '#' (indicates that the argument should be

a constant). A *variable type* can be a type defined normally by a list of unary predicates or a Prolog built-in function.

The hypotheses search in CProgol can be controlled through a set of parameters, integrity constraints, and prune statements. By reducing the amount of search, the amount of resources (time and memory) spent during searching will also be reduced. The default parameter settings of CProgol can sometime be enough to solve simple problems. However, complex problems may need different settings. How these parameters have to be set depends on the problem's needs. Some of these parameters, which are used for experiments in this research, are shown in Table 2.1. The next subsection (Hypotheses Construction) may help to understand this table better.

Table 2.1: CProgol parameter settings. The default values shown are those for CProgol4.4.

Name	Default	Meaning
posonly	off	If on, then CProgol turns on the positive-only learning mechanism.
inflate	400	This gives a weighing to the examples.
i	3	The maximum depth of variables which may occur in the most specific clause.
c	4	The maximum number of atoms in the body of the rules constructed.
nodes	200	The maximum number of nodes explored during clause searching.
h	30	The maximum depth of the stack used when proving before starting backtracking.
r	400	The maximum depth of resolutions allowed when proving. If exceeded, the proof is failed.

The parameters which may only take the values 'On' and 'Off', such as **posonly**, are set to 'On' using `set/1` command

```
:- set(Parameter)?
```

and set to 'Off' using `unset/1` command

```
:- unset(Parameter)?
```

while those parameters which values are of type integers, are set using `set/2` command as follows.

```
:- set(Parameter,Value)?
```

Integrity constraints are represented as headless clauses, but are stored as clauses with head *false*. They are used to check the consistency of every hypothesis, and thus work only after the hypothesis is constructed. On the contrary, *prune statements* can disallow certain clauses to be considered during hypothesis construction, and thus can be more efficient. Prune statements often take the form

```
prune(Head,Body) :- Conditions.
```

where `Conditions` is a set of literals describing the condition(s) that have to be fulfilled by the clauses being considered to be pruned (i.e., the clauses with head `Head` and body `Body`).

Hypotheses Construction

Given examples, background knowledge, a hypotheses space and parameter settings (all incorporated into one '.pl' file), CProgol tries to construct a general rule which can explain the given examples. It first checks if the input was consistent, and then constructs the *most specific* (also called *bottom*) clause, denoted by '$\perp$', of the first input positive example. The bottom clause is usually

a definite clause with many atoms in its body. The process of generating the most specific clause before the search starts is called *saturation*.

CProgol then uses an A*-like (best-first) search strategy to search for hypotheses H, starting from the *empty* clause, denoted by '$\square$', using general-to-specific approach through the *sub-lattice* such that the empty clause is *at least as general as* H and H is at least as general as the most specific clause ($\square \preceq H \preceq \bot$) (see Figure 2.1 for an illustration).

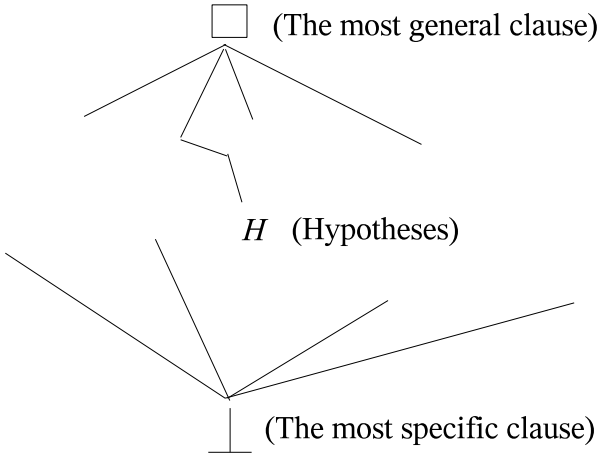

Figure 2.1: A lattice. The most general clause and the most specific clause are used as bounds for the hypotheses search.

General-to-specific hypotheses construction is done through a refinement operator which adds one literal at a time. This process can be considered as searching a tree structure, with the empty clause $\square$ as its root node and the children of a node are the possible refinements at that node. At each node, a *compression* value is calculated to measure how well the clause (the path from root to that particular node) explains the examples. The compression function is

$$f_c = p_c - n_c - l_n$$

where p_c and n_c are the number of positive and negative examples deducible

from clause c, and l_n is the number of atoms in the body of clause c.

The clause (the path) which has the maximum compression value will be added to CProgol's knowledge base. The examples which are explained by that clause are retracted from CProgol's knowledge base. If there were still more positive examples unexplained, CProgol repeats the whole process again, starting from the first of the remaining unexplained positive examples. This whole process uses a *covering* approach as the aim is to cover/explain all the positive examples.

2.4.2 Aleph

This section briefly introduces Aleph, another ILP system used in this research (see Chapter 7). More details on Aleph can be consulted from the Aleph online manual (Srinivasan, 1999).

Aleph stands for A Learning Engine for Proposing Hypotheses. Although Aleph has the same root with CProgol, it has evolved differently. Its development is influenced by many researchers, but Ashwin Srinivasan, the main author, is still the person in-charge of its maintenance. Unlike CProgol, Aleph is written in Prolog and is best use with Yap[3] Prolog compiler. The current Aleph is Aleph Version 5.

Representing Input and Defining Hypotheses Space

Generally Aleph needs positive examples, negative examples and background knowledge to construct hypotheses. They are kept in three different files with the same filestem but with different extensions (i.e., '.f', '.n', and '.b', respectively). The positives and negatives are represented in the same way, usually as ground unit clauses. Besides domain knowledge, the '.b' file also contains description of the hypotheses space as well as the search restrictions for Aleph (including integrity constraints and prune statements, if any).

[3] `http://www.ncc.up.pt/~vsc/Yap/`

As with CProgol, types and modes are also used in Aleph. Types are declared in the same way as in CProgol. *modeh* and *modeb* are not necessarily declared differently as in CProgol; both modes can take the form

```
:- mode(RecallNumber,Predicate).
```

where `RecallNumber` is either a number > 0 specifying how many times the `Predicate` can be called successfully, or a '$*$' which specifies that the `Predicate` can be called successfully many times; and `Predicate` is an n-ary predicate, with $n \geq 1$, and each of the arguments of the `Predicate` is either a *simple* or a *structured* (Srinivasan, 1999). A *simple* is a *variable type* preceded by either a '+' (indicates that the argument should be an input), '-' (indicates that the argument should be an output), or '#' (indicates that the argument should be a constant). While a *structured* is an m-ary functor, with $m \geq 1$ and each of the arguments is either a *simple* or a *structured*. A *variable type* can be a type defined in the background knowledge or a Prolog built-in function.

Modes are less important in Aleph than in CProgol. This is because of *determination* statements. Determination statements are declared in the form of

```
determination(TPredicate/TArity, BPredicate/BArity).
```

where `TPredicate/TArity` is the name and arity of the target predicate (the predicate that is being learned) and `BPredicate/BArity` is the name and arity of a predicate that can be used in the body of `TPredicate`. Since each of the predicates that can be used to construct a hypothesis should be declared in determination statements, there are usually many determination statements declared for one target predicate (note that Aleph can only learn one target predicate at a time). The existence of determination reduces the learning complexity. The determination statements together with background knowledge can be used to automatically extract modes information when they are not given (by using `induce_modes` feature).

Just like CProgol, Aleph uses parameter settings, integrity constraints and prune statements to control the hypotheses search. Several of the Aleph's parameters are shown in Table 2.2. The next subsection (Hypotheses Construction) may help to understand this table better.

Table 2.2: Aleph parameter settings.

Name	Default	Meaning
i	2	The maximum depth of new variables.
clauselength	4	The maximum number of atoms in an acceptable clause (hypothesis).
nodes	5000	The maximum number of nodes explored during clause searching.
depth	10	The maximum depth of the stack used when proving before starting backtracking.
samplesize	0	When set > 0, Aleph uses random selection for saturation.
construct_bottom	*saturation*	When set to *reduction*, the bottom clause is constructed during the search; when set to *false*, no bottom clause is constructed.
evalfn	*coverage*	When set to *posonly*, Aleph uses positive-only learning.

The parameters are set using `set/2` command, the same as in CProgol (note that CProgol's commands end with '?', but Aleph's commands end with '.'), and un-set using `noset/1` command

```
:- noset(Parameter).
```

Integrity constraints in Aleph are represented as rules of the form:

```
false :- Body.
```

where `Body` is a set of literals that describe the condition(s) that should not be violated by the clause being considered. Besides internal pruning, Aleph also accepts user-defined prune statements, which take the form

```
prune((Head:-Body)) :- Conditions.
```

where `Conditions` is a set of literals describing the condition(s) that have to be fulfilled by the clauses being considered to be pruned (i.e., the clauses with head `Head` and body `Body`).

Hypotheses Construction

When compared to CProgol, Aleph gives more choices to the way hypotheses are constructed. In this section, some of these choices as well as the basic one are discussed briefly. The basic algorithm (issued by `induce` command) that Aleph uses for hypotheses construction is similar to the one employed by CProgol. Rather than just choosing the first positive example to be saturated, in Aleph there is an option to randomly selecting n examples to be saturated; this is done by setting **samplesize** (see Table 2.2). Unlike in CProgol, where the bottom clause is always constructed before the search starts, Aleph allows the bottom clause to be constructed *lazily* (during the search) or not to be constructed at all (by setting **construct_bottom**).

Like CProgol, Aleph's basic algorithm removes the examples covered by the clause added to the knowledge base. This removal reduces the number of examples in the knowledge base and thus affects clause scoring (compression measure in CProgol). This problem can be alleviated by using `induce_cover`, which keeps the examples but prevents them from being chosen for the next saturation. Aleph's default clause scoring function is

$$coverage = p_c - n_c$$

where p_c and n_c are the number of positive and negative examples covered by clause c. Different scoring function can be used by setting `evalfn`.

Both CProgol's and Aleph's basic algorithms select the first positive example to be saturated and therefore the rules constructed may depend on the ordering of the input positive examples. Aleph provides a way to remove this dependency through `induce_max`. However, it comes with a trade off in computation time. Furthermore, the resulting rules may overlap with each other. Aleph's feature `induce_max` is explored further in Chapter 7.

Both, CProgol and Aleph, offer the positive-only learning.

2.5 Evaluating and Measuring Performances of ILP Systems

In principle, there are three phases involved in applying ILP for scientific knowledge discovery. First is the *training phase*, where the ILP system generates hypotheses (a set of rules) from a set of training examples with known classes and background knowledge. Second is the *testing phase*, where the set of rules generated during training phase is evaluated on a test set with known classes. The last phase is when the set of rules generated during training phase is applied to unseen examples, whose classes are not known, in order to classify novel examples. Examples with known classes are sometime called labelled examples while those whose classes are not known are sometime called unlabelled examples.

2.5.1 Evaluation using an Independent Test Set

The hypothesis generated in the training phase needs to be evaluated in the testing phase. The overall aim of using ILP is to generate the best hypothesis which can classify future unseen examples. Therefore, the most common evaluation method used is to evaluate the hypothesis against an *independent test*

set (a set of labelled examples which is not used in the training phase). Measuring the performance on the data used for training would give an optimistic estimate of the predictive ability of the hypothesis (Witten and Frank, 2000, Chapter 5). When using this evaluation method, the full data set with known classes is usually divided into two parts, e.g. $\frac{2}{3}$ is used for training and the other $\frac{1}{3}$ for testing. Hence, this method is also known as *holdout*. Holdout is especially best when data are large.

A shortcoming of holdout is that the training set and/or test set may not be representative. To overcome this, *stratification* can be applied. Here, both the training and test set are randomly sampled in such a way that all classes are properly represented. To get a better estimation, holdout can be repeated several times with each iteration using different training set and test set. An estimate of the predictive performance is calculated by taking the average of the performances from each iteration.

2.5.2 Evaluation using Cross-validation

Repeated holdout becomes the basis for an evaluation method called *cross-validation*. It is a method where data is divided into a fixed (N) equal partitions (called *folds*), and each of these folds are in turn used as test set while the remainder is used as training set. When $N = 10$, the method is called tenfold cross-validation, and if stratification is applied, it is then called stratified tenfold cross-validation. Although $N = 10$ is standard, $N = 3$ is also common. To get a more reliable performance estimation, the tenfold cross-validation is usually repeated ten times. The final estimation from the ten tenfold cross-validation is the average of each tenfold cross-validation.

When data are scarce, *leave-one-out* cross-validation can be used for evaluation. In this case, each instance in the data set is in turn used as a test set, while all the remaining instances are used as a training set. An estimate of the performance is the average of the performances from each iteration.

2.5.3 Performance Measures

To assess the performance of a classifier, performance measures are used. For binary classification problems, the classifications are grouped as true positive, true negative, false positive and false negative. Table 2.3 shows the meaning of each group; this table is usually called a *confusion matrix* (also known as a *contingency table*).

Table 2.3: Group of classification for binary problems. Each of the cells contains the number of examples fall into each group.

		Predicted Class	
		Positive	Negative
Actual Class	Positive	True positive (TP)	False negative (FN)
	Negative	False positive (FP)	True negative (TN)

Predictive Accuracy

The most widely used performance measure in machine learning is *predictive accuracy*, which describes the proportion of the test set which are correctly classified.

$$\text{Predictive accuracy} = 100 \times \frac{(\text{TP}+\text{TN})}{(\text{TP}+\text{TN}+\text{FP}+\text{FN})}$$

It is usually expressed as a percentage. When using predictive accuracy, the cost of the two types of misclassification are assumed to be equal.

Although predictive accuracy has been widely used and often works well, it is not suitable for domains in which imbalanced data (a big difference in proportion between the number of positives and negatives) is observed. Predictive accuracy gives a poor estimate when used in domains where the positives are

rare but the negatives are abundant or the other way around while the weight of both positives and negatives are equal.

Lift Factor

One of performance measures which works when positives are rare and negatives are abundant is *lift factor*. It is mainly used in direct marketing as a measure of increase in response rate using a model over random sampling (Ling and Li, 1998).

To be able to work out a lift factor, a classifier has to give some confidence measure (probability) along with the classifications made. The confidence values are used to sort, in decreasing order, the examples in the test set. The lift factor is defined as the ratio of success proportion on a chosen sample set divided by success proportion on the whole test set.

$$\text{Lift factor} = \frac{\text{TP}_{Sample}/\text{N}_{Sample}}{\text{TP}_{Testset}/\text{N}_{Testset}}$$

Precision, Recall, F_1 Score, Sensitivity and Specificity

Besides those performance measures that have been discussed in this section, there are also precision and recall which are popular in information retrieval. *Precision* measures the fraction of predicted positives which are true positives, and *recall* measures the fraction of positives which are predicted as positives. Both measures are used together in F_1 score.

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \qquad \text{Recall or Sensitivity} = \frac{\text{TP}}{\text{TP+FN}}$$

$$\text{F}_1 \text{ score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Related with recall are sensitivity and specificity. *Sensitivity* is identical to recall. While *specificity* is recall for negative cases, i.e., it measures the fraction of negatives which are predicted as negatives.

$$\text{Specificity} = \frac{\text{TN}}{\text{FP+TN}}$$

A plot of sensitivity (on the Y-axis) against $1 -$ specificity (on the X-axis) is called *ROC diagram*. This diagram is often related with cost/benefit analysis. Deeper review of standard performance measures can be read from Provost and Fawcett (1997), Lavrač, Flach and Zupan (1999), Provost and Fawcett (2001) and Flach and Lavrač (2003).

Relative Advantage

Muggleton, Bryant, Srinivasan, Whittaker, Topp and Rawlings (2001) defined a performance measure called relative advantage (RA), which they claimed can be used to measure the performance of a recognition model (a set of rules) for any domain where the proportion of positives in the example set is very small, while the proportion of positive examples in the population is not known, acquiring negatives is difficult and a benchmark recognition method does not exist.

In Chapters 5 and 6, RA is used as a performance measure. This is because the uORF domain has the characteristics for which RA is claimed to be useful. These include the fact that the proportion of positives (functional uORFs) in the example set is very small, while the proportion of positive examples in the population (the whole *S. cerevisiae* genome) is not known, acquiring negatives is difficult (since these have to be verified by laboratory experiments) and a benchmark recognition method does not exist.

The idea of using RA is to predict cost reduction in finding functional

uORFs using a recognition model compared to using random sampling. Here, we adopted the RA definition from Muggleton et al. (2001, Appendix A).

$$RA = \frac{A}{B} \tag{2.1}$$

where

- A = the expected cost of finding one functional uORF by repeated independent random sampling from the set of possible uORFs and performing a lab analysis of each uORF;
- B = the expected cost of finding one functional uORF by repeated independent random sampling from the set of possible uORFs and analysing only those uORFs which are predicted by the learned model as functional uORFs.

Up to this point, RA may sound similar to lift, but they are different. RA is suitable for domains where positives are rare and negatives are difficult or expensive to get, while lift works well for domains where positives are rare and negatives are abundant. Although the lift factor gives a measure of increase in response rate using a model over random sampling, it does not give information (prediction) about the size of the reduction in cost or the gain in profit in using the model compared to using random sampling. Furthermore, to be able to work out the lift factor, a classifier has to give some confidence measure along with the classifications made. Although getting a confidence measure for each classification is not impossible in ILP, it is not usually provided by classical ILP systems. A branch of ILP, known as probabilistic ILP, emerged to answer probabilistic challenges within ILP systems (see e.g. Raedt and Kersting (2004)).

The rest of this section describes how RA is defined in terms of probability and how RA is estimated. This description is adapted from Muggleton et al. (2001, Appendix A), to suit the uORF problem addressed in this thesis. Within

Table 2.4: 2×2 Contingency table for the test set[a].

	Functional uORFs	Random examples
H	n_1	n_2
$\overline{H}$	n_3	n_4

[a] H (hypothesis predictions) and $\overline{H}$ (complement of H).

$n_1 + n_2 + n_3 + n_4 = n$, where n is the number of instances in the test set.

this description, a set of *random examples* refers to a set of unlabelled examples which would mainly contain negatives and maybe a small number of positives.

In terms of probability, RA can be defined as follows.

$$RA = \frac{C/Pr(FuORF)}{C/Pr(FuORF|Rec)} = \frac{Pr(FuORF|Rec)}{Pr(FuORF)} \tag{2.2}$$

where

- C = the cost of verifying whether a uORF is functional or not via laboratory experiments;
- $FuORF$ = uORF is functional;
- Rec = the model recognises (predicts) the uORF as a functional uORF.

Suppose Table 2.4 shows the result of testing the model on a test set of size n and therefore $n_1 + n_2 + n_3 + n_4 = n$. The set of random examples referred to in the right-hand column may include some positives (i.e., functional uORFs). If the proportion of functional uORFs in the test set was known to be the same as the proportion of functional uORFs in the set of all possible uORFs in *S. cerevisiae* genome, then one could estimate $Pr(FuORF)$ to be $(n_1 + n_3)/n$ and $Pr(FuORF|Rec)$ to be $n_1/(n_1 + n_2)$. However, since the proportion of functional uORFs in the test set cannot be assumed to be the same as the proportion of functional uORFs in the set of possible uORFs, these estimates cannot be used. Therefore, given both a set of positives and a set of randoms,

Table 2.5: 2×2 Contingency table for the set of all possible uORFs in the yeast *S. cerevisiae* genome[a].

	Functional uORFs	Negative examples
H	$\left(\frac{n_1}{n_1+n_3}\right) M$	$\left(\frac{n_2}{n_2+n_4}\right) (S-M)$
$\overline{H}$	$\left(\frac{n_3}{n_1+n_3}\right) M$	$\left(\frac{n_4}{n_2+n_4}\right) (S-M)$

[a] H (Hypothesis predictions) and $\overline{H}$ (complement of H). The total of the counts or frequencies in the four cells $= S$, where S is the total number of all possible uORFs in the yeast *S. cerevisiae* genome.

$Pr(FuORF)$ and $Pr(FuORF|Rec)$ are estimated as follows. Let S be the total number of possible uORFs in *S. cerevisiae* genome, of which M are functional uORFs.

$$\begin{aligned} Pr(FuORF) &= \frac{no.\ of\ functional\ uORFs\ in\ the\ set\ of\ possible\ uORFs}{no.\ of\ possible\ uORFs} \\ &= M/S \end{aligned} \tag{2.3}$$

$$Pr(FuORF|Rec) = \frac{N_{FuORF\_recognised\_by\_model}}{N_{uORFs\_predicted\_as\_functional}} \tag{2.4}$$

where $N_{FuORF\_recognised\_by\_model}$ is the number of functional uORFs in the set of possible uORFs which are recognised by model and $N_{uORFs\_predicted\_as\_functional}$ is the number of possible uORFs which the model predicts to be functional.

Now, the expected result of using the learned recognition model on the set of all possible uORFs in the yeast *S. cerevisiae* genome is shown in Table 2.5. From Equation 2.4 and Table 2.5 it follows that:

$$\begin{aligned} Pr(FuORF|Rec) &\simeq \frac{\left(\frac{n_1}{n_1+n_3}\right) \times M}{\left(\frac{n_1}{n_1+n_3}\right) M + \left(\frac{n_2}{n_2+n_4}\right) (S-M)} \\ &= (Mp_1)/(Mp_1 + (S-M)p_2) \end{aligned} \tag{2.5}$$

where $p_1 = n_1/(n_1+n_3)$ and $p_2 = n_2/(n_2+n_4)$. Substituting Equations 2.3 and 2.5 into Equation 2.2 gives

$$RA = \frac{(Mp_1)/(Mp_1 + (S-M)p_2)}{M/S}$$

$$
\begin{aligned}
&= \frac{Sp_1}{Mp_1 + (S-M)p_2} \\
&= \frac{Sp_1}{Sp_2 + M(p_1 - p_2)}
\end{aligned} \tag{2.6}
$$

Since the value of M (the total number of functional uORFs in the population) is not known, the relative advantage over the entire population ($\sum_{M_{min}}^{M_{max}} RA$) is estimated by integrating Equation 2.6 with respect to M. The lower limit of M (M_{min}) is equal to the number of known functional uORFs in *S. cerevisiae* genome. The upper limit of M (M_{max}) is the most probable number of functional uORFs in the yeast *S. cerevisiae* genome i.e., a total of the known functional uORFs and those uORFs which have yet to be scientifically recognised as functional uORFs.

$$
\begin{aligned}
\sum_{M=M_{min}}^{M_{Max}} RA &\simeq Sp_1 \times \int_{M=M_{min}}^{M_{max}+1} \frac{1}{(p_1 - p_2)M + Sp_2} \partial M \\
&= \left[\frac{Sp_1}{(p_1 - p_2)} \ln((p_1 - p_2)M + Sp_2) + k \right]_{M_{min}}^{M_{max}+1} \\
&= \frac{Sp_1}{(p_1 - p_2)} \ln \frac{(M_{max}+1)(p_1 - p_2) + Sp_2}{M_{min}(p_1 - p_2) + Sp_2}
\end{aligned} \tag{2.7}
$$

$\sum_{M=M_{min}}^{M_{max}} RA$ is estimated by summing an estimate of the $\sum_{M=M_{min}}^{M_{max}} RA$ for each instance in the test set as follows, where n is the number of instances in the test set and ra in lower case represents the relative advantage over a sample.

$$
\sum_{k=1}^{n} \sum_{M=M_{min}}^{M_{max}} ra_k \tag{2.8}
$$

From Equation 2.8 and the contingency table it follows that:

$$
\sum_{M=M_{min}}^{M_{max}} ra = \frac{1}{n} \sum_{i=1}^{4} \left(n_i \sum_{M=M_{min}}^{M_{max}} ra_i \right) \tag{2.9}
$$

Each $\sum_{M=M_{min}}^{M_{max}} ra_i$ is estimated by substituting $p_1 = \frac{a}{a+c}$ and $p_2 = \frac{b}{b+d}$ into Equation 2.7. The values of a, b, c and d are determined by three steps.

1. a, b, c and d are initially given the values of the corresponding counts in the contingency table for the test set (see Table 2.4).

2. Each one of a, b, c and d, is decremented providing that the value before subtraction is greater than 1.

3. The value of either a, b, c or d is incremented to reflect the classification of an instance in the cell n_i.

For instance, if $i = 2$ and all the counts in the contingency table are greater than one then $a = n_1 - 1, b = n_2, c = n_3 - 1, d = n_4 - 1$.

Finally, RA is estimated by taking the mean of the summed values.

$$Mean\ RA = \frac{\sum_{M=M_{min}}^{M_{max}} ra_i}{M_{max} - (M_{min} - 1)} \tag{2.10}$$

2.6 Summary

This chapter gave an overview of ILP and ILP Systems, CProgol and Aleph, and showed why ILP was selected to be used in this research. The next chapter will briefly present the field of bioinformatics, which is the application domain of this research, as well as a review of some ILP/ML work on bioinformatics.

Chapter 3

An Overview of ILP/ML Work on Bioinformatics

The previous chapter presented the framework of ILP, the ML technique used in this research. In this chapter, we give an overview of some types of problems in bioinformatics and review some of ML (particularly ILP) work on bioinformatics.

As mentioned in Chapter 1, the central information in bioinformatics is deoxyribonucleic acid (DNA) sequences. It is from the DNA sequences that the protein sequences are determined (see Figure 4.3 on page 51). The protein sequence dictates the protein structure and the structure of a protein determines its function in life processes. Thus from its early development, bioinformatics has dealt primarily with *biological sequence analysis*, such as in sequence similarity and sequence alignment, and *structural biology*, such as for determining protein structure and understanding the relationship between structure and function.

Here, for the purpose of clarity, we will first outline protein related problems, then problems in functional genomics, which involves gene (the segment of DNA which contains the information for protein synthesis), ribonucleic acid (RNA) and protein, and finally messenger RNA (mRNA) related problems.

Table 3.1: The 20 standard amino acids.

Amino acid	Symbol		Amino acid	Symbol	
	3-letter	1-letter		3-letter	1-letter
Alanine	Ala	A	Leucine	Leu	L
Arginine	Arg	R	Lysine	Lys	K
Asparagine	Asn	N	Methionine	Met	M
Aspartic acid	Asp	D	Phenylalanine	Phe	F
Cysteine	Cys	C	Proline	Pro	P
Glutamine	Gln	Q	Serine	Ser	S
Glutamic acid	Glu	E	Threonine	Thr	T
Glycine	Gly	G	Tryptophan	Trp	W
Histidine	His	H	Tyrosine	Tyr	Y
Isoleucine	Ile	I	Valine	Val	V

3.1 Protein Structure

A *protein* is a molecule composed of *amino acids*. The amino acids are linked to each other by *peptide bonds*. When many amino acids are linked together, they formed a *polypeptide*. A protein is usually made of one or more polypeptide chains. There are twenty standard amino acids (Table 3.1). Each of these amino acids is identified by a different *side chain* (the R-group in Figure 3.1). The unit in the grey box of Figure 3.1 is the same for all amino acids. Therefore, a polypeptide chain contains a repeat of these units, which form the *main chain* (also called the *backbone*), and variable side chains.

The structures of proteins are classified into four levels. The amino acid sequence of a protein, such as

```
his ser gln gly thr phe thr ...
```

is the *primary structure* of the protein. Due to the flexibility of the chain,

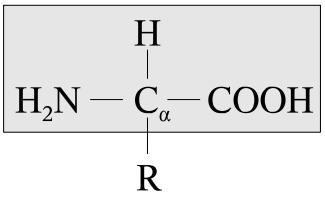

Figure 3.1: The general structure of an amino acid.

different subregions of a polypeptide can form different local substructures. These substructures are called *secondary structures*. Two types of secondary structures are very common to many proteins; they are *α-helix* and *β-sheet*. Interactions between the helices and sheets in one polypeptide create different *folding patterns*. An overall three-dimensional shape of one polypeptide is the *tertiary structure* of a protein. If a protein has two or more polypeptide chains then the interactions between these polypeptide chains create a three-dimensional shape called *quarternary structure*.

The function of a protein can be determined by its three-dimensional shape. However, predicting the three-dimensional shapes of proteins from their amino acid sequence has proved to be one of the hardest challenges and remains unsolved.

3.1.1 Experimental Approaches

For a long period, X-ray crystallography was the main experimental technique used for determining three-dimensional protein structure. This technique suffers from a long preparation time for purification and crystallisation of the protein, especially for complex protein molecules.

In the 1980s, a technique called nuclear magnetic resonance (NMR) spectroscopy started to be used for protein structure determination. Unlike X-ray crystallography which requires protein crystals, protein samples for NMR can be in a solution or in a solid state. Although NMR usually takes a very short

preparation time, it can take a long time to interpret NMR spectra. Another limitation of NMR is that it can only be efficiently used for proteins with less than 350 amino acids (Pevsner, 2003, Chapter 9).

Another technique that is used in protein structure determination is the low-temperature electron microscopy (cryoEM). Its main advantage over X-ray crystallography and NMR is the ability to process large protein complexes. Nevertheless, throughput from cryoEM is still low.

While experimental approaches are still the best way for determining protein structure (Lesk, 2002), these approaches tend to be time-consuming. Therefore, it is difficult to get the structure of every protein of interest by relying only on these approaches. This situation has boosted the development of pure computational methods for protein structure prediction. A recent review on protein structure determination is given by Liu and Hsu (2005).

3.1.2 Contemporary Bioinformatics Approaches

In principle, predicting protein structures computationally can be done in two different approaches (Lesk, 2004). The first approach is to learn from the natural protein folding process, and then simulate the folding process in a computer. This way is usually implemented as finding the global minimum of the conformational energy function. Nonetheless, it is not a successful approach due to the difficulty in accurately calculating the energy of a large number of different conformations.

The second approach is to ignore the natural protein folding process and to consider the sequence information. Included in this approach are homology modelling and fold recognition. In *homology modelling*, the sequence of a target protein is aligned with one or more related (*homologous*) proteins, whose three-dimensional structures are known. The homologous protein which gives the optimal alignment is then used as a template. The three-dimensional structure of the target protein is predicted based on the alignment and the template

structure. Therefore, the quality of the prediction using homology modelling depends on the sequence similarity between the target protein and the template protein. When the sequence similarity is low or when a related protein cannot be identified, *fold recognition* is more appropriate. The latter tries to fit a target sequence to the backbone structures of a library of known protein structures (or folds); the backbone structures were used as anonymous placeholders (Jones, Taylor and Thornton, 1992). The fold which gives the best fit will be chosen as the predicted three-dimensional structure of the target protein. More detail review about homology modelling and fold recognition can be read from Forster (2002).

Beside these bioinformatics approaches, there are also other computational approaches in trying to predict protein structures such as those described in the following subsection.

3.1.3 Machine Learning Work on Protein Structure

Protein Secondary Structure Prediction

As it is very difficult to determine the three-dimensional shape of a protein directly from its primary sequence, scientists have tackled this problem in two stages: first, by predicting the secondary structures (α-helices and β-sheets) and second, assembling them.

Machine learning has been applied to protein secondary structure prediction. An example of such work is Muggleton, King and Stenberg's (1992) work. They used an ILP system, GOLEM (Muggleton and Feng, 1990), to learn a set of rules for predicting secondary structure of α/α domain type proteins and were able to achieve better performance when compared with previous work on identical data, such as King and Sternberg (1990).

Protein Fold Signature

Turcotte, Muggleton and Sternberg (2001) compared attribute-valued and relational background knowledge on learning protein three-dimensional *fold signatures* (structural features of a fold), using CProgol4.4. Although they were able to show that the augmentation of relational background knowledge increased the learner's performance and improved the explanatory insight, they realised the need to incorporate structural superpositions (multiple structure alignment) in the background knowledge. This was realised in Cootes, Muggleton and Sternberg (2003) and has increased the effectiveness in capturing the global fold features.

3.2 Functional Genomics

The ultimate goal of *functional genomics* is to identify the function(s) of all genes. Experimental work in this area has benefited from high-throughput technologies, as will be shown briefly in Section 3.2.1. Such technologies produce a lot of data, which in turn increases the need for computational approaches to process and analyse such data effectively. In Section 3.2.2, we review some of the functional genomics research where machine learning has been used.

3.2.1 Experimental Approaches

Experimentally, determination of gene function can be done in several ways. The most direct way is by knocking out (deleting) a gene in question and thus creating a deletion mutant and then observing the phenotype differences between this mutant and its wild (normal) type (Delneri, 2004; Oliver, Winson, Kell and Baganz, 1998).

A more effective way of determining novel gene function at a genome-wide scale is the quantitative analysis of mRNA level under various conditions. This can be done through hybridisation-array technologies such as

complementary DNA and oligonucleotide microarrays (Duggan, Bittner, Chen, Meltzer and Trent, 1999) or serial analysis of gene expression (SAGE) technology (Velculescu, Zhang, Vogelstein and Kinzler, 1995).

Elucidation of gene function can also be reached by measuring the level of protein present in a given set of conditions. This usually involves using two-dimensional gel electrophoresis and/or mass spectrometry (Ong and Mann, 2005).

3.2.2 Machine Learning Work on Functional Genomics

Just as experimental approaches are the best for protein structure determination, wet experimentation is the only way to determine the real function(s) of a gene. However, this type of experiment is very costly, time consuming and highly dependent on human scientists. Consequently, scientists have been trying to develop computational methods for functional genomics.

Sequence-Function Relationships

Functional genomics has become an interesting application domain to work on for many machine learning researchers over the years. For example, a group of researchers at Aberystwyth, UK, have been working on trying to predict protein function from sequence (King, 2004). Realising that it is highly difficult to get a general model of sequence-function relationships, their aim was set to get good rules which can explain some of the relationships.

Throughout their works (Clare and King, 2003; King, Karwath, Clare and Dehaspe, 2001; King, Karwath, Clare and Dehaspe, 2000a; King, Karwath, Clare and Dehaspe, 2000b), they used a combination of several data types and system softwares to achieve their aim. Among the systems they used are WARMR (Dehaspe and Toivonen, 1999)-an ILP-relational learner and C5.0[1]-a propositional decision tree learner. Their experiments using the propositional

[1] `http://www.rulequest.com/see5-unix.html`

learner with and without ILP showed that ILP increased the predictive accuracy. Although their rules cannot predict the specific function of the proteins, the rules can predict the functional class. Moreover, their rules can predict the functional class of proteins even without information of sequence homology.

In different ways, the approach of using different types of data and system softwares is also used in this thesis (Chapters 5 - 7).

Automated Phenotypic Analysis

While the Aberystwyth group was working on predicting function from sequence, Bryant and Muggleton (2000) simulated phenotypic analysis of a highly simplified cell model using ASE-Progol, a machine learning system which uses CProgol, to actively selecting experiments. Furthermore, Bryant, Muggleton, Oliver, Kell, Reiser and King (2001) applied an updated version of ASE-Progol to simulate the phenotypic analysis of the aromatic amino acid pathway of the yeast *Saccharomyces cerevisiae*; here, the system designs which experiments to do automatically. Both of these works aimed to create a system which can design and perform the experiments of phenotypic analysis automatically. This aim was realised by 'The Robot Scientist' (King, Whelan, Jones, Reiser, Bryant, Muggleton, Kell and Oliver, 2004; Whelan and King, 2004), the testing of which demonstrated that it is able to perform phenotypic analysis automatically to determine the function of genes participating in aromatic amino acid pathway of yeast.

Protein Interaction

Proteins carry out their roles in life processes by interacting with other proteins or other biological molecules. Hence, mapping protein interactions can increase understanding of protein function. Information about protein interactions can be obtained from data on protein-microarray (for protein-protein interaction) and/or DNA microarray (for protein-DNA interaction) (Sauer, Lange, Gobom,

Nyarsik, Seitz and Lehrach, 2005).

Because large volumes of data are generated from these microarray technologies, computational tools are needed to analyse the data. Furthermore, *in silico* prediction of protein interaction data can help to reduce the number of proteins to be screened *in-vitro*. However, there are not so many ML work on this problem yet. Recently, Tran, Satou and Ho (2005) reported their work using Aleph (Srinivasan, 1999) for predicting protein-protein interactions from several protein databases. Although the biological impact of their work is not clear, they have shown that the results are promising in terms of getting high sensitivity and specificity. Beside that, Thierry-Mieg and Trilling (2000) have tried to use CProgol to generate rules which could explain protein-protein interactions data in InterDB[2]. However, their trial was not successful. This was because CProgol was instructed to learn rules where the bodies are conjunction of many binary predicates from a large volume of data in InterDB, which of course creates a huge search space.

3.3 mRNA Related Problems

In Sections 3.1.3 and 3.2.2, we have reviewed some of ML work on protein structure and functional genomics. In this section, we will review some of ML work on mRNA related problems.

One of the current mRNA related open problems is the identification of *transcription start sites* (TSSs). An example of ML work in this area is the work by Gordon et al. (Gordon, Towsey, Hogan, Mathews and Timms, 2006; Towsey, Gordon and Hogan, 2006). By experimenting with different types of support vector machines (SVMs), they made their attempts to predict the transcription start sites of a bacterial genome (*Escherichia coli*). Beside trying to predict the exact location of the TSSs, they did some experiments to predict the segment

[2]InterDB is a protein interaction database for *Caenorhabditis elegans.*

where TSSs may lie. When compared to the segment approach, they found that attempting to predict the exact location gives a higher false positive rate.

Like Gordon et al. (2006), Bockhorst, Qiu, Glasner, Liu, Blattner and Craven (2003) also used ML for their study on predicting three types of regulatory elements (operons, promoters and terminators) in *E. coli* genome. The model used for making predictions was learned from two different sources of observations, namely DNA sequences and an analysis of array experiments. Generally, they found that using both sources together gives better predictions than using each of the sources independently.

Another example of ML work on an mRNA related problem is the work by Horváth, Wrobel and Bohnebeck (2001). The authors compared three different representations, i.e., relational representation with lists, relational representations with terms and propositional (flat) representation, on the problem of classifying regulatory elements[3] of mRNA. Four classes of regulatory elements, i.e., iron responsive element (IRE), trans activating region (TAR), selenocysteine insertion sequence (SECIS) and histone stemloops, were studied. Horváth et al. (2001) found that representing mRNA structure in the background knowledge became much easier using relational representations. They measured the accuracies and the running times of their experiments when background knowledge was represented using each of the three representations. Their results showed that the relational representations, especially with terms, have improved the accuracies, but at a cost in increased running times. In Chapter 6, we will describe how we make use of mRNA structural features for our study.

Although ML techniques have been applied to the mRNA related studies, such as those which were reviewed in this section, we are not aware of any previous ML work on learning particular regulatory elements called upstream open reading frames. The latter is the main theme of this thesis and will be discussed in Chapters 4 - 7.

[3]Horváth et al. (2001) called the regulatory elements as *signal structures*.

3.4 Summary

This chapter gave an overview on bioinformatics as a suitable application domain and a promising field for the theme of my research. From the work that are reviewed here, several lessons can be learned.

In general, ILP is an expressive technique for bioinformatics. The relational representation was found particularly useful for some bioinformatics problems, as shown by Horváth et al. (2001). If ILP is to show its power, it is important to carefully choose both a way of representing the problem and relevant background knowledge. Combining knowledge from different sources may be necessary and useful for some problems, as demonstrated by Bockhorst et al. (2003). As a machine learning technique, ILP can be suited to work at different stages of the knowledge discovery process. For example, ILP can be used for working with raw data (biological sequences) or even designing which experiments to do. However, like other ML techniques, ILP may need to be combined with other systems to solve a very complex problem, as in functional genomics.

Up to this point, the main computational technique which will be used to carry out our experiments (Chapter 2) and the general application domain where this thesis lies (this chapter) have been presented. The next chapter will present the biological foundation of this thesis.

Chapter 4

A New Bioinformatics Problem for ILP

In the previous chapter, we have shown the contribution of machine learning, particularly ILP, to solve or at least to take steps towards solving a range of bioinformatics problems. In this chapter, we introduce a new bioinformatics problem for ILP and machine learning in general. This problem is related with regulation of gene expression.

We begin by giving an overview of messenger RNA (mRNA) and its involvement in gene expression. We then introduce one type of regulatory element in the mRNA called the upstream open reading frame (uORF). We briefly illustrate how several uORFs regulate gene expression and describe the implication of this regulation on health-related cases. Finally, we identify a challenge facing biologists working in the uORF domain, and by doing so justify the need for the research described later in this thesis.

4.1 mRNA and Gene Expression

Messenger RNA (mRNA) is an RNA molecule which carries the instructions from DNA for protein synthesis (see Figure 4.3 on page 51). In this section,

the biological background of mRNA is explained in terms of its relation with RNA and DNA.

DNA (deoxyribonucleic acid), as illustrated in Figure 4.1, belongs to a group of organic compounds called *nucleic acids*, which are polymers of nucleotides. A *nucleotide* consists of a *pentose* (five-carbon) sugar, one or more phosphate groups, and a *nitrogenous base* (a compound that contain nitrogen). The pentose sugar for DNA is *deoxyribose*. Nitrogenous bases of DNA are adenine (A), guanine (G), cytosine (C) and thymine (T).

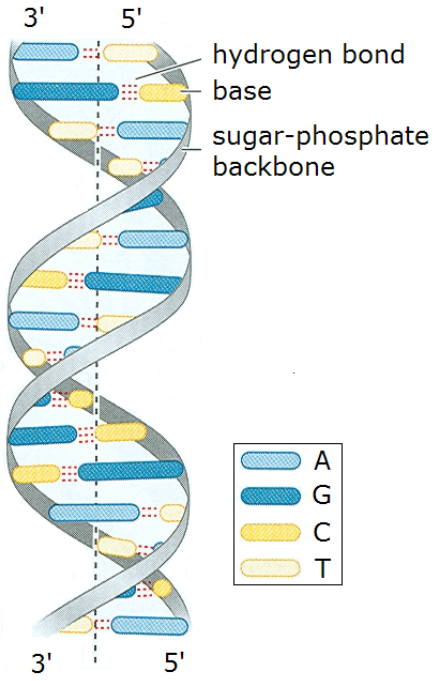

Figure 4.1: DNA structure. (Source: Adapted from Watson et al. (2004), Molecular Biology of the Gene, 5^{th} ed., fig 6-1a, p. 98. Copyright ©2004. Reprinted by permission of Pearson Education, Inc.).

Each DNA molecule consists of two polynucleotide chains (or strands) which run *antiparallel* (i.e., run in opposite direction: $5'$ to $3'$ and the other $3'$ to $5'$). The two strands form a double helix. The outside of the helix (also called the *backbone*) consists of alternating sugar and phosphate groups. The bases of

the two chains are paired along the inside of the helix. The pairings hold the two strands together. Adenine is always paired with thymine, and guanine with cytosine. Due to this arrangement, the two chains are said to be *complementary* to each other. If the sequence in one strand of DNA is

```
5'-ATGAAAGTAACACCCCAT...-3'
```

then the other strand (the complementary strand) will have the following sequence

```
3'-TACTTTCATTGTGGGGTA...-5'
```

An implication of this is that each strand of DNA can be used as a template to synthesise the other strand.

Like DNA, RNA (ribonucleic acid) is also a polymer of nucleic acids. However, RNA is different than DNA. Its sugar is *ribose* instead of deoxyribose. The base thymine is replaced with uracil (U) in RNA. This uracil is usually paired with adenine, but sometime can also be paired with guanine. Unlike DNA which is double stranded, RNA is usually single stranded. Being single stranded, RNA sometimes folds back on itself, creates base pairings between parts of the sequence which are complementary to each other and then forms various *stem-loop* structures (see Figure 4.2).

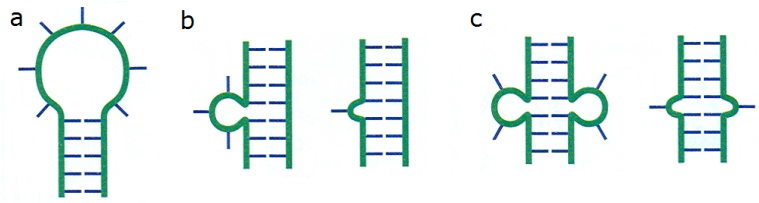

Figure 4.2: RNA secondary structure: a) hairpin, b) bulge, c) loop. (Source: Adapted from Watson et al. (2004), Molecular Biology of the Gene, 5^{th} ed., fig 6-30, p. 123. Copyright ©2004. Reprinted by permission of Pearson Education, Inc.).

Information from DNA for protein synthesis is copied into mRNA during *transcription* (see Figure 4.3). The mRNA is transcribed from one of the DNA

strands. In organisms whose cells have nuclei, namely *eukaryotes*, after the transcription is finished, the mRNA has to be transported from nucleus into cytoplasm before protein synthesis can take place (see Figure 4.4). The process of synthesising proteins by making use of the information in the mRNAs is known as *translation*.

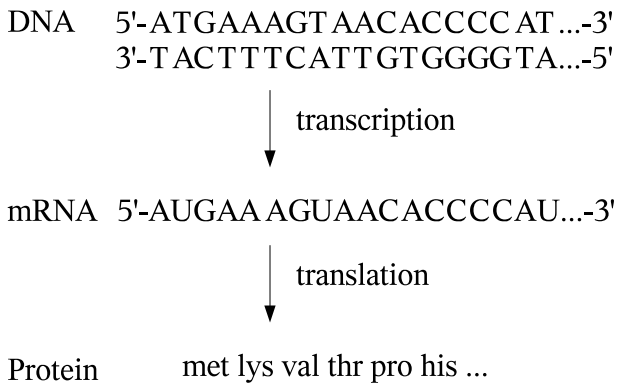

Figure 4.3: Outline of gene expression.

In order for an mRNA sequence to be translated into protein, every three consecutive bases in the mRNA are grouped together into a *codon*. Since there are four bases in mRNA, therefore there are 64 possible codons. 61 of these specify amino acids (see Table 3.1 on page 38 for a list of standard amino acids). Three codons that do not specify amino acids (i.e., UAA, UAG, and UGA) are used to give signals to terminate protein synthesis and thus are known as *stop codons*. Codon AUG, which codes for the amino acid methionine, is used to give a signal to start protein synthesis and thus is known as *start codon*. A series of non-overlapping codons creates a *reading frame*. A reading frame which starts with a start codon and ends with a stop codon creates an *open reading frame* (ORF). A subsequence of mRNA that codes for a protein (i.e., the coding sequence in Figure 4.5) is actually an ORF. A protein coding sequence is usually the longest ORF in an mRNA. Throughout this thesis, we may refer to this protein coding sequence by "main ORF", "main gene", or "coding sequence".

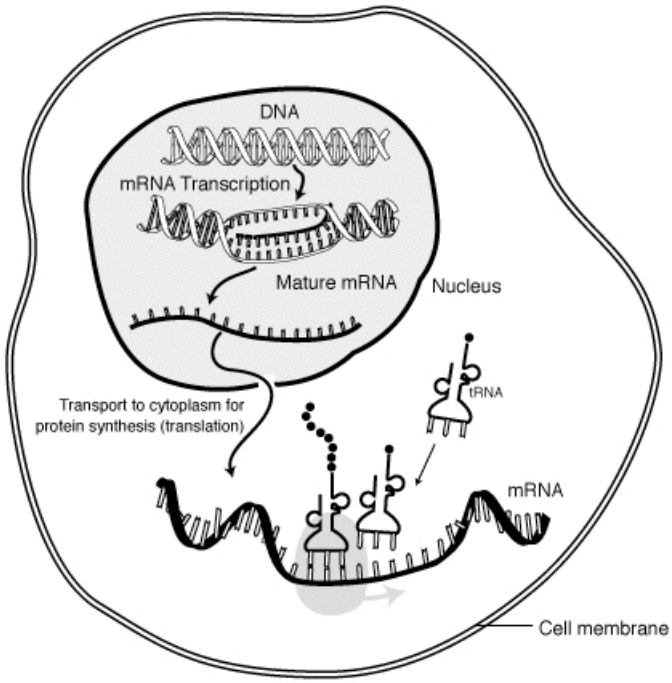

Figure 4.4: mRNA in a cell. (Source: freely available image from National Human Genome Research Institute www.genome.gov).

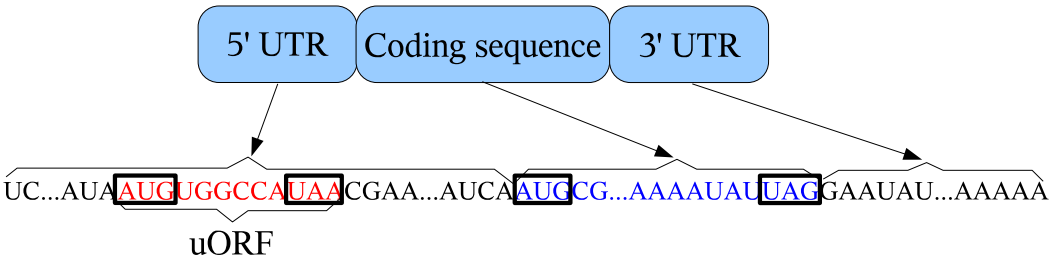

Figure 4.5: Simplification of mRNA primary structure. A, U, G, and C are RNA's bases. A codon is a triplet of bases. AUG is the start codon. A stop codon can be UAA, UAG, or UGA. A $5'$ UTR may have zero or more uORFs. Note: figure is not drawn to scale.

Beside containing the protein coding sequence, an mRNA also contains *untranslated regions* (UTRs) at its $5'$ and $3'$ ends (see Figure 4.5). These UTRs,

specifically the 5′ UTR, are known to play several key roles in regulating the expression of the coding sequence (Vilela et al., 1999; Mignone, Gissi, Liuni and Pesole, 2002; Ringnér and Krogh, 2005; Fiaschi et al., 1997; Wilkie, Dickson and Gray, 2003; Pickering and Willis, 2005). However, it is not yet clear through what mechanism the UTRs regulate the translation process. One of the reasons for this is that very little is known about regulatory elements in the UTRs.

There are several elements at the 5′ UTR that can regulate gene expression. These include 7-methyl-guanosine (m^7G) cap, stem-loop structure, upstream ORF (uORF), interacting protein, and internal ribosome entry site (IRES) (Mignone et al., 2002; Wilkie et al., 2003). In this thesis, only uORFs and stem-loop structures (Pesole, Mignone, Gissi, Grillo, Licciulli and Liuni, 2001) will be discussed further. Reviews on the other regulatory elements of mRNA include Pesole, Grillo, Larizza and Liuni (2000), Mignone et al. (2002) and Wilkie et al. (2003).

4.2 Upstream Open Reading Frame (uORF)

In this section, we describe the importance of *upstream ORFs* (uORFs) in the regulation of gene expression. The uORFs are the main focus of this thesis. The stem-loop structures will be discussed in relation with uORFs in Chapter 6.

A uORF is identified by the presence of both a start codon before (i.e., *upstream of*) the start codon of the coding sequence and a stop codon in the same reading frame. Zero or more uORFs may present at a 5′ UTR of an mRNA. Research has revealed that several transcribed uORFs regulate the translation process (i.e., the uORFs are *functional*) (Vilela and McCarthy, 2003; Vilela et al., 1999; Hinnebusch, 1997; Fiaschi et al., 1997; Iacono et al., 2005), while a few others do not (i.e., the uORFs are *non-functional*) (Morris and Geballe, 2000; Krummeck et al., 1991).

Higher frequency of transcribed uORFs were found in genes with criti-

cal roles, such as homeobox (development-controlling) genes, proto-oncogenes (whose mutation or overexpression can lead to cancer), growth factors, and transcription factors (Kwon, Lee, Lee, Edenberg, ho Ahn and Hur, 2001). This suggests that uORFs are important to living organisms and has attracted researchers' interest, as evidenced by recent uORF-related publications, such as Zhang and Dietrich (2005a), Crowe, Wang and Rothnagel (2006), Neafsey and Galagan (2007), and Cvijovic, Dalevi, Bilsland, Kemp and Sunnerhagen (2007); these publications will be discussed both in this chapter and in the next three.

4.2.1 uORFs and Post-transcriptional Regulation

It is now widely accepted that the translation follows the *scanning model* (Kozak, 1999; Kozak, 2002). According to this model, the translation machinery (the *ribosome*) scans along the mRNA from $5'$ UTR to $3'$ UTR to translate the coding sequence of mRNA. The simplest case of this model would be if there is nothing to interrupt the ribosome at the $5'$ UTR and that the first AUG that the ribosome meets is the AUG of the coding sequence. However, in some cases translation was found to be more complex than the simplest case.

By following the scanning hypothesis, we briefly discuss several proposed mechanisms of how uORFs regulate the translation of the coding sequence. The material discussed in this section is the basis for some of the background knowledge used in our ILP experiments described in later chapters.

Leaky Scanning and Reinitiation

It has been observed that in some eukaryotic genes, the first AUG in the mRNA is not the AUG of the coding sequence. In several cases, the ribosome skips over one or more AUGs before finally starting the translation; this situation is known as *leaky scanning*. This can happen due to several things, such as if the sequences surrounding (*the context of*) AUG codons are not optimum. Research on mammals, done by Kozak (Kozak, 1981; Kozak, 2005), has shown

that AUG context has influence on the recognition of AUG by the ribosome. The optimum AUG context was found to be A or G in position -3 and G in position +4, where the A of AUG is position +1[1]. In their paper, Baim and Sherman (1988) gave evidence that A or G in position -3 are also favourable context for yeast. Earlier research, such as Morris and Geballe (2000) and Meijer and Thomas (2002), that studied upstream AUGs also revealed that when the ribosome recognises an upstream AUG, the ribosome may start translation and then:

- terminate and reinitiate;
- terminate and leave the mRNA, resulting a reduced translation of the coding sequence.

This can be illustrated by the uORFs of yeast gene *GCN4*.

In relation with uORFs, *GCN4* is one of the most studied genes in yeast, and is an interesting example. It has four uORFs (see Figure 4.6). uORF2 has length 3 codons (including start and stop codons). Each of the others has length 4 codons.

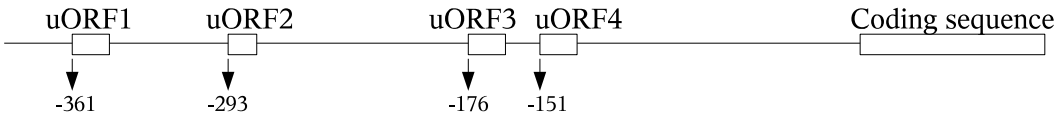

Figure 4.6: *GCN4* and its uORFs.

uORF1 of *GCN4* has optimum AUG context (A on position -3 and G in position +4) and thus efficiently recognised by the ribosome. As an implication, the ribosome translates this uORF. However, the impact of uORF1 by itself to the translation of the coding sequence is much smaller when compared to uORF4 by itself. This difference was found to be partly due to their

[1]There is no position 0; position -1 and +1 are next to each other.

distances to the coding sequence but mainly due to the context of their stop codons (Abastado, Miller, Jackson and Hinnebusch, 1991; Grant and Hinnebusch, 1994). Interestingly, when the two uORFs are both present, the impact of uORF4 on translation becomes less (Gaba, Wang, Krishnamoorthy, Hinnebusch and Sachs, 2001). This interesting phenomenon can be explained in relation with the function of *GCN4* protein.

The protein products of *GCN4* are involved in activating amino acid production in the cells. Therefore, the coding sequence of *GCN4* is only translated efficiently when the cells are short of amino acids (Abastado et al., 1991; Grant and Hinnebusch, 1994; Hinnebusch, 1997). The uORFs of *GCN4* act as regulator for the translation of the coding sequence. When amino acids are in high supply (which also means high energy for ribosome), the ribosome reinitiates soon after translating uORF1. Depending on how soon the reinitiation takes place, the ribosome will translate one of the other uORFs and then leave the mRNA. When amino acids are scarce (which also means less energy for ribosome), it will take a while before the ribosome will be ready to translate again. Thus the ribosome just passes through uORF2-4 (i.e., leaky scanning happens) and only reinitiates translation when it arrives at the coding sequence; this will then trigger production of amino acid.

Encoding Bioactive Peptides

Unlike *GCN4* which has four short uORFs, the yeast gene *CPA1* only has one uORF with length 26 codons (including start and stop codons). As a regulatory element, the uORF of *CPA1* works differently than those of *GCN4* (Delbecq, Werner, Feller, Filipkowski, Messenguy and Piérard, 1994; Gaba et al., 2001). Neither the distance between uORF and the coding sequence nor the sequence context of the uORF were found to be that essential in regulating translation in *CPA1* mRNA (Delbecq et al., 1994). Instead, the peptide encoded by the *CPA1* uORF was suggested to be the important element for reducing the translation

of *CPA1* coding sequence (Delbecq et al., 1994).

uORFs which may encode bioactive peptides (like that of *CPA1*) have also been identified in other organisms, such as human and mouse (Crowe et al., 2006).

Stimulating mRNA Decay

Some uORFs have been observed to reduce the expression of the coding sequence by stimulating mRNA decay. An example of this is the uORFs of yeast gene *YAP2* (Vilela, Linz, Rodrigues-Pousada and McCarthy, 1998; Vilela et al., 1999; Meijer and Thomas, 2002).

Stress-Mediated Regulation

As previously explained, the mechanism that *GCN4* uORFs take to regulate the expression of *GCN4* coding sequence depends on the supply of amino acids. Beside *GCN4*, there are also other genes whose response to stress conditions are regulated by uORFs. In Vilela et al. (1998), it was shown that uORFs regulate *YAP2* stress response to *Cadmium* (a heavy metal).

4.2.2 uORFs and Staying Healthy

In Section 4.2.1, we briefly described how uORFs may control the efficiency of translation of the coding sequence. In living organisms, this mechanism is important as certain proteins are not needed to be synthesised at high rates and maybe harmful if they were (Kozak, 1991; Meijer and Thomas, 2002). In this section, we present a few examples where uORFs play important roles in keeping organisms healthy. More examples of similar cases can also be read from Willis (1999) and Xu, Rabadan-Diehl, Nikodemova, Wynn, Spiess and Aguilera (2001).

Proinsulin

Proinsulin is a precursor of hormone *insulin*. The latter is produced and secreted in adult pancreas. However, proinsulin is also expressed in embryonic cells (before the pancreas is developed); in this case, proinsulin is essential to promote cell survival. The amount of proinsulin needed in embryonic cells is much smaller when compared to what is needed in adult pancreas. To know what regulates the production of proinsulin, Hernández-Sánchez, Mansilla, de la Rosa, Pollerberg, Martínez-Salaz and de Pablo (2003) studied the proinsulin mRNAs in embryos and pancreas of chicken. They observed that the embryonic proinsulin mRNA contains an extended region at $5'$ end when compared to the pancreatic one. The extended region contains two upstream AUGs which give two uORFs to an embryonic proinsulin mRNA. Their findings also showed that too much proinsulin in an embryo generated abnormal development and, more importantly, it is the presence of uORFs that keeps the production of proinsulin at low level in a chick embryo.

MDM2 Oncogene[2]

The level of *MDM2* proteins were found to be high in various human tumor cells. This overexpression was due to an increased translation of *MDM2* coding sequence (Landers, Cassel and George, 1997). In their study, Landers et al. (1997) compared *MDM2* mRNAs in normal human cells and in human tumor cells. They found that the *MDM2* mRNAs of the tumor cells are shorter than those of normal cells.

Extending Landers et al.'s (1997) work, Brown, Mize, Pineda, George and Morris (1999) investigated what keeps the production of *MDM2* proteins in normal human cells low. They observed that an *MDM2* mRNA of normal human cells contains two uORFs. Furthermore, their experiments demonstrated

[2]An oncogene is a gene whose product can contribute to the transformation of cells to a malignant phenotype (Weaver, 2005).

that each of these uORFs by itself has a medium impact in reducing the translation of *MDM2* coding sequence. But, the two uORFs together reduce the translation of *MDM2* coding sequence by more than 95%.

Thrombocythaemia

Thrombocythaemia[3] is a condition where blood contains too many *platelets*, a type of blood cell involved in blood clotting. People with this condition have a higher risk of developing a blood clot, a stroke or heart attack. The production of platelets in the blood cells is controlled by the hormone expressed from the gene thrombopoietin. According to Kozak's (1999) review, based on the studies of Wiestner et al. (1998), under normal conditions, the uORFs of thrombopoietin mRNA act to limit the translation of the thrombopoietin coding sequence and thus limit the production of the platelets in the blood cells. When uORFs are somehow eliminated from the thrombopoietin mRNA, the translation of thrombopoietin coding sequence is increased and thus the amount of the platelets, causing thrombocythaemia.

4.3 A Need to Identify Functional uORFs

Although uORFs have been estimated to occur in up to 25% of eukaryotic 5′ UTRs (Neafsey and Galagan, 2007), transcribed uORFs have only been verified in a small number of genes in several organisms. From this data, a partial understanding of how uORFs regulate protein expression has been achieved. As more and more uORFs have been found in the mRNA of genes with critical roles, it has become important to get a complete understanding of how uORFs are involved in the regulatory mechanism of gene expression. To be able to draw a complete understanding of the mechanism, we would expect that a

[3]Definition is from `http://www.cancerhelp.org.uk/help/default.asp?page=6412`, accessed on 12 September 2007.

large number of functional uORFs would be needed.

4.3.1 Experimental Work

Until recently studies on uORFs have been largely limited to lab-based experiments. The most direct test to verify that uORFs are transcribed and whether they are functional is by comparing the amount of mRNA and the amount of protein produced from the main gene in its proper chromosomal context with and without site-specific mutation(s) on the uORF(s) of interest[4]. Overall, these experiments, to verify that uORFs are transcribed and whether they are functional, are costly and time-consuming (≈ 4 man-months per gene, Sunnerhagen, P., personal communication. 4 October 2005). As a result, the simplest approach to searching for functional uORFs, i.e., by sampling genes at random and testing their uORFs in the laboratory, is simply not effective, even for the simplest eukaryotic (the yeast *Saccharomyces cerevisiae*) genome. *S. cerevisiae* genome has ≈ 6000 genes. It was suggested that no more than 10% of yeast genes will have one or more functional uORFs and each of these genes will on average have two functional uORFs (Sunnerhagen, P., personal communication. 26 August 2005). Thus, if one searched for functional uORFs by selecting genes at random and testing them in the lab, then on average it would take $\approx$20 man-months to find a single functional uORF. Therefore, an *in silico* prediction method which can help in selecting sets of candidate functional uORFs for experimental studies is essential.

4.3.2 Computational Work

Although a large number of genomic (DNA) sequences are now available, the task of computationally identifying functional uORFs is still very challenging. The reason is as follows. As explained in Section 4.2 on page 53, a functional

[4]The site-specific mutation is usually done on one of the bases of a uORF's start codon to remove the uORF.

uORF is a transcribed uORF which can regulate the translation of a coding sequence. A uORF can be transcribed if it lies at the 5′ UTR of an mRNA. However, as mentioned in Section 3.3 on page 45, the transcription start sites (the locations where the transcription starts) are often not known. Determining the start of 5′ UTRs computationally becomes even more difficult due to the fact that some genes have multiple transcription start sites (Miura, Kawaguchi, Sese, Toyoda, Hattori and Morishita, 2006). This situation makes the task of determining which genes contain uORF(s) at their 5′ UTRs very challenging, not to mention identifying which of these uORFs are functional.

This PhD Project

In an attempt to select sets of candidate functional uORFs for experimental studies, we decided to use inductive logic programming (ILP), a machine learning technique, to automatically generate a set of rules (a model) which can be used to identify functional uORFs (i.e., uORFs which can regulate gene expression) and then to use the resulting model to predict novel functional uORFs. For this task, we chose to work on the yeast *Saccharomyces cerevisiae* genome. This yeast is famously known as baker's and brewer's yeast.

Although ILP has been successfully applied to a diverse range of real-world problems, we are not aware of any previous work which explored using ILP (or even machine learning in general) to identify which uORFs regulate gene expression. Unlike other machine learning techniques, ILP is able to utilise existing knowledge from domain experts and/or the literature. Moreover, all ILP's input and output can be easily translated into English. Consequently domain experts can help with the selection and integration of potential helpful knowledge and the final dissemination of discoveries to the wider scientific community.

The yeast *Saccharomyces cerevisiae* is one of the most and best studied biological models. Although yeast is simpler than other eukaryotes, it has the

characteristics of complex eukaryotes. The fact that the size of its genome is relatively small compared to other eukaryotes makes it more appealing. Moreover, yeast can be grown very fast making lab experimental studies, to verify whether particular uORFs do indeed regulate translation, feasible.

Other Work

Beside ours, there are very few computational studies on uORFs. The closest study to ours is a study on yeast uORFs published in August 2007 by Cvijovic et al. (2007). Another study on yeast uORFs was published in October 2005 by Zhang and Dietrich (2005a). We will discuss the differences between our work and that of Zhang and Dietrich and that of Cvijovic et al. in Chapters 5 and 7.

Neafsey and Galagan (2007) published their studies on uORFs in the genome of fungal pathogen *Cryptococcus neoformans* in May 2007. Their work was really intended to find the proportion of uORFs conserved in four strains of *C. neoformans*. Similar to the work of Neafsey and Galagan, Crowe et al. (2006) looked for conserved uORFs between human and mouse genomes.

4.4 Summary

With more and more evidence, it is clear that some uORFs play important roles in gene regulation. However, a complete understanding of the mechanism of how uORFs regulate gene expression is still unclear. The main reason for this is because lab-based experiments to identify functional uORFs are extremely expensive and time-consuming. Therefore, an *in silico* prediction method which can help in selecting sets of candidate functional uORFs for experimental studies is essential. This PhD project sets out to develop such a method using ILP as the learning method and the yeast *S. cerevisiae* as the model organism. ILP has never been explored for this task previously.

In the next chapter, we will present the first approach that we took towards achieving this task.

Chapter 5

A First Step towards Learning which uORFs Regulate Gene Expression

The background to this thesis has been laid out in Chapters 2 - 4. This chapter and the next two will describe our *in silico* experiments to learning which upstream open reading frames (uORFs) regulate gene expression in the yeast *Saccharomyces cerevisiae*. To the best of our knowledge, this is the first time that the use of ILP has been explored for this learning task.

We begin this chapter by explaining how we represented the uORFs problem as an ILP learning task and how we transformed the biological sequence data into examples and background knowledge (Section 5.1). We present our training method in Section 5.2 and the result of training stage in Section 5.3. We describe our first attempt to predict novel functional uORFs in Section 5.4 and analyse the result in Section 5.5. Some related work are discussed in Section 5.6. Conclusions are in Section 5.7.

The results from the experiments in this chapter were published in the Journal of Integrative Bioinformatics (Selpi, Bryant, Kemp and Cvijovic, 2006).

5.1 Transforming Sequence Data into Examples and Background Knowledge

As explained in Chapter 4, a 5′ UTR may have zero or more uORFs. A uORF is identified by the presence of both a start codon before (i.e., *upstream of*) the start codon of the coding sequence and a stop codon in the same reading frame, as illustrated in Figure 4.5 on page 52. Thus ideally, in order to find functional uORF(s), one should look for uORF(s) in the 5′ UTR sequences. However, at the time we started this study, the lengths of 5′ UTRs in the yeast *S. cerevisiae* were only known for a small number of genes. Only 248 genes could be assigned unambiguously from European Molecular Biology Laboratory (EMBL) database[1]. Therefore, for the experiments in this chapter we used ORF Finder (Stothard, 2000) to search for open reading frames (ORFs) in the *intergenic* (between two genes) sequences of the yeast *S. cerevisiae*. An ORF is defined as a series of non-overlapping codons which starts with a start codon and ends with a stop codon. The lengths of intergenic sequences were taken from the supplementary material of Philippakis, He and Bulyk (2005). This step gives a collection of 51,904 crude uORFs from 5,602 *S. cerevisiae* genes. We described this set as crude because it consists of uORFs which can be transcribed within mRNAs and those which cannot; uORFs which can regulate gene expression will only be found among the transcribed ones. The task of collecting uORFs by making use of ORF Finder and Philippakis et al.'s (2005) data was done by our collaborator (Cvijovic, 2005).

17 of these 5,602 genes have been well-studied and are documented to have uORFs transcribed within their mRNAs, as summarised by Vilela and McCarthy (2003) and Cvijovic (2005). The detailed composition of the data used for experiments in this chapter is summarised in Table 5.1. In October 2005, Zhang and Dietrich (2005a) reported 15 additional genes which contain uORFs

[1] `ftp://ftp.ebi.ac.uk/pub/databases/UTR/data/5UTR.Fun_nr.dat.gz`

Table 5.1: Detailed composition of uORFs obtained using ORF Finder.

Number of Genes		Transcribed uORFs			Other uORFs (Not known if transcribed)
		Functional	Non-functional	Unknown	
17 studied genes	8	20	-	-	269
	2	-	2	-	8
	7	-	-	8	103
5,585 other genes		-	-	-	51,494
5,602 genes		20	2	8	51,874

transcribed within their mRNAs. However, we did not include the uORFs from these 15 genes for our training, rather we used them for the purpose of analysing the results of our ILP experiments (see Section 5.4).

Since our goal is to learn how to recognise which uORFs regulate gene expression, we can consider this learning task to be a classification problem. Ideally, a typical classification system in ILP (or machine learning in general) learns from a mixture of positive and negative examples. In this domain, positive examples would be uORFs that are transcribed and regulate gene expression (i.e., functional) and negative examples would be uORFs that are transcribed but do not regulate gene expression (i.e., non-functional). The uORF data from 5,585 genes (see Table 5.1) are all unlabelled. Hence, for the training stage in this study, only the uORF data of the 17 studied genes were used.

As summarised in Table 5.1, among the uORF data of the 17 studied genes, 20 uORFs have been verified experimentally as functional. These were used as positive examples. Cvijovic (2005, p. 32) pointed out that there are only 2 uORFs from 2 genes which have been verified to be non-functional. Therefore, there were only 2 negative examples in our data set. The rest of the transcribed uORFs (8 uORFs) and all other uORFs (which are not known to be transcribed) for those 17 genes ($269 + 8 + 103 = 380$ uORFs) were used

as randoms. Here randoms are data that are likely to be negative, although there is still a small probability that the data are positive. The detailed uORF composition from 17 studied genes within the collection obtained using ORF Finder is shown in Table 5.2.

Table 5.2: Detailed uORF composition from 17 studied genes within the collection obtained using ORF Finder.

Gene Name[a]	Systematic Name[a]	Transcribed uORFs	Other uORFs	Positive Examples	Negative Examples	Random Examples
CLN3	YAL040C	1	7	1	-	7
GCN4	YEL009C	4	15	4	-	15
HAP4	YKL109W	2	26	2	-	26
TIF4631	YGR162W	5	202	5	-	202
YAP1	YML007W	1	3	1	-	3
YAP2	YDR423C	2	-	2	-	-
HOL1	YNR055C	1	15	1	-	15
PET111	YMR257C	4	1	4	-	1
SCO1	YBR037C	1	4	-	1	4
CBS1	YDL069C	1	4	-	1	4
INO2	YDR123C	1	9	-	-	10
PPR1	YLR014C	1	2	-	-	3
URA1	YKL216W	1	13	-	-	14
LEU4	YNL104C	1	12	-	-	13
RCK1	YGL158W	2	49	-	-	51
DCD1	YHR144C	1	17	-	-	18
SCH9	YHR205W	1	1	-	-	2
17 Genes		30	380	20	2	388

[a]Names are taken from SGD (`http://www.yeastgenome.org`).

Given the characteristics of the data (i.e., the number of negative examples is too few compared to the positive examples and there is an abundance of random examples), we explore the positive-only setting (Muggleton, 1996) of CProgol (Muggleton, 1995) version 4.4 (Muggleton and Firth, 2001). CPro-

gol4.4 is an ILP system which has been applied in another domain with similar characteristics as uORF domain (Muggleton et al., 2001). The positive-only setting of CProgol allows learning from positive and random examples.

CProgol4.4 was instructed to learn a predicate `has_functional_role/1` from a set of training examples. **Positive** examples were represented as ground unit clauses of the predicate `has_functional_role(X)`, where `X` is a uORF ID. A uORF ID is a composite of the systematic name of the gene to which the uORF is associated with (e.g., those listed in second column of Table 5.2) and a uORF identifier (e.g., uORF1, uORF2, etc.). The set of positive examples was divided into two parts, with two thirds (14 uORFs) of the data set used for training and the remaining one third (6 uORFs) used for testing. The 388 **random** examples were also partitioned, with two thirds used for training and the remainder used for testing.

In addition to positive and random examples, the ILP system was provided with extensional and intensional background knowledge.

Extensional Background Knowledge. Vilela and McCarthy (2003) and Cvijovic (2005) suggested several important features that can determine the impact of a uORF on post-transcriptional gene expression, such as: the distance of the uORF to the start of the coding sequence in bases; the sequence context (the frequency of AU and GC base-pairs) upstream of (before) the uORF's start codon and downstream of (after) the uORF's stop codon; and the length[2] of the uORF in codons. $5'$ UTR related properties, such as the number of uORFs predicted by ORF Finder in the intergenic sequence, the length of intergenic sequence, and the relationship between UTR and uORF were also included (see Table 5.3).

Intensional Background Knowledge. The declarative rules shown in Table 5.4 capture concepts that are potentially useful for helping to identify

[2]When a uORF is overlapped with the coding sequence, the length of the uORF is only measured up to the start of the coding sequence.

Table 5.3: Predicates representing background knowledge of uORFs and UTRs. A set of ground unit clauses was generated for each predicate.

`uORF(X,Y,Z)`	represents uORF `X` which starts `Y` nucleotides upstream of the coding sequence, with length `Z` codons.
`utr(X,Y,Z)`	represents UTR `X` which has `Y` uORF(s) within `Z` nucleotides upstream of gene `X` (`Z` is the length of intergenic region).
`has_uORF(X,Y)`	represents relation between UTR `X` and uORF `Y`.
`belongs_to(X,Y)`	represents relation between uORF `X` and UTR `Y`.
`context(X,Y,Z)`	states that within the 20 nucleotides downstream of `X`'s stop codon, the frequency of AU is `Y` and the frequency of GC is `Z`.
`up_context(X,Y,Z)`	states that within the 20 nucleotides upstream of `X`'s start codon, the frequency of AU is `Y` and frequency of GC is `Z`.

Table 5.4: Intensional background knowledge.[a]

```
has_shortest_dist_in_UTR(UORF):-
    uORF(UORF,ShortestDist,_), belongs_to(UORF,UTR),
    setof(Dist,(has_uORF(UTR,UORFX),uORF(UORFX,Dist,_)),List),
    List = [ShortestDist|_].
has_shortest_len_in_UTR(UORF):-
    uORF(UORF,_,ShortestLen), belongs_to(UORF,UTR),
    setof(Len,(has_uORF(UTR,UORFX),uORF(UORFX,_,Len)), List),
    List = [ShortestLen|_].
gcrich_down_up(UORF):-
    context(UORF,Au,Gc), Gc > Au,
    up_context(UORF,A,G), G > A.
gcrich_down_aurich_up(UORF):-
    context(UORF,Au,Gc),  Gc > Au,
    up_context(UORF,A,G), G < A.
aurich_down_up(UORF):-
    context(UORF,Au,Gc), Gc < Au,
    up_context(UORF,A,G), G < A.
gcrich_up_aurich_down(UORF):-
    context(UORF,Au,Gc), Gc < Au,
    up_context(UORF,A,G), G > A.
```

[a] CProgol's built-in predicate setof(X,P,L) produces a list L of objects X that satisfy P. L is ordered ascending and duplicate items are eliminated.

functional uORFs and therefore might be included in the hypotheses induced by the ILP system. We matched the verified functional uORFs from Vilela and McCarthy (2003) and Cvijovic (2005) to the uORF data obtained using ORF Finder. From this, we observed that the majority of the functional uORFs are the ones closest to the coding sequences. Therefore, we defined `has_shortest_dist_in_UTR/1` that identifies whether a uORF is closer to the coding sequence than all others within the same UTR. Verified functional uORFs are often very short, so one might be interested to identify the shortest uORF of each gene. Hence, we defined `has_shortest_len_in_UTR/1`. Vilela et al. (1998) and Grant, Miller and Hinnebusch (1995) suggest that the sequence context of a uORF's start and stop codons have an impact on translation. Therefore, we defined the rules `gcrich_down_up/1`, `aurich_down_up/1`, `gcrich_down_aurich_up/1` and `gcrich_up_aurich_down/1` that examine the abundance of AU and GC base pairs immediately upstream and downstream of each uORF.

5.2 Generating a Model that Identifies Functional uORFs

In this experiment, we investigate whether ILP could automatically generate a model that identifies functional uORFs and whether this model, when used as a filter, could be more efficient than random sampling. The training set consists of 14 positives and 259 randoms and the test set consists of 6 positives and 129 randoms (Figure 5.1).

CProgol's parameters were set as follows: **posonly** is set to 'on' so that CProgol learns from positives and randoms only; **inflate** (gives a weighing to the examples) is set to 4,200%; **c** (the maximum number of atoms in the body of the rules constructed) is set to 6; **nodes** (the maximum number of nodes explored during clause searching) is set to 7,000; and **r** (the maximum depth

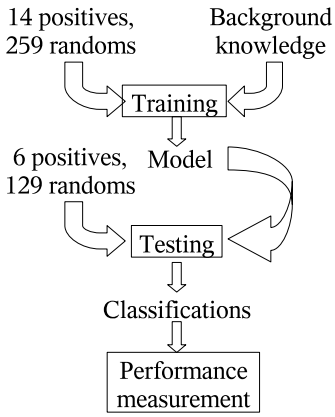

Figure 5.1: A summary of our experimental method.

of resolutions allowed when proving) is set to 700. More details on CProgol's parameters can be found in Section 2.4.1 on page 18.

4,200% was chosen as an inflate parameter following some preliminary experiments (not shown in this thesis), where several experiments were run with the same training set, test set, and the same CProgol settings except for the value of inflate parameter (70%, 101%, 200%, 300%, 400%, 500%, 1,000%, 1,200%, 4,200%, 10,000%, and 15,000%). In these preliminary experiments, the performance reached its best at 4,200% and stayed the same above 4,200%.

We defined the hypothesis space for CProgol 4.4 so that it can construct a definition for the target predicate `has_functional_role/1`. This was done by giving mode declarations (see Table 5.5). The types `uORF` and `utr` were declared by defining a set of ground unit clauses of the predicate `uORF(X)`, where `X` is a uORF ID; and a set of ground unit clauses of the predicate `utr(X)`, where `X` is a UTR ID. The types of `uORFlength`, `distancefromstart`, `intergeniclength` and `numberofuORF` were all defined as integer. Table 5.6 shows the resulting model.

Table 5.5: Mode declarations for generating a model that identifies functional uORFs.[a]

```
:- modeh(1,has_functional_role(+uORF))?
:- modeb(1,uORF(+uORF,-distancefromstart,-uORFlength))?
:- modeb(1,belongs_to(+uORF,-utr))?
:- modeb(1,utr(+utr,-numberofuORF,-intergeniclength))?
:- modeb(1,+distancefromstart=< #int)?  :- modeb(1,+uORFlength=< #int)?
:- modeb(1,+distancefromstart>= #int)?  :- modeb(1,+uORFlength>= #int)?
:- modeb(1,+distancefromstart= #int)?   :- modeb(1,+uORFlength= #int)?
:- modeb(1,+intergeniclength=< #int)?   :- modeb(1,+numberofuORF=< #int)?
:- modeb(1,+intergeniclength>= #int)?   :- modeb(1,+numberofuORF>= #int)?
:- modeb(1,+intergeniclength= #int)?    :- modeb(1,+numberofuORF= #int)?
:- modeb(1,has_shortest_dist_in_UTR(+uORF))?
:- modeb(1,has_shortest_len_in_UTR(+uORF))?
:- modeb(1,gcrich_down_up(+uORF))?
:- modeb(1,aurich_down_up(+uORF))?
:- modeb(1,gcrich_down_aurich_up(+uORF))?
:- modeb(1,gcrich_up_aurich_down(+uORF))?
```

[a] `modeh` describes the clauses to be used in the head of a hypothesis, and `modeb` describes the clauses to be used in the body of a hypothesis. The type `uORFlength` was printed as `codonlength` in the paper by Selpi et al. (2006).

Table 5.6: The model which predicts functional uORFs.

```
has_functional_role(A) :- uORF(A,B,C), B=<204.
has_functional_role(A) :- uORF(A,B,C), belongs_to(A,D), B=<409,
    C=<6, utr(D,E,F), F>=589.
```

English translation: A uORF has functional role if it satisfies at least one of the following rules.

- if its distance from the start of coding sequence is less than or equal to 204;
- if its distance from the start of coding sequence is less than or equal to 409, its length is less than or equal to 6, and the intergenic length is greater than or equal to 589.

5.3 Measuring Model Performance using Relative Advantage

The default performance measure in CProgol4.4 is predictive accuracy. However, this measure gives a poor estimate when used in a domain where positives are rare, which is the case in this uORF domain. Therefore, we do not use this performance measure.

Instead we adapted Relative Advantage (RA) (Muggleton et al., 2001, Appendix A). This uORF domain has the characteristics for which RA is claimed be useful. These include the fact that the proportion of positives (functional uORFs) in the example set is very small, while the proportion of positive examples in the population (the whole *S. cerevisiae* yeast genome) is not known, acquiring negatives is difficult (as this has to be verified via lab experiments) and a benchmark recognition method does not exist.

The idea behind using RA is to predict cost reduction in finding functional uORFs using the model compared to using random sampling. In this application domain, RA is defined as

$$RA = \frac{A}{B}; \text{ where}$$

- A = the expected cost of finding a functional uORF by repeated independent random sampling from a set of 51,904 crude uORFs and testing each uORF in the lab.

- B = the expected cost of finding a functional uORF by repeated independent random sampling from a set of 51,904 crude uORFs and analysing only those which are predicted by the model to be functional.

In order to use RA, the following parameters were used. The total number of uORFs considered was set to 51,904. The minimum number of functional uORFs was set to 20 as there are at least 20 known functional uORFs. The most probable number of functional uORFs in *S. cerevisiae* genome was set to

Table 5.7: A summary of classification and performance measurement of experiment generating a model which predicts functional uORFs (in Section 5.2).

Positives correctly classified as positives	3
Randoms falsely classified as positives	4
Positives falsely classified as randoms	3
Randoms correctly classified as randoms	125
mean RA	17.3

1,200. This assumption was made based on a prediction that no more than 10% of yeast genes will have functional uORFs and each of these genes will on average have two functional uORFs ($10\% \times 6000 \times 2 = 1,200$) (Sunnerhagen, P., personal communication. 26 August 2005).

A summary of the classifications made and the performance measurement from the experiment in Section 5.2 is presented in Table 5.7. Using our model as a predictor makes the search for novel functional uORFs 17 times more efficient than random sampling. Reducing the number of randoms that are falsely classified as positives is very important in this domain because verification via lab analysis is costly.

5.4 Predicting Novel Functional uORFs

Although our model (Table 5.6) looks simple, its mean RA value shows that the model makes the search for novel functional uORFs more efficient than using random sampling. Thus, it is expected that the positive-only setting of CProgol4.4 can help in predicting novel functional uORFs. To support this argument, an experiment was conducted to predict novel functional uORFs.

The method used (Figure 5.2) was the same as that described in Section 5.2 except that the positive and random examples from the 17 studied genes were

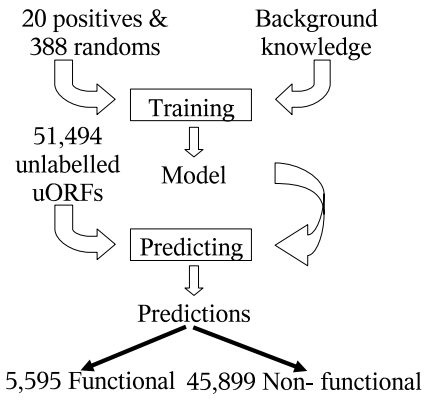

Figure 5.2: A summary of our first attempt to predict novel functional uORFs.

all used for training. Thus, the training set consists of 20 positives and 388 randoms from the 17 studied genes. The resulting model was then used to predict novel functional uORFs from 51,494 unlabelled uORFs (from 5,585 genes, see Table 5.1 on page 66). Table 5.8 shows the model generated from the experiment to predict novel functional uORFs. 5,595 out of 51,494 uORFs are predicted as functional uORFs by this model.

5.5 Discussion

5,595 out of 51,494 uORFs were predicted as functional uORFs by the model shown in Table 5.8. This result shows that our method was able to filter out almost 90% of the unlabelled uORFs used for prediction, of which majority would be negatives. However, 5,595 is still much larger than the predicted maximum number of functional uORFs in the yeast genome i.e., 1,200 (details in Section 5.3). This suggests that our rules were too general. This was because of the limited number of positive examples, the high degree of noise in the data due to using intergenic sequences and also because of the limited background knowledge. Nevertheless, some promising indications that our method has potential to help in selecting sets of candidate functional uORFs are given by

Table 5.8: The model generated from the experiment to predict novel functional uORFs.

```
has_functional_role(A) :- uORF(A,B,C), belongs_to(A,D), B=<204,
    utr(D,E,F), E>=207.
has_functional_role(A) :- uORF(A,B,C), belongs_to(A,D), B=<409,
    C=<6, utr(D,E,F), E>=5.
has_functional_role(A) :- belongs_to(A,B), utr(B,C,589).
has_functional_role(A) :- uORF(A,B,C), has_shortest_dist_in_UTR(A),
    C=<8, B>=23.
has_functional_role(A) :- uORF(A,57,B).
has_functional_role(A) :- uORF(A,250,B).
```

English translation: A uORF has functional role if it satisfies at least one of the following rules.

1. if its distance from the start of coding sequence is less than or equal to 204 and the UTR to which it belongs has at least 207 uORFs;
2. if its distance from the start of coding sequence is less than or equal to 409, its maximum length is 6 codons, and the UTR to which it belongs has at least 5 uORFs;
3. if intergenic length of its UTR to which it belongs is 589;
4. if it is the closest uORF to the coding sequence within its UTR, its length is less than or equal to 8 codons, and its distance from the start of coding sequence is greater or equal to 23;
5. if its distance from the start of coding sequence is 57;
6. if its distance from the start of coding sequence is 250.

comparing our predictions with experimental lab results from a study by Zhang and Dietrich (2005a).

Further to the 17 genes and 30 verified transcribed uORFs mentioned in Table 5.1, Zhang and Dietrich (2005a) have reported an additional 15 genes which contain 19 verified transcribed uORFs in the yeast *S. cerevisiae*. Their focus was to find additional genes which contain transcribed uORF(s). Therefore,

for many of these 19 newly verified transcribed uORFs, they did not study whether they are functional or not. However, as uORFs which can regulate gene expression are among the transcribed ones, we used their findings for the purpose of analysing the results of our ILP experiments.

Zhang and Dietrich (2005a) provide some evidence that our rules may be biologically meaningful. In their paper, they wrote "We observed that uORFs are present in over 95% of 250 bp 5′ upstream regions of *S. cerevisiae*". But for their analysis, a 210 bp (base pair) 5′ upstream region was used as the upper boundary to eliminate "spurious potential uORFs". Zhang and Dietrich's observation suggested that functional uORFs are likely to be found within 250 bp upstream of the start of coding sequence (because the functional uORFs have to be transcribed). Our rules (i.e., rules 1, 5, and 6 in Table 5.8) reflect that condition.

Rules 2 and 4 in Table 5.8 suggested that functional uORFs are short, with a maximum length of 8 codons. Of course, this is true for many of the currently known functional uORFs that were used in our data. However, it is interesting to note that many of the newly verified transcribed uORFs that were reported by Zhang and Dietrich (2005a) are shorter than 8 codons (14 of 19 uORFs are shorter than 8 codons).

The feature "being closest to the coding sequence" is present in rule 4 in Table 5.8. In our experiments, this feature has helped to filter out many of the unlabelled uORFs which probably would not even be transcribed in reality. We think that this feature may not be that important when the training data are extracted from the 5′ UTR sequences. This is because the majority of genes, which have uORFs in their UTRs, will probably only have one uORF each. 13 of the 15 genes that were reported by Zhang and Dietrich (2005a) only have one uORF each.

Overall, given the limited data and background knowledge, we think that our rules are very reasonable. Of the 15 genes reported by Zhang and Dietrich

(2005a), our model predicts that 12 will have functional uORFs, and that 13 of the 19 transcribed uORFs will be functional (Table 5.9).

Table 5.9: Predictions made using the model in Table 5.8 for the 15 genes reported by Zhang and Dietrich (2005a).

Gene Name	Systematic Name	uORF's Position	uORF's Length	uORF Identifier		Predicted as Functional
				Z&D[c]	This study[d]	
ARV1	YLR242C	-125	12	uORF1	uORF5	No
		-108	3	uORF2	uORF4	Yes
		-40	7	uORF3	uORF6	Yes
ECM7[a]	YLR443W	-15	5	uORF	-	-
HEM3	YDL205C	-129	9	uORF	uORF8	No
RPC11	YDR045C	-60	4	uORF	uORF5	Yes
AVT2	YEL064C	-11	4	uORF	uORF7	Yes
TPK1	YJL164C	-42	5	uORF	uORF4	Yes
MBR1	YKL093W	-70	7	uORF	uORF5	Yes
APC2	YLR127C	-27	5	uORF	uORF3	Yes
SPE4	YLR146C	-41	6	uORF	uORF5	Yes
SPH1	YLR313C	-25	4	uORF	uORF3	Yes
IMD4[a,b]	YML056C	-99	14	uORF	-	-
SLM2	YNL047C	-110	24	uORF1	uORF11	No
		-84	6	uORF2	uORF9	Yes
		-70	4	uORF3	uORF10	Yes
FOL1	YNL256W	-65	4	uORF	uORF4	Yes
WSC3	YOL105C	-50	7	uORF	uORF4	Yes
MKK1	YOR231W	-71	10	uORF	uORF5	No

[a]No uORF with the same position and length in our data set.

[b]Our model predicts uORF3 (in our data set) of *IMD4* as functional.

[c]uORF identifiers used in Zhang and Dietrich (2005a).

[d]uORF identifiers used in the supplementary material of Selpi et al. (2006).

5.6 Related Work

The work presented here uses a machine learning approach to learn functional uORFs in the yeast *S. cerevisiae*. We are not aware of any previous work of this kind. However, there is other work where machine learning methods have been used to investigate other aspects of post-transcriptional regulation. There is also work using other methods to investigate the regulatory role of uORFs in mammalian species, and work using other computational approaches to investigate the regulatory role of other UTR features in yeast.

Machine learning methods have been used for predicting translation initiation sites. Zeng, Yap and Wong (2002) and Tzanis and Vlahavas (2006) used feature generation and feature selection with standard classifiers such as decision trees, artificial neural networks, naïve Bayes, and support vector machines, while Li and Jiang (2005) have used edit kernels for support vector machines. However, we are not aware of any previous work applying machine learning to the problem of identifying functional uORFs.

Crowe et al. (2006) have identified uORFs of over 20 codons in length that are conserved in human and mouse genomes. Those uORFs that are conserved between human and mouse are predicted to code for bioactive peptides. They cite studies that suggest that some of these peptides play a role in regulation. In this work we do not place a lower limit on the lengths of uORFs that are considered, and the prediction model does not depend on sequence conservation across species.

Kwon et al. (2001) have carried out experimental work to investigate the regulatory role of uORFs and secondary structures in 5′ UTRs. They carried out site-directed mutagenesis studies of human *ADH5/FDH* and *Myf6* genes, measuring the RNA transcripts, investigating the interactions between mRNA and proteins involved in translation, and analysing the RNA secondary structures of the 5′ UTRs. Their results suggest that uORFs and stem-loops in the 5′ UTR can reduce translation of the coding sequence.

While the related work mentioned above has examined the regulatory role of 5′ UTRs in mammalian species, Ringnér and Krogh (2005) have carried out computational studies to investigate the regulatory role of secondary structure in yeast 5′ UTRs. They have computed the folding free energies of the 50 nucleotides immediately upstream of the coding sequence for all verified genes in *S. cerevisiae* and have found that "weakly folded 5′ UTRs have higher translation rates, higher abundances of the corresponding proteins, longer half-lives, higher numbers of transcripts, and are upregulated after heat shock".

The work of Kwon et al. (2001) and Ringnér and Krogh (2005) gave us an idea to extend our study, that is to consider additionally the locations of uORFs with respect to predicted secondary structure in the 5′ UTRs (see Chapter 6).

5.7 Conclusions

In this chapter, for the first time ILP was used for learning which uORFs regulate gene expression in the yeast *Saccharomyces cerevisiae*. The characteristics of uORF data (i.e., the number of negative examples are too few compared to the positive examples, and there is an abundance of random examples) makes the uORF domain very challenging. It is shown here that the positive-only setting of an ILP system, CProgol4.4, can be used to automatically generate rules that when used as a predictor can make the search for novel functional uORFs 17 times more efficient than using random sampling.

The rules are simple and easy to understand. Moreover, some of the rules are supported by the literature. However, the rules appear to be too general. This, we believe, is because of the limited number of positive examples, the high degree of noise in the data due to using intergenic sequences and also because of the limited background knowledge. As an implication, we suspect that some of the predictions made here are false positives, that is to say that some of the uORFs predicted as functional by our hypotheses here may actually be non-

functional. In the next chapter, we will investigate whether making background knowledge of mRNA secondary structures available to the ILP learner leads to a better performance.

From an evaluation point of view, this work has some shortcomings. First, we only used one ordering of positive examples. However, the ILP system that we used, CProgol4.4, uses a covering approach. This means that CProgol4.4 repeatedly generalises the positive examples in the order in which they are input to generate its rules. Thus, CProgol4.4 may generate different rules when given different orderings of positive examples. Second, the performance was measured after one execution of stratified holdout method. While one holdout execution is a valid and a common evaluation method, repeating holdout several times will reduce sampling bias and thus will give a better estimation of the predictive capability of the hypotheses. These shortcomings will be addressed in Chapter 6.

Chapter 6

Incorporating mRNA Secondary Structure in Learning Functional uORFs

In the previous chapter, we have shown that ILP was able to automatically generate a model (a set of rules) which makes searching for novel functional uORFs in the yeast *S. cerevisiae* more efficient than random sampling. The rules were simple and easy to understand, but appeared to be too general. This is due not only to the limited number of positive examples and the high degree of noise in the data, two problems of which cannot be easily rectified (see Section 5.1 on page 65), but also due to the limited background knowledge. Relevant background knowledge has been shown to be effective in helping ILP to produce good models (Srinivasan, King and Bain, 2003). In this chapter, we investigate whether incorporating mRNA secondary structure as background knowledge will increase the performance of the resulting rules in recognising functional uORFs in the yeast *S. cerevisiae*.

We describe why we consider mRNA secondary structure in Section 6.1. The experimental method, including how we incorporate mRNA secondary structure as background knowledge, is detailed in Section 6.2. We present our

results and analysis in Section 6.3 and conclude in Section 6.4.

6.1 Secondary Structure of mRNA

mRNA secondary structure is a stem-loop structure that is formed when the transcribed sequence contains an inverted repeat sequence. Some basic forms of the stem-loop structures are shown in Figure 4.2 on page 50.

In the context of gene regulation, mRNA secondary structure has been studied over a number of years. Kozak (1986), Kozak (2005), Baim and Sherman (1988), Laso, Zhu, Sagliocco, Brown, Tuite and McCarthy (1993), Pesole et al. (2000), and Meijer and Thomas (2002) are all in agreement that a stable secondary structure at 5′ UTR region can regulate the expression of the coding sequence. These studies were done on genes from different organisms, such as rat, rabbit, and yeast. The scanning model (see also Section 4.2.1 on page 54) suggests that the secondary structure at the 5′ UTR region must be unwound, by special proteins called *helicases*, before the translation machinery can translate the coding sequence (Marsden, Nardelli, Linder and McCarthy, 2006). The more stable a secondary structure, the more energy is needed to unwind it and the more it can inhibit translation. The influence of mRNA secondary structure on translation in *S. cerevisiae* has been suggested to be greater than on translation in higher eukaryotes (Baim and Sherman, 1988).

Beside the work which only studied mRNA secondary structures, there is also other work which studied both mRNA secondary structures and uORFs in the context of gene regulation. Wang and Wessler (2001), based on their study on a maize gene, concluded that uORF and mRNA secondary structure are regulating the translation of the coding sequence independently. However, Kwon et al.'s (2001) results, based on studies on human genes, are somehow rather different; the present of secondary structure seems to increase the recognition of uORF's start codon and this affects the translation of the coding sequence. The

difference between the conclusions of Wang and Wessler's (2001) and those of Kwon et al.'s (2001) leave open the question whether mRNA secondary structure influences uORF's ability to regulate translation. This motivates us to do this study; to test whether mRNA secondary structure could help in recognising known functional uORFs in the yeast *S. cerevisiae*.

6.2 Experimental Method

In this experiment, we want to test whether incorporating mRNA secondary structure as background knowledge could help when learning which uORFs in yeast are functional. Thus, we design experiments with and without mRNA secondary structure as part of the background knowledge. In Section 6.2.1, we describe the common method that is used in all experiments with and without mRNA secondary structure. The details of how we incorporate mRNA secondary structure as background knowledge is described in Section 6.2.2.

6.2.1 Common Method Used in Experiments With and Without mRNA Secondary Structure

The learning method that we used is the positive-only (Muggleton, 1996) setting of CProgol (Muggleton, 1995) version 4.4 (Muggleton and Firth, 2001). The data set consists of positive and random examples from 17 well-studied genes used in Chapter 5 (Table 5.2 on page 67). The ILP learner was instructed to learn a predicate `has_functional_role/1` from a set of training examples. Positive examples were represented as ground unit clauses of the predicate `has_functional_role(X)`, where `X` is a uORF ID. A uORF ID is a composite of the systematic name of the gene to which the uORF belongs (for example, YDR423C is the systematic name of gene *YAP2*) and a uORF identifier (e.g., uORF1, uORF2, *etc.*).

For the purpose of testing whether mRNA secondary structure could help

in learning yeast functional uORFs, we run experiments with and without incorporating mRNA secondary structure as background knowledge. Stratified 10-fold cross-validation was used to evaluate our method. This means that the set of positive examples is divided into ten roughly equal partitions and the same is done to the set of random examples. Each of these positive and random partitions are in turn used as a test set while the rest of the partitions are used as training set.

CProgol4.4 uses a covering approach (Muggleton and Firth, 2001) to generate a hypothesis. CProgol4.4 repeatedly generalises the positive examples in the order in which they are input (Muggleton and Firth, 2001). Thus, CProgol4.4 may generate different rules when given different orderings of positive examples. We want to test whether the performances of the rules generated using different orderings of positive examples are also different. Therefore, we run the stratified 10-fold cross-validation with and without incorporating mRNA secondary structure for 100 times. In each of the 100, we randomly permute the order of positive training examples.

The common extensional and intensional background knowledge used in the experiments with and without mRNA secondary structure are the same as those used in Chapter 5 (Table 5.3 on page 69 and Table 5.4 on page 69). Some common mode declarations were defined, to be used in experiments with and without mRNA secondary structure (Table 6.1). The types `uORF` and `utr` were declared by defining a set of ground unit clauses of the predicate `uORF(X)`, where `X` is a uORF ID; and a set of ground unit clauses of the predicate `utr(X)`, where `X` is a UTR ID. The types of `uORFlength`, `dist2gene`, `intergeniclength`, and `numberofuORF` were all defined as integer.

Some adjustments were made to the parameter settings used in Chapter 5 to allow CProgol to consider a larger hypotheses space (Table 6.2). The parameter **c** was increased from 6 to 10; **nodes** was increased from 7,000 to 50,000; and **h** was increased from 30 (default value) to 100.

Table 6.1: The common mode declarations for experiments with and without mRNA secondary structure.[a]

```
:- modeh(1,has_functional_role(+uORF))?
:- modeb(1,uORF(+uORF,-dist2gene,-uORFlength))?
:- modeb(1,+dist2gene=< #int)?          :- modeb(1,+uORFlength=< #int)?
:- modeb(1,+dist2gene>= #int)?          :- modeb(1,+uORFlength>= #int)?
:- modeb(1,+dist2gene= #int)?           :- modeb(1,+uORFlength= #int)?
:- modeb(1,has_shortest_dist_in_UTR(+uORF))?
:- modeb(1,has_shortest_len_in_UTR(+uORF))?
:- modeb(1,gcrich_down_up(+uORF))?
:- modeb(1,aurich_down_up(+uORF))?
:- modeb(1,gcrich_down_aurich_up(+uORF))?
:- modeb(1,gcrich_up_aurich_down(+uORF))?
:- modeb(1,belongs_to(+uORF,-utr))?
:- modeb(1,utr(+utr,-numberofuORF,-intergeniclength))?
:- modeb(1,+numberofuORF=< #int)?       :- modeb(1,+intergeniclength=< #int)?
:- modeb(1,+numberofuORF>= #int)?       :- modeb(1,+intergeniclength>= #int)?
:- modeb(1,+numberofuORF= #int)?        :- modeb(1,+intergeniclength= #int)?
```

[a] In essense, these mode declarations are the same as those in Table 5.5 on page 72. The type `dist2gene` was printed as `distancefromstart` in Table 5.5. `modeh` describes the clauses to be used in the head of a hypothesis, and `modeb` describes the clauses to be used in the body of a hypothesis.

Table 6.2: CProgol parameter settings used in experiments with and without mRNA secondary structure.

Name	Value	Meaning
posonly	on	CProgol turns on the positive-only learning mechanism.
inflate	4,200	This gives a weighing to the examples.
c	10	The maximum number of atoms in the body of the rules constructed.
nodes	50,000	The maximum number of nodes explored during clause searching.
h	100	The maximum depth of the stack used when proving before starting backtracking.
r	700	The maximum depth of resolutions allowed when proving. If exceeded, the proof is failed.

6.2.2 Method for Incorporating mRNA Secondary Structure

Getting mRNA Secondary Structure from Sequence Data

There are several pieces of software for predicting RNA secondary structure. Among these, two are widely used; they are RNAfold, which is part of the Vienna RNA Package (Hofacker, Fontana, Stadler, Bonhoeffer, Tacker and Schuster, 1994), and Mfold (Zuker, Mathews and Turner, 1999). In this work, we used RNAfold[1] because of its simplicity, easy installation on our school's server, and it has recently been used by Ringnér and Krogh (2005) to study the effect of mRNA folding free energies on post-transcriptional gene regulation.

Given a sequence, RNAfold will output a predicted secondary structure, in a *dot-bracket* format, as well as the predicted minimum free energy of that

[1]ViennaRNA-1.6.1 was downloaded from `http://www.tbi.univie.ac.at/~ivo/RNA/`

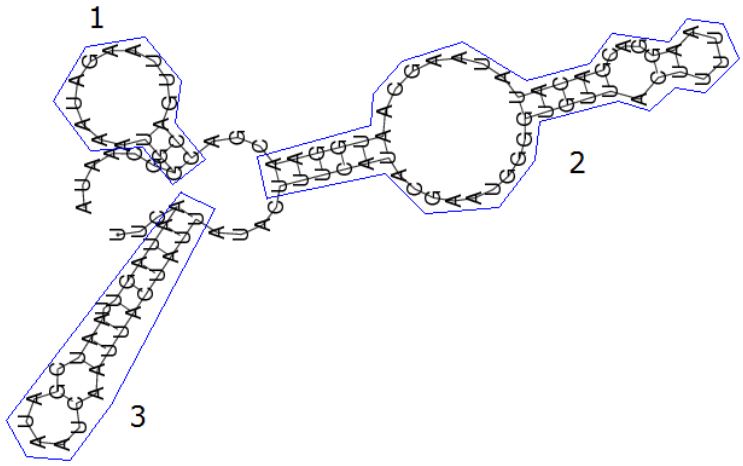

Figure 6.1: A predicted secondary structure of the 5′ UTR sequence and ten nucleotides of gene *YAP2* (YDR423C), made by RNAfold. The blue boxes were added to show how we view the structure as three stem-loop structures.

structure. In a dot-bracket format, matching brackets represent paired bases and dots represent unpaired bases (Hofacker et al., 1994, Appendix B).

For each of the 17 well-studied genes, the 5′ UTR sequence and the first ten nucleotides of the coding sequence was used as an input for RNAfold. The length of 5′ UTRs were taken from European Molecular Biology Laboratory (EMBL) database[2], where available, or, failing that, Vilela and McCarthy (2003). To get the predicted secondary structures, we used the default settings of RNAfold. The predicted secondary structures were then transformed into Prolog (Clocksin and Mellish, 1981) predicates ready to be used as background facts for the ILP system CProgol4.4. An example of secondary structure prediction, made by RNAfold, for the 5′ UTR sequence and ten nucleotides of a coding sequence is shown in Figure 6.1.

[2] `ftp://ftp.ebi.ac.uk/pub/databases/UTR/data/5UTR.Fun_nr.dat.gz` version 16 June 2006

Transforming mRNA Secondary Structure into Background Knowledge

Perl[3] and Prolog scripts were used to transform RNAfold's outputs into Prolog predicates. Table 6.3 shows the predicates used to represent mRNA secondary structure as extensional background knowledge. In this work, we view the mRNA secondary structure from the highest level. This means that we do not consider a nested stem-loop as an independent stem-loop. For example, we only consider *YAP2* to have the three stem-loop structures shown in Figure 6.1 and Table 6.4.

Table 6.3: Background predicates representing mRNA secondary structure.

`stemloop(W,X,Y,Z)`	states that stem-loop `W` has its opening and closing positions in `X` and `Y` bases to the coding sequence; and there are, in total, `Z` base-pairing within `W`.
`has_stemloop(X,Y)`	represents relation between UTR `X` and stem-loop `Y`.

Table 6.4: Representation of a predicted structure shown in Figure 6.1.

```
has_stemloop(YDR423C, YDR423C_sl3).
stemloop(YDR423C_sl3, 98, 71, 10).
has_stemloop(YDR423C, YDR423C_sl2).
stemloop(YDR423C_sl2, 66, 17, 13).
has_stemloop(YDR423C, YDR423C_sl1).
stemloop(YDR423C_sl1, 13, -3, 3).
```

Baim and Sherman (1988) and Laso et al. (1993) suggested that the stability of a secondary structure and the distance of a secondary structure to the coding sequence influence the strength of a secondary structure to inhibit the

[3]Perl is a programming language. Its source codes and documentation can be found at `http://www.perl.org/`

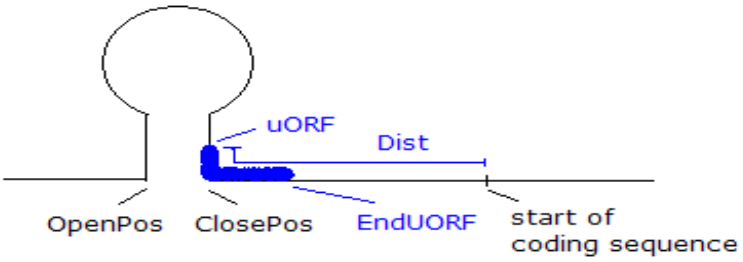

Figure 6.2: Illustration of a uORF intersects with an mRNA secondary structure on the uORF's left (upstream) part.

translation of the coding sequence. Therefore, the predicate `stemloop/4` was designed to capture both the distance (the opening and the closing positions in Figure 6.2) of a stem-loop structure to the coding sequence and the stability. Here, the stability was represented by the number of base pairing (the length of the stem); the longer the stem the more stable the secondary structure and the more energy is needed to unwind it. We do not use the predicted minimum free energy because of the way we view the mRNA secondary structure. For example, we consider three stem-loop structures while there was only one predicted minimum free energy for the overall structure shown in Figure 6.1.

With the biological knowledge gained from literature, we defined several declarative rules relating mRNA secondary structures to uORFs (Table 6.5). The rule `intersectleft_with_stemloop/1` identifies if a uORF intersects with any secondary structure on the uORF's left (upstream) part (see an illustration in Figure 6.2). The rule `intersectright_with_stemloop/1` identifies if a uORF intersects with any secondary structure on the uORF's right (downstream) part, and the rule `is_inside_stemloop/1` identifies if a uORF is inside any secondary structure.

To instruct CProgol to include mRNA secondary structure in its hypothesis space, we defined additional mode declarations (Table 6.6). The type `stemloop` was declared by defining a set of ground unit clauses of the predicate

Table 6.5: Background rules relating mRNA secondary structure to uORFs.

```
% The units of ‘‘Len’’ are codons and the units of ‘‘Dist’’ are nucleotides.
% Thus Len must be multiplied by 3 to convert its units to nucleotides.

% uORF intersects with a secondary structure on the uORF's left part
intersectleft_with_stemloop(UORF):-
    uORF(UORF,Dist,Len), belongs_to(UORF,UTR), has_stemloop(UTR,SL),
    stemloop(SL, OpenPos, ClosePos, NoOfPair),
    EndUORF is Dist-(Len*3), Dist =< OpenPos, ClosePos > EndUORF.

% uORF intersects with a secondary structure on the uORF's right part
intersectright_with_stemloop(UORF):-
    uORF(UORF,Dist,Len), belongs_to(UORF,UTR), has_stemloop(UTR,SL),
    stemloop(SL, OpenPos, ClosePos, NoOfPair),
    EndUORF is Dist-(Len*3), Dist > OpenPos, ClosePos =< EndUORF.

% uORF is within a secondary structure
is_inside_stemloop(UORF):-
    uORF(UORF,Dist,Len), belongs_to(UORF,UTR), has_stemloop(UTR,SL),
    stemloop(SL, OpenPos, ClosePos, NoOfPair),
    Dist =< OpenPos, EndUORF is Dist-(Len*3), ClosePos =< EndUORF.
```

Table 6.6: Additional mode declarations used in experiments with mRNA secondary structure.

```
:- modeb(1,is_inside_stemloop(+uORF))?
:- modeb(1,intersectleft_with_stemloop(+uORF))?
:- modeb(1,intersectright_with_stemloop(+uORF))?
:- modeb(*,has_stemloop(+uORF,-stemloop))?
:- modeb(1,stemloop(+stemloop,-pospair1,-pospair2,-numberofpairs))?
:- modeb(1,+numberofpairs=< #int)?
:- modeb(1,+numberofpairs>= #int)?
:- modeb(1,+numberofpairs= #int)?
:- modeb(1,+pospair1=< #int)?          :- modeb(1,+pospair2=< #int)?
:- modeb(1,+pospair1>= #int)?          :- modeb(1,+pospair2>= #int)?
:- modeb(1,+pospair1= #int)?           :- modeb(1,+pospair2= #int)?
```

`stemloop(X)`, where `X` is a stem-loop ID. The types of `pospair1`, `pospair2`, and `numberofpairs` were all defined as integer.

6.3 Results and Analysis

To statistically evaluate the impact of incorporating mRNA secondary structure as part of the background knowledge on the task of recognising yeast functional uORFs, we compared the relative advantage (RA) values (Muggleton et al., 2001, Appendix A) from 100 experiments with and without mRNA secondary structure. In 87 experiments out of 100, the mean RA values from the experiments with mRNA secondary structure are better than the mean RA values from the corresponding experiments without mRNA secondary structure (see Figure 6.3). The result from a *Wilcoxon Signed Rank test* shows that there was a statistically significant increase from the mean RA values from the experiments without mRNA secondary structure to those from the corresponding experiments with mRNA secondary structure (mean RA values: mean with-

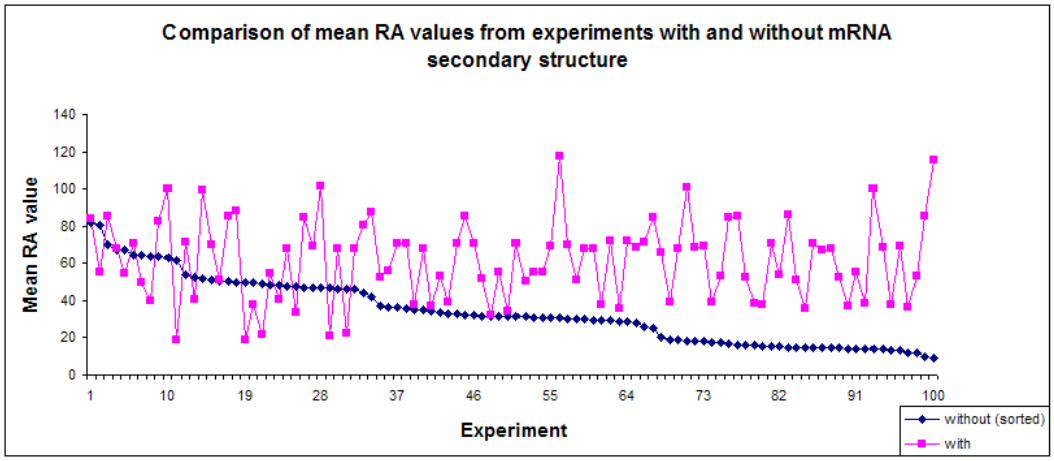

Figure 6.3: Comparison of mean RA values from 100 experiments with and without mRNA secondary structure. In 87 experiments, the mean RA values with mRNA secondary structure are better than without.

out=34.05, mean with=61.53, $Z = -7.159, p < 0.0005$). The summary of our experimental results regarding the mean RA values from the 100 experiments with and without mRNA secondary structure is shown in Table 6.7.

The range of mean RA values from 100 experiments with mRNA secondary structure is from 18.73 to 117.51, while from the experiments without mRNA secondary structure is from 9.38 to 81.64. The spread of sorted mean RA values in 100 experiments with and without mRNA secondary structure are shown in

Table 6.7: Summary of mean RA values from experiments with and without mRNA secondary structure.

Mean RA values	with	without
Minimum	18.73	9.38
Maximum	117.51	81.64
Average	61.53	34.05
Std. Deviation	21.45	17.53

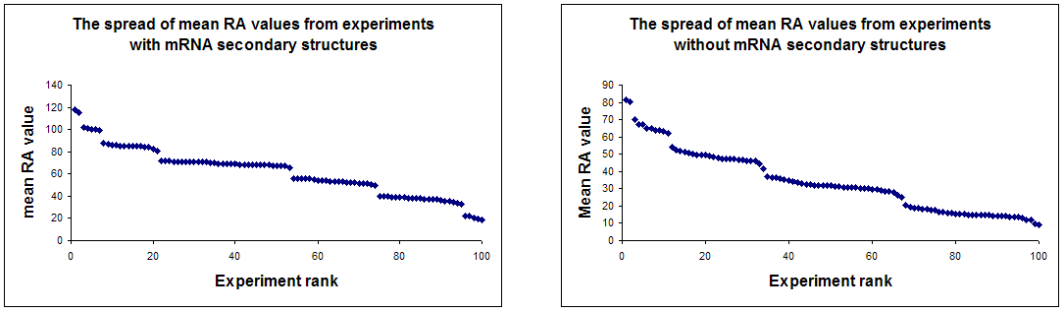

Figure 6.4: The spread of sorted mean RA values in 100 experiments with and without mRNA secondary structure. Note: different scale on Y axes.

Figure 6.4. The two graphs in Figure 6.4 show that for both experiments, with and without mRNA secondary structure, some variation in the mean RA values can be observed as a result of using random orderings of positive examples. The highest mean RA value for the experiments without mRNA secondary structure is almost 9 times of the minimum value in the same set of experiments. And the highest mean RA value for the experiments with mRNA secondary structure is more than 6 times of the minimum value in the same set of experiments. These results indicate that there is a dependency between performances of the hypotheses constructed by CProgol and the orderings of positive training examples.

The analysis made so far is based on the mean RA values from our experiments. However, RA is less well known than other performance measures such as precision, recall (also known as sensitivity), specificity, and F_1 score. Therefore, to support our analysis, we also measured the precision, recall, specificity, and F_1 score (Table 6.8)[4]. We found that there were statistically significant increase in the values of precision, recall, specificity, and F_1 score from the experiments without mRNA secondary structure to those from the corresponding experiments with mRNA secondary structure (precision: mean without=0.45,

[4]For this purpose, the random examples were considered as negatives.

Table 6.8: Summary of precision, recall, specificity, and F_1 score from the experiments with and without mRNA secondary structure.

Statistics	Precision		Recall		Specificity		F_1 score	
	with	without	with	without	with	without	with	without
Minimum	0.45	0.25	0.70	0.50	0.93	0.89	0.55	0.33
Maximum	0.78	0.65	0.95	0.95	0.98	0.97	0.81	0.71
Mean	0.63	0.45	0.87	0.77	0.96	0.94	0.70	0.54
Std. Deviation	0.07	0.08	0.05	0.09	0.01	0.02	0.05	0.07

mean with=0.63, $Z = -8.516$; recall: mean without=0.77, mean with=0.87, $Z = -7.427$; specificity: mean without=0.94, mean with=0.96, $Z = -8.278$; F_1 score: mean without=0.54, mean with=0.70, $Z = -8.553$; all were based on Wilcoxon Signed Ranks test with $p < 0.0005$).

Spearman's rank correlation was used to find out whether there are relationships between RA and the other measures (Table 6.9). We conclude that mean RA has a strong positive correlation with precision and specificity. This is as we expected. Reducing the number of randoms that are falsely classified as positives is very important in this domain, because verification via lab analysis is costly, and thus higher precision and higher specificity are desirable. Spearman's correlation also shows that there was a strong positive correlation between mean RA and F_1 score. This is due to the strong positive correlation between mean RA and precision, as there was no significant correlation between mean RA and recall; precision and recall are the two components used for calculating F_1 score. The plot of mean RA against other performance measures from the experiments with and without mRNA secondary structure are presented in Figure 6.5.

The hypotheses from 10 experiments that give 10 highest average cross-validation performance (mean RA) suggest that mRNA secondary structure influences uORF's ability to regulate translation in the yeast *S. cerevisiae*. Some

Table 6.9: Spearman's rank correlation between mean RA and other performance measures from 100 experiments with and without mRNA secondary structure.

Experiment		Precision	Recall	Specificity	F_1 score
with	Mean RA	0.94	-0.02	0.73	0.74
without	Mean RA	0.91	-0.05	0.72	0.70

Note: There is no significant correlation between mean RA and recall. All other correlations are significant with $p < 0.0005$.

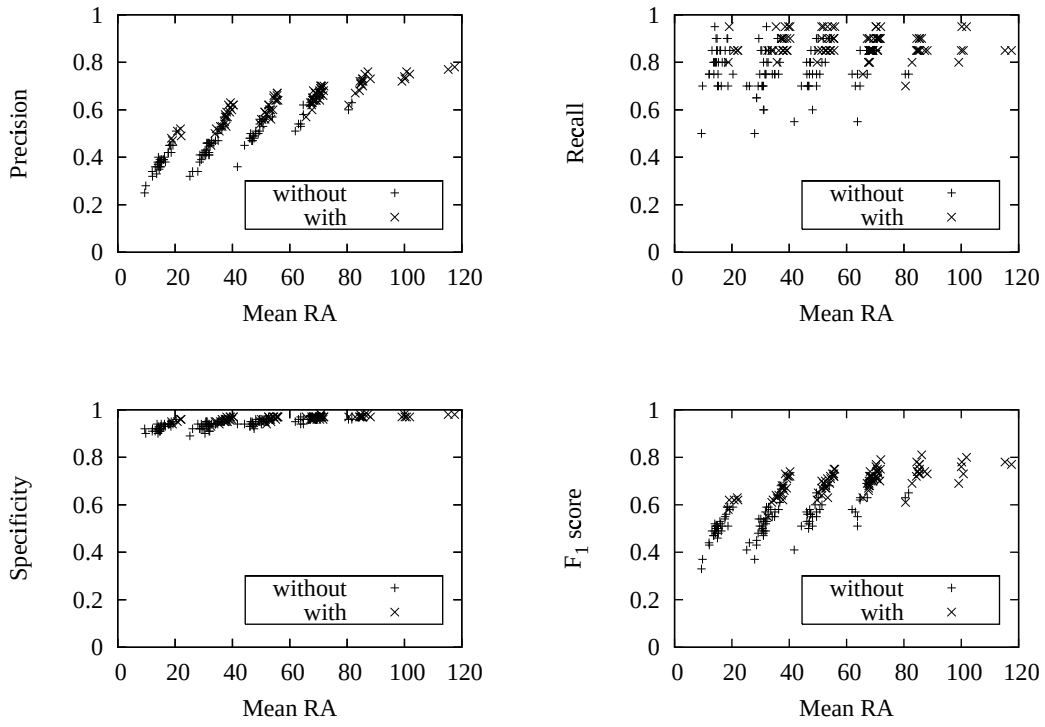

Figure 6.5: Plot of mean RA against precision, recall, specificity and F_1 score from 100 experiments with random order of positive training examples with and without mRNA secondary structure.

Table 6.10: Some typical hypotheses which give high mean RA values.

The rules that give the highest mean RA.

```
has_functional_role(A) :- is_inside_stemloop(A).
has_functional_role(A) :- intersectleft_with_stemloop(A).
has_functional_role(A) :- uORF(A,249,B).
```

The rules that give the 2^{nd} highest mean RA.

```
has_functional_role(A) :- is_inside_stemloop(A).
has_functional_role(A) :- has_shortest_dist_in_UTR(A).
has_functional_role(A) :- uORF(A,249,B).
```

The rules that give the 3^{rd} highest mean RA.

```
has_functional_role(A) :- uORF(A,B,C), is_inside_stemloop(A), B=<361.
has_functional_role(A) :- belongs_to(A,B), utr(B,C,589).
has_functional_role(A) :- uORF(A,B,C), B=<249, C=<10.
```

typical hypotheses which give high mean RA values are shown in Table 6.10. This table shows how simple our rules are. These rules also suggest that a functional uORF is likely to lie inside a stem-loop structure, or to intersect with a stem-loop structure on the uORF's left part. In our data, 17 of the 20 functional uORFs (positive examples) lie inside stem-loop structures predicted on the associated UTRs. For 3 of the 20 uORFs, their left part intersect with stem-loop structures predicted on the associated UTRs; 2 of these 3 uORFs do not lie inside stem-loop structures predicted on the associated UTRs.

6.4 Conclusions

This work is the first machine learning work to study uORFs and mRNA secondary structures together in the context of gene regulation. Although there have been many pieces of work that have studied either uORFs or mRNA

secondary structures in the context of gene regulation, we are only aware of two previous studies investigating both together; both of these studies involved wet-lab experiments.

In this chapter, we tested whether mRNA secondary structure could help in recognising known functional uORFs in the yeast *S. cerevisiae*. Our empirical results show that the performance of an ILP system, CProgol 4.4, in recognising known functional uORFs in the yeast *S. cerevisiae* significantly increases when mRNA secondary structure is added to the background knowledge (mean RA values: mean without=34.05, mean with=61.53, $Z = -7.159, p < 0.0005$). This conclusion still holds when performance is measured using precision, recall, specificity, and F_1 score, which are very well known in both machine learning and bioinformatics domains.

Spearman's correlation shows that mean RA has a strong positive correlation with precision, specificity and F_1 score. The correlation between mean RA and F_1 score was due to the correlation between mean RA and precision, as there was no significant correlation between mean RA and recall. Here, because of a much smaller number of positives compared to randoms, getting a higher recall is less important than getting a higher precision and specificity. Chapter 7 will show a situation where recall (also called sensitivity) is the most suitable performance measure.

From the hypotheses, it seems that mRNA secondary structure influences uORFs' ability to regulate translation. The hypotheses also suggest that a functional uORF is likely to lie inside a stem-loop structure, or intersect with a stem-loop structure on the uORF's left part.

This study also shows that there is a dependency between the performance of the hypotheses generated by CProgol and the ordering of positive training examples. While it may be obvious from the description of the covering approach, to the best of our knowledge there was no published work investigating the dependency between performances of the hypotheses constructed by CPro-

gol4.4, which uses the covering approach, and the orderings of positive training examples.

After learning from our experiments in this chapter and the previous one, we will present a new and finer approach of learning functional uORFs in the next chapter.

Chapter 7

Learning Functional uORFs: A Finer Approach

The experiments described in Chapter 5 have shown that ILP has the potential to help in selecting sets of candidate functional uORFs for wet-experimental studies. In Chapter 6, it was shown that the background knowledge used in Chapter 5 was very limited; by adding more relevant background knowledge, ILP's performance in recognising known functional uORFs can be improved.

In this chapter, a new approach to learning functional uORFs is researched. There are three main differences between this approach and the previous ones. First, we employ a different ILP system, Aleph (Srinivasan, 1999). Why we do this is explained in Section 7.1. Second, instead of intergenic sequences, we use 5′ UTR sequences. Third, in addition to the knowledge from *S. cerevisiae*'s sequences, knowledge derived from sequences of other yeast species, an analysis of expression data sets, and gene ontology annotations are also used to form the background knowledge for ILP[1]. Why we think these heterogeneous data could be useful for this study and how we transform them into ILP format are

[1]While we believe mRNA secondary structure can be relevant background knowledge, we do not include this as background knowledge here; this is mainly because the analysis of the work in Chapter 6 was not finished when we started the study described in this chapter.

discussed in Sections 7.2 - 7.5. We describe how we evaluate our new approach in Section 7.6 and use this new approach to predict novel functional uORFs in Section 7.7. We discuss our analysis in Section 7.8 and conclude in Section 7.9.

7.1 Removing Dependency on the Ordering of Positive Examples

The work described in the previous two chapters uses CProgol version 4.4. As explained in Chapter 2 and shown in Chapter 6, the hypotheses constructed by CProgol may depend on the ordering of the input positive examples. Here, rather than trying to find the best or near optimal ordering which can give the best or near optimal performance, we use a different ILP system, Aleph (Srinivasan, 1999). The latter system provides a way of inducing hypotheses without dependency on the ordering of positive examples, which is done through `induce_max` command.

Unlike CProgol which each time only saturates the first positive example, Aleph's `induce_max` saturates every example and hence does not depend on the ordering of input positive examples. In recent work by Specia, Srinivasan, Ramakrishnan and das Graças Volpe Nunes (2007), `induce_max` was shown to give better performance when compared to Aleph's basic algorithm for constructing hypotheses, through `induce` command, for the task of identifying the correct sense of words.

7.2 Deriving Knowledge from 5′ UTR Sequences

When the work in Chapter 5 was done, the length of 5′ UTR was only known in a small number of genes, and therefore the intergenic sequences of *S. cerevisiae* were used as our main source of knowledge. In between that study and this study, we noticed that Zhang and Dietrich (2005b), David, Huber, Granovskala,

Toedling, Palm, Bofkin, Jones, Davis and Steinmetz (2006) and Miura et al. (2006) published their work on genome-scale mapping of the transcription start sites (the start of 5′ UTRs) of the yeast *S. cerevisiae*. Both the technologies they used and the results they produced vary. Zhang and Dietrich (2005b) used serial analysis of gene expression (SAGE), David et al. (2006) used tiling array, while Miura et al. (2006) used complementary DNA (cDNA) clones. As it is difficult to assess which mapping is better in quality, we made an arbitrary decision to use the results of David et al.'s (2006) experiments.

From the supplementary material (Table 3) of David et al.'s (2006) paper, we extracted coordinates of the start of 5′ UTRs[2]. From the ENSEMBL database at the Biomart Central Server[3], we downloaded coordinates of the protein coding genes and 1000 bases upstream sequences of the protein coding genes of *S. cerevisiae*. 1000 bases was chosen because 5′ UTR lengths in *S. cerevisiae* are mainly distributed below 500 bases, with a small percentage between 500 and 1000 bases, and only very rarely are they above 1000 bases (David et al., 2006; Miura et al., 2006). Having the coordinates of the 5′ UTRs and the coordinates of the protein coding genes, we calculated the lengths of 5′ UTRs, which were then used to extract 5′ UTR sequences from the upstream sequences. This gives a total of 4,938 5′ UTR sequences. By using *getorf* of the EMBOSS package (Rice, Longden and Bleasby, 2000), uORFs with minimum length 3 codons (including start codon and stop codon) were extracted from these 4,938 5′ UTR sequences. The result is 3,647 (21+2+3,624) uORFs from 1,493 *S. cerevisiae*'s protein coding genes (Tables 7.1 and 7.2).

[2]The first base coordinate of the first probe of each segment in that table was used as the coordinate of the 5′ UTR. If there are more than one segment associated to a same gene, the segment which gives the longest length was chosen. For a few of the 18 studied genes, we use the lengths from EMBL database (`ftp://ftp.ebi.ac.uk/pub/databases/UTR/data/5UTR.Fun_nr.dat.gz` version 16 June 2006) because those calculated from David et al.'s (2006) experiments are far too short.

[3]`http://www.biomart.org/biomart/martview/`, accessed 15 March 2007.

Table 7.1: Detailed composition of uORFs obtained using *getorf* of the EMBOSS package.

Number of Genes		uORFs		
		Functional	Non-functional	Unlabelled
18 studied genes	9[a]	21	-	-
	2	-	2	-
	7	-	-	8
1,475 other genes		-	-	3,616
1,493 genes		21	2	3,624

[a]This cell is 9 rather than 8 because *CPA1* has been included. *CPA1* was not part of our data in Chapter 5, as its intergenic's length was not listed in Philippakis et al. (2005).

Aleph is instructed to learn a predicate `has_functional_role/1` from a set of positive examples and a set of background facts and rules. Positive examples were represented as ground unit clauses of the predicate `has_functional_role(X)`, where `X` is a uORF ID. A uORF ID is a composite of the systematic name of the gene to which the uORF is associated with (e.g., those listed in second column of Table 7.2) and a uORF identifier (e.g., uORF1, uORF2, etc.).

In addition to the uORF's properties used in Chapter 5 (i.e., distance of the uORF to the start of the coding sequence in bases, frequency of AU and GC base pairs immediately upstream and downstream of each uORF, and the length of the uORF in codons), we also extracted the bases in positions -3 and +4. In Kozak's experiments (Kozak, 1981; Kozak, 2005) with mammalian sequences, these positions were found to give an optimum context for an AUG to be recognised by ribosome (see Section 4.2.1 on page 54 for details). In Chapter 5, the UTR properties were actually representing the properties of intergenic regions. Here, the UTR properties represent the properties of the

Table 7.2: Detailed uORF composition from 18 studied genes within the collection obtained using *getorf* of the EMBOSS package.

Gene Name[a]	Systematic Name[a]	Positive Examples	Negative Examples	Unlabelled Examples
CLN3	YAL040C	1	-	-
GCN4	YEL009C	4	-	-
HAP4	YKL109W	2	-	-
TIF4631	YGR162W	5	-	-
YAP1	YML007W	1	-	-
YAP2	YDR423C	2	-	-
HOL1	YNR055C	1	-	-
PET111	YMR257C	4	-	-
CPA1	YOR303W	1	-	-
SCO1	YBR037C	-	1	-
CBS1	YDL069C	-	1	-
INO2	YDR123C	-	-	1
PPR1	YLR014C	-	-	1
URA1	YKL216W	-	-	1
LEU4	YNL104C	-	-	1
RCK1	YGL158W	-	-	2
DCD1	YHR144C	-	-	1
SCH9	YHR205W	-	-	1
18 Genes		21	2	8

[a]Names are taken from SGD (`http://www.yeastgenome.org`).

Table 7.3: Background predicates representing knowledge derived from 5′UTR sequences.[a]

`uORF(X,Y,Z)`	represents uORF `X` which starts `Y` nucleotides upstream of the coding sequence, with length `Z` codons.
`utr(X,Y,Z)`	represents UTR `X` which has `Y` uORF(s) within `Z` nucleotides upstream of gene `X` (`Z` is the length of 5′ UTR region).
`has_uORF(X,Y)`	represents relation between UTR `X` and uORF `Y`.
`belongs_to(X,Y)`	represents relation between uORF `X` and UTR `Y`.
`context(W,X,Y,Z)`	states that within the 20 nucleotides downstream of `W`'s stop codon, the frequency of AU is `X` and the frequency of GC is `Y`; and the base in position +4 is `Z`.
`up_context(W,X,Y,Z)`	states that within the 20 nucleotides upstream of `W`'s start codon, the frequency of AU is `X` and frequency of GC is `Y`; and the base in position -3 is `Z`.

[a] A set of ground unit clauses was generated for each predicate.

5′ UTRs, which include the number of uORFs found in the region, and the length of the region. All of this information was represented as extensional background knowledge for Aleph (Table 7.3).

The rules in Chapter 5 that examine the abundance of AU and GC base pairs immediately upstream and downstream of each uORF are still relevant and thus are kept. However, the rules `has_shortest_dist_in_UTR/1` and `has_shortest_len_in_UTR/1` are removed. The reason for this is that they are now not necessary, as we add rules regarding conservation (see Section 7.3).

Several new rules were also defined (Table 7.4). The rule `has_G_in_Plus4/1` examines whether the base in position +4 relative to the uORF's start codon is G, while the rule `has_A_or_G_in_Min3/1` examines whether the base A or G can be found in position -3 relative to the uORF's start codon. The rules `lteq/2` and `gteq/2` check binary comparison less than or equal, and greater than or equal.

Table 7.4: Background rules regarding knowledge derived from 5′UTR sequences.

```
lteq(X,Y):-
    not(var(X)), not(var(Y)),
    number(X), number(Y), X =< Y, !.
lteq(X,X):-
    not(var(X)), number(X).

gteq(X,Y):-
    not(var(X)), not(var(Y)),
    number(X), number(Y), X >= Y, !.
gteq(X,X):-
    not(var(X)), number(X).

gcrich_down_up(UORF):-
    context(UORF,Au,Gc,_), Gc > Au,
    up_context(UORF,A,G,_), G > A.
gcrich_down_aurich_up(UORF):-
    context(UORF,Au,Gc,_),  Gc > Au,
    up_context(UORF,A,G,_), G < A.

aurich_down_up(UORF):-
    context(UORF,Au,Gc,_), Gc < Au,
    up_context(UORF,A,G,_), G < A.
gcrich_up_aurich_down(UORF):-
    context(UORF,Au,Gc,_), Gc < Au,
    up_context(UORF,A,G,_), G > A.

has_A_or_G_in_Min3(UORF):-
    up_context(UORF,_,_,'A'), !.
has_A_or_G_in_Min3(UORF):-
    up_context(UORF,_,_,'G').

has_G_in_Plus4(UORF):-
    context(UORF,_,_,'G').
```

7.3 Deriving Knowledge from Other Yeast Species

The availability of many types of genome-scale data for different eukaryotes have made possible genome-scale comparisons. These comparisons revealed that there are a large number of genes which take part in core biological process(es) and/or carry out core biological function(s) shared by all eukaryotes (Ashburner, Ball, Blake, Botstein, Butler, Cherry, Davis, Dolinski, Dwight, Eppig, Harris, Hill, Issel-Tarver, Kasarskis, Lewis, Matese, Richardson, Ringwald, Rubin and Sherlock, 2000). This finding suggests that functional genes are likely to be conserved in at least several closely related species. This insight may also applied to functional uORFs, that is to say, uORFs which are functional are likely to be conserved in closely related species. Therefore, information about uORFs in closely related species to *S. cerevisiae* could be beneficial for this study.

5′ UTR sequences of several other species in the *genus* (family) *Saccharomyces* (see Figure 7.1) would be ideal sources to get the information about uORFs in closely related species to *S. cerevisiae*, but these are not available. However, upstream sequences of six other *Saccharomyces* species i.e., *S. bayanus*, *S. castellii*, *S. kluyveri*, *S. kudriavzevii*, *S. mikatae* and *S. paradoxus* are available. Two of these six (*S. castellii* and *S. kluyveri*) are considered quite far from *S. cerevisiae* (Cliften, Hillier, Fulton, Graves, Miner, Gish, Waterston and Johnston, 2001), and thus would have lesser degree of conservation to *S. cerevisiae*. Of the four closer to *S. cerevisiae*, three (*S. paradoxus*, *S. mikatae* and *S. bayanus*) were studied recently by Kellis, Patterson, Endrizzi, Birren and Lander (2003). They found that the order of genes among these three genomes and *S. cerevisiae* are well conserved. In their analysis, they also suggested that the three genomes and *S. cerevisiae* have diverged enough to allow functional elements to be recognised. Therefore, we chose to use these three species.

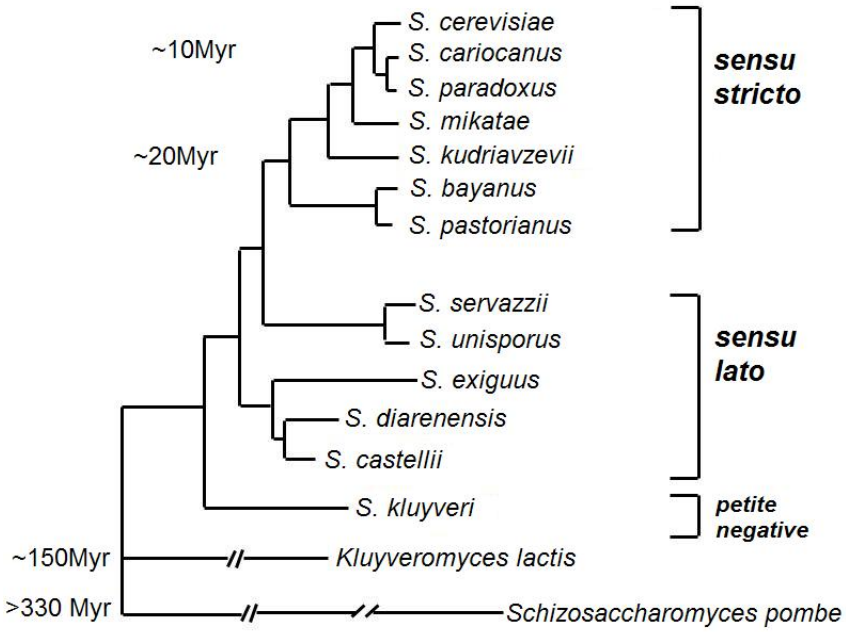

Figure 7.1: *Saccharomyces* phylogeny. The figure shows a rough estimation, in million years (Myr), of when evolutionary separation of species took place. (Source: Adapted, with permission, from supplementary materials of Cliften et al. (2003) shown at `http://www.genetics.wustl.edu/saccharomycesgenomes/yeast_phylogeny.html`).

We downloaded upstream sequences of *S. cerevisiae*'s ortholog genes in *S. paradoxus*, *S. mikatae* and *S. bayanus* from Saccharomyces Genome Database (SGD)[4] (Hong, Balakrishnan, Christie, Costanzo, Dwight, Engel, Fisk, Hirschman, Livstone, Nash, Oughtred, Park, Skrzypek, Starr, Andrada, Binkley, Dong, Hitz, Miyasato, Schroeder, Weng, Wong, Zhu, Dolinski, Botstein and Cherry, 2007). We used upstream sequences of length 500 bases. 500 was chosen be-

[4] `ftp://genome-ftp.stanford.edu/pub/yeast/sequence/fungal_genomes/S_bayanus/MIT/orf_dna/utr5_500.fasta.gz` was downloaded on 2 May 2007. `ftp://genome-ftp.stanford.edu/pub/yeast/sequence/fungal_genomes/S_paradoxus/MIT/orf_dna/utr5_500.fasta.gz` was downloaded on 3 May 2007. And `ftp://genome-ftp.stanford.edu/pub/yeast/sequence/fungal_genomes/S_mikatae/MIT/orf_dna/utr5_500.fasta.gz` was downloaded on 3 May 2007

cause the degree of conservation among yeast species could be expected much lower beyond this region. This expectation was based on Mahony, Corcoran, Feingold and Benos's (2007) studies on mammalian species; they found that the degree of conservation of the upstream sequences of protein coding genes decreased beyond 500 bases. Furthermore, according to Cliften et al. (2003), the average of intergenic regions in the *Saccharomyces* family is around 500 bases.

Getorf of the EMBOSS package (Rice et al., 2000) was then used to find uORFs with minimum length 3 codons (including start codon and stop codon) in the upstream sequences. The results were then transformed into a set of ground facts of the following Prolog predicates:

```
spar_uORF(X,Y,Z).
```

to represent uORF `X` of *S. paradoxus* which starts `Y` nucleotides upstream of the ortholog coding sequence, with length `Z` codons,

```
spar_utr(X,Y,Z).
```

to represent UTR `X` of *S. paradoxus* which has `Y` uORF(s) within `Z` nucleotides upstream of the ortholog coding sequence `X`,

```
spar_belongs_to(X,Y).
spar_has_uORF(Y,X).
```

to represent relationships between uORF `X` and UTR `Y` of *S. paradoxus*. The predicates used to represent uORFs information in *S. mikatae* and *S. bayanus* are similar to those for *S. paradoxus*, with `spar` is changed into `smik` or `sbay` respectively.

In contemporary bioinformatics, conservation testing is usually done using sequence alignment, where sequence refers to the sequence of RNA bases in the case of mRNA. In this work, we define several Prolog rules for checking whether a uORF is conserved in 0, 1, 2, or 3 other species (Tables 7.5 and 7.6).

Table 7.5: Prolog rules for checking conservation of a uORF (part 1).

```
conserved_in_x_species(UORF,3):-
    conserved_in_spar(UORF), conserved_in_smik(UORF),
    conserved_in_sbay(UORF), !.
conserved_in_x_species(UORF,2):-
    conserved_in_spar(UORF), conserved_in_smik(UORF), !.
conserved_in_x_species(UORF,2):-
    conserved_in_spar(UORF), conserved_in_sbay(UORF), !.
conserved_in_x_species(UORF,2):-
    conserved_in_smik(UORF), conserved_in_sbay(UORF), !.
conserved_in_x_species(UORF,1):-
    conserved_in_spar(UORF), !.
conserved_in_x_species(UORF,1):-
    conserved_in_sbay(UORF), !.
conserved_in_x_species(UORF,1):-
    conserved_in_smik(UORF), !.
conserved_in_x_species(UORF,0).

% Ascii codes: 95=_ ; 117=u ; 79=O ; 82=R ; 70=F .
conserved_in_spar(UORF):-
    uORF(UORF,_,Len1), name(UORF,AsciiuORF),
    get_index(AsciiuORF,Index1), belongs_to(UORF,UTR),
    utr(UTR,Total1,_), spar_utr(UTR,Total2,_),
    Index2 is (Total2-Total1)+Index1,
    name(Index2,AsciiIndex2), name(UTR,AsciiUTR),
    append(AsciiUTR,[95,117,79,82,70],A),
    append(A,AsciiIndex2,AsciiUORF2),
    name(UORF2,AsciiUORF2), spar_uORF(UORF2,_,Len2),
    diff(Len1,Len2,X), X =< 3.
```

...continues to Table 7.6. `conserved_in_smik/1` and `conserved_in_sbay/1` are defined similarly to `conserved_in_spar/1`.

Table 7.6: Prolog rules for checking conservation of a uORF (part 2).

```
get_index(AsciiuORF,Index):-
    get_after_underscore(AsciiuORF,AsciiEnd),
    remove_uORF_word(AsciiEnd,AsciiIndex),
    name(Index,AsciiIndex).

get_after_underscore([95|T],T).
get_after_underscore([_|T],T1):-
    get_after_underscore(T,T1).

remove_uORF_word([70|T],T).
remove_uORF_word([_|T],T1):-
    remove_uORF_word(T,T1).

append([],L,L).
append([H|L1],L2,[H|L3]):-
    append(L1,L2,L3).

diff(Len1,Len2,X):-
    Len1 > Len2,
    X is Len1 - Len2, !.
diff(Len1,Len2,X):-
    Len1 < Len2,
    X is Len2 - Len1, !.
diff(_,_,0).
```

The definition of conserved used in this work is when uORFs from different species share similar length (i.e., if the difference is not more than 3 codons) and same position in the sequence of uORFs relative to the coding sequence. Thus, the closest uORF to a *S. cerevisiae*'s gene would be compared with the closest uORF to that ortholog gene in other species, and the second closest uORF to a *S. cerevisiae*'s gene would be compared with the second closest uORF to the ortholog gene in other species (Figure 7.2).

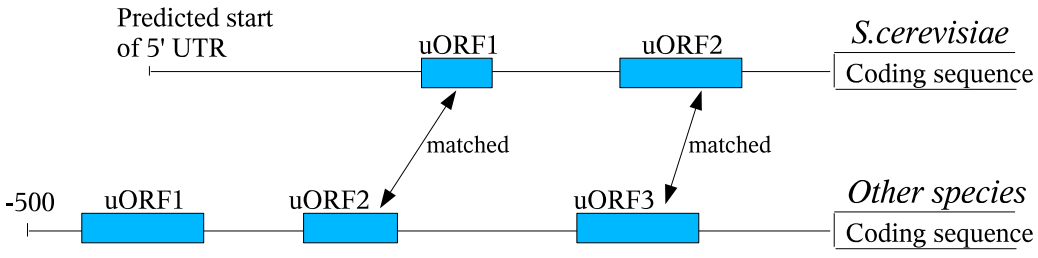

Figure 7.2: An illustration of conservation checking.

7.4 Deriving Knowledge from Gene Ontology Annotations

The analysis of Ashburner et al. (2000), that stated a large number of "important" genes are shared by eukaryotes, provides more support for scientists to make inferences about genes under investigation by applying knowledge about similar genes from other organisms. To make transferring knowledge from one organism to another more effective, there has been a community effort to create common vocabularies, such as Gene Ontology (GO) (Ashburner et al., 2000), for describing gene and gene product attributes in any organisms.

GO uses a hierarchical concept. That means that specific terms are considered as children of broader terms. The relationships between terms in GO are many-to-many, and thus a term may have many children and many parents. The GO was especially built to accommodate the need to have common annotations (vocabularies) for describing three main aspects of gene products: molecular function, biological process, and cellular component. These aspects represent the three GO categories.

The idea of using GO here is to allow ILP to examine uORFs associated with genes which share the same or related annotations(s). The basis for this is that one may wonder whether uORFs tend to be functional in the UTRs of genes whose products are involved in a specific function(s), or in a specific process(es), or expressed in a specific cellular component(s).

GO contains a large number of terms. As of 12 April 2007, there are a total of 22,968 terms. 13,464 of these terms are associated with biological process, 7,657 terms are associated with molecular function and the rest of 1,937 are associated with cellular component. The terms in each of the GO categories can be arranged in a tree structure according to GO hierarchy, i.e., process.ontology,

Table 7.7: Number of nodes in the first five levels of GO trees.

Level	GO category		
	Molecular Function	Biological Process	Cellular Component
1	1	1	1
2	20	20	17
3	730	677	279
4	737	1892	902
5	1531	6149	2125

function.ontology and component.ontology[5]. The tree for category molecular function consists of 15 levels, for biological process 18 levels and for cellular component 16 levels. The number of nodes in the first five levels in each tree are shown in Table 7.7.

In this work, we use the GO annotations for yeast genes[6] provided by SGD (Dwight, Harris, Dolinski, Ball, Binkley, Christie, Fisk, Issel-Tarver, Schroeder, Sherlock, Sethuraman, Weng, Botstein and Cherry, 2002). Although not all of the GO terms are used for annotating yeast genes, the GO annotations for yeast can be very specific. There are some annotations which only cover one gene. For the purpose of our study, terms as specific as this are not useful. We want more general annotations for yeast genes, so that each used annotation covers more genes. GO slim[7] provides such mapping. However, this mapping is too general for our study. Therefore, we mapped the GO annotations for yeast into their third level terms.

The results were represented as background facts of the following predicates:

[5]The three ontologies, version 26:03:2007, were downloaded from `http://www.geneontology.org/GO.downloads.ontology.shtml`

[6]gene_association.sgd.gz, version 24 March 2007, downloaded from `ftp://genome-ftp.stanford.edu/pub/yeast/data_download/literature_curation/`

[7]`http://www.yeastgenome.org/help/goslimhelp.html`, accessed 11 April 2007.

```
function(Gene,GOID).
process(Gene,GOID).
component(Gene,GOID).
```

to represent gene product of `Gene` is involved in a molecular function coded by `GOID`, gene product of `Gene` is involved in a biological process coded by `GOID`, and `Gene` is expressed in a cellular component coded by `GOID`, respectively. An example of `GOID` is 'GO:0050789', which codes for GO term 'regulation of biological process'.

To relate these background facts with uORFs and UTRs, several background rules were defined (Table 7.8). `mainORF_is_in/2` relates a uORF with the component-annotations of the main gene associated to that uORF. `mainORF_involved_in_process/2` and `mainORF_involved_in_function/2` relate a uORF with the process-annotations, and function-annotations, respectively, of the main gene associated to that uORF.

7.5 Deriving Knowledge from Expression Data

In Section 3.2, we mentioned that array technologies and serial analysis of gene expression (SAGE) can provide an effective way for elucidating the function of genes. Then, in Section 7.2, we briefly mentioned that these technologies have recently been used for mapping the transcription start sites of *S. cerevisiae*. In this work, we use an analysis of microarray experiments to investigate whether functional uORFs could be explained in terms of how genes respond to different stress conditions.

Microarray data can be stored as a gene expression matrix where each row represents a gene and each column represents a condition, and the value of each position in the matrix represents the expression of a certain gene in a certain condition. Such data allows us not only to study the expression of individual genes under different conditions in a genome-wide scale, but also allows us to

Table 7.8: Background rules regarding yeast association to GO.

```
% The main gene associated to that uORF is localised or expressed in...
mainORF_is_in(UORF,Comp):-
    uORF(UORF,_,_),
    belongs_to(UORF,UTR),
    component(UTR,Comp).
% The product of the main gene associated to that uORF is involved in
% process...
mainORF_involved_in_process(UORF,Process):-
    uORF(UORF,_,_),
    belongs_to(UORF,UTR),
    process(UTR,Process).
% The product of the main gene associated to that uORF is involved in
% function...
mainORF_involved_in_function(UORF,Function):-
    uORF(UORF,_,_),
    belongs_to(UORF,UTR),
    function(UTR,Function).
```

group genes which respond similarly to a set of conditions.

Here, derived knowledge from an analysis of four publicly available microarray data sets measuring translational activity under four different stress conditions (i.e., rapamycin stress, oxidative (H_2O_2) stress, butanol stress and amino acid starvation) are used as part of background knowledge. The four data sets were used previously for studying stress impact on translation. The rapamycin data set was used in Preiss, Baron-Benhamou, Ansorge and Hentze (2003), the oxidative data set was used in Shenton, Smirnova, Selley, Carroll, Hubbard, Pavitt, Ashe and Grant (2006), butanol and amino acid data sets were used in Smirnova, Selley, Sanchez-Cabo, Carroll, Eddy, McCarthy, Hubbard, Pavitt, Grant and Ashe (2005). Adding this knowledge to the hypotheses space will allow the ILP system to consider hypotheses that explain functional uORFs in terms of how their main genes respond to different stress conditions.

The *polysome-to-monosome*[8] *log-fold change* (the log value of the difference of polysome-to-monosome ratio) between stressed and normal condition was used to determine whether the expression of a gene is increased (*up-regulated*), decreased (*down-regulated*), or similar (not-regulated) under each stress. This analysis was done by Erik Kristiansson, Alexandra Jauhiainen and Janeli Sarv from the Department of Mathematical Statistics, Chalmers University of Technology, Sweden.

Whether a gene is up-regulated, down-regulated or not-regulated under each stress was represented in predicate logic as ground facts of the predicate:

```
regulated(X,Y,Z).
```

which states that under stress `Y` (1 for amino acid starvation, 2 for butanol, 3 for low concentration of H_2O_2, and 5 for rapamycin), gene product of `X` is either up-regulated (if `Z` is 1), down-regulated (if `Z` is -1), or not-regulated (if `Z` is 0).

[8]Polysome (short form of polyribosome) is used to describe a group of many ribosomes attached to an mRNA, while monosome is used to describe a single ribosome attached to an mRNA.

To relate a uORF with information on whether the main gene associated to that uORF is regulated (up and/or down) or not regulated under certain stress, we define rules `regulated_under/2` and `not_regulated_under/2` (Table 7.9).

Table 7.9: Background rules regarding expression data.

```
% The main gene associated to this uORF is regulated under...
regulated_under(UORF,Cond):-
    uORF(UORF,_,_),
    belongs_to(UORF,UTR),
    regulated(UTR,Cond,Val),
    Val \= 0.
% The main gene associated to this uORF is not regulated under...
not_regulated_under(UORF,Cond):-
    uORF(UORF,_,_),
    belongs_to(UORF,UTR),
    regulated(UTR,Cond,0).
```

7.6 Leave-one-out Cross-validation

In this experiment, we investigate whether our new approach could generate hypotheses with good performance. For this, Aleph's parameters were set as follows: **evalfn** is set to 'posonly' so that Aleph learns from positive examples only; **i** (the maximum depth of new variables) is set to 5; **clauselength** (the maximum number of atoms in an acceptable clause) is set to 15; **nodes** (the maximum number of nodes explored during clause searching) is set to 100,000; and **depth** (the maximum depth of the stack used when proving before backtracking) is set to 1,000. More details on Aleph's parameters can be found in Section 2.4.2 on page 23.

To tell Aleph which predicates can be used for constructing hypotheses about the target predicate `has_functional_role/1` and in what mode the

predicates should be called, we declare determination and mode statements (see Tables 7.10 and 7.11).

Table 7.10: Mode and determination statements regarding uORFs and UTRs properties.

```
:- modeh(1,has_functional_role(+uORF)).

:- modeb(1,uORF(+uORF,-distancefromstart,-uORFlength)).
:- modeb(1,belongs_to(+uORF,-utr)).
:- modeb(1,utr(+utr,-numberofuORF,-utrlength)).
:- modeb(1,lteq(+distancefromstart,#int)).  :- modeb(1,lteq(+uORFlength,#int)).
:- modeb(1,+distancefromstart= #int).       :- modeb(1,+uORFlength= #int).
:- modeb(1,gteq(+distancefromstart,#int)).  :- modeb(1,gteq(+uORFlength,#int)).
:- modeb(1,lteq(+numberofuORF,#int)).       :- modeb(1,lteq(+utrlength,#int)).
:- modeb(1,+numberofuORF= #int).            :- modeb(1,+utrlength= #int).
:- modeb(1,gteq(+numberofuORF,#int)).       :- modeb(1,gteq(+utrlength,#int)).
:- modeb(1,has_G_in_Plus4(+uORF)).    :- modeb(1,has_A_or_G_in_Min3(+uORF)).
:- modeb(1,gcrich_down_up(+uORF)).    :- modeb(1,gcrich_down_aurich_up(+uORF)).
:- modeb(1,aurich_down_up(+uORF)).    :- modeb(1,gcrich_up_aurich_down(+uORF)).

:- determination(has_functional_role/1,uORF/3).
:- determination(has_functional_role/1,lteq/2).
:- determination(has_functional_role/1,'='/2).
:- determination(has_functional_role/1,gteq/2).
:- determination(has_functional_role/1,gcrich_down_up/1).
:- determination(has_functional_role/1,aurich_down_up/1).
:- determination(has_functional_role/1,gcrich_down_aurich_up/1).
:- determination(has_functional_role/1,gcrich_up_aurich_down/1).
:- determination(has_functional_role/1,has_G_in_Plus4/1).
:- determination(has_functional_role/1,has_A_or_G_in_Min3/1).
:- determination(has_functional_role/1,belongs_to/2).
:- determination(has_functional_role/1,utr/3).
```

Table 7.11: Mode and determination statements regarding conservation, yeast association to GO, and expression data.

```
:- modeb(1,conserved_in_x_species(+uORF,#int)).
% Recall number * is used as each gene may be associated with many GO IDs
:- modeb(*,mainORF_is_in(+uORF,#goid)).
:- modeb(*,mainORF_involved_in_function(+uORF,#goid)).
:- modeb(*,mainORF_involved_in_process(+uORF,#goid)).
% Recall number 4 is used as there are only 4 different stress conditions
:- modeb(4,regulated_under(+uORF,#stressid)).
:- modeb(4,not_regulated_under(+uORF,#stressid)).

:- determination(has_functional_role/1,conserved_in_x_species/2).
:- determination(has_functional_role/1,mainORF_is_in/2).
:- determination(has_functional_role/1,mainORF_involved_in_function/2).
:- determination(has_functional_role/1,mainORF_involved_in_process/2).
:- determination(has_functional_role/1,regulated_under/2).
:- determination(has_functional_role/1,not_regulated_under/2).
```

The types `uORF`, `utr`, and `goid` were declared by defining a set of ground unit clauses of the predicate `uORF(X)`, where `X` is a uORF ID; `utr(X)`, where `X` is a UTR ID; and `goid(X)`, where `X` is a GO ID. The types of `distancefromstart`, `uORFlength`, `numberofuORF`, `utrlength`, and `stressid` were all defined as integer.

Unlike the standard positive-only setting of CProgol which allows users to give their own random examples, the standard positive-only setting of Aleph will automatically generate random examples, and therefore, users are only expected to give positive examples. Using automatically generated randoms reduces the kinds of information that users have to provide and is suitable for domains where random examples are not available or are scarce. However, automatically generated random examples are less informative than the users'

own random examples and they may not represent true examples; this may affect the learning process.

Here, due to the change from using intergenic sequences to using 5′ UTR sequences, randoms are no longer abundant. Thus, the standard positive-only setting of Aleph is suitable. Due to using this setting, only the positive examples of the 18 studied genes are used for training and testing in this experiment. Since there are only 21 positive examples in our data set, we evaluate our method using leave-one-out cross-validation (see Figure 7.3). This means that each example in turn is used as a test set, while the other 20 examples are used as a training set. Thus in total, we do 21 executions to evaluate our method.

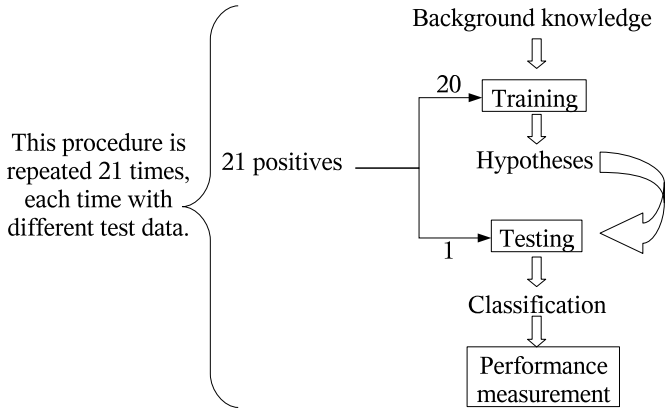

Figure 7.3: Leave-one-out cross-validation. The final estimate of performance is an average over 21 measurements.

Since we only use positive examples for training and testing, the performance of the hypotheses is measured using sensitivity (also known as recall), which measures the fraction of positives which are recognised by the hypotheses as positives. In 17 experiments, from a total of 21, the hypotheses can correctly recognise the test set. Thus, the estimate sensitivity of our method is 81%.

7.7 Predicting Novel Functional uORFs

Having achieved a reasonably high sensitivity, we were encouraged to do another experiment. The same background knowledge, parameter settings, type and mode declarations, as well as determination statements were used again but this time all of the 21 positive examples were used to generate a set of hypotheses. The resulting hypotheses are shown in Table 7.12. We then use this set of hypotheses to classify the negative (2) and unlabelled (8) examples within the 18 studied genes (see Table 7.1 on page 103). Only 2 of 10 examples were classified as positives by the hypotheses; one from the negative set and one from the unlabelled set. Thus, we believe that the high sensitivity is not because the hypotheses tend to classify any example as positive. Rather, we think that this is because the hypotheses are quite specific.

When the same set of hypotheses (Table 7.12) was used to classify 3,616 unlabelled examples from 1,475 genes (see Table 7.1 on page 103), they predict 450 uORFs from 299 genes as functional. Clearly, extensive lab work would be required to verify whether these 450 uORFs from 299 genes are indeed functional. When compared to the results described in Chapter 5, the size of this set is much smaller and thus it is more possible to use this set for lab experimental studies. Due to some constraints on funding, facilities, time and expertise, the author could not do these biological experiments during her doctoral project. However, some promising indications and strong supports for our method and predictions are discussed in Section 7.8.

7.8 Discussion

Our new approach to predicting functional uORFs in the yeast *S. cerevisiae* makes use of knowledge about biological conservation, gene ontology annotations, and genes' response to different stress conditions. Although the ideas of using some of these types of knowledge are not new in machine learning re-

Table 7.12: The hypotheses generated from total positive examples (part 1).

```
has_functional_role(A) :- regulated_under(A,2),
    mainORF_involved_in_function(A,'GO:0003676').
has_functional_role(A) :- conserved_in_x_species(A,2), aurich_down_up(A),
    mainORF_is_in(A,'GO:0005622').
has_functional_role(A) :- conserved_in_x_species(A,3), regulated_under(A,2),
    belongs_to(A,B), utr(B,C,D), gteq(D,326).
has_functional_role(A) :- mainORF_involved_in_function(A,'GO:0003676'),
    belongs_to(A,B), utr(B,C,D), gteq(D,163).
has_functional_role(A) :- uORF(A,B,C), lteq(C,23), aurich_down_up(A),
    mainORF_involved_in_process(A,'GO:0050789').
has_functional_role(A) :- uORF(A,B,C), lteq(C,6),
    mainORF_involved_in_process(A,'GO:0050789').
has_functional_role(A) :- conserved_in_x_species(A,2), aurich_down_up(A),
    mainORF_is_in(A,'GO:0005622'), mainORF_involved_in_process(A,'GO:0044237').
has_functional_role(A) :- regulated_under(A,2), belongs_to(A,B), utr(B,C,D),
    C=4, gteq(D,472).
```

...English translation in Table 7.13.

Table 7.13: The hypotheses generated from total positive examples (part 2).

English translation: A uORF has functional role if it satisfies at least one of the following rules.

1. if the main gene associated to this uORF is regulated under butanol stress and the product of the main gene is involved in nucleic acid binding;
2. if the uORF is conserved in two other yeast species, the sequence context immediately upstream and downstream of the uORF is AU rich, and the main gene associated to that uORF is localised in intracellular (or protoplasm);
3. if the uORF is conserved in three other yeast species, the main gene associated to this uORF is regulated under butanol, and the length of the UTR to which the uORF belongs is greater than or equal to 326;
4. if the product of the main gene associated to this uORF is involved in nucleic acid binding and the length of the UTR to which the uORF belongs is greater than or equal to 163;
5. if its length is less than or equal to 23 codons, the sequence context immediately upstream and downstream of the uORF is AU rich, and the product of the main gene associated to this uORF is involved in regulation of biological process;
6. if its length is less than or equal to 6 codons and the product of the main gene associated to this uORF is involved in regulation of biological process;
7. if the uORF is conserved in two other yeast species, the sequence context immediately upstream and downstream of the uORF is AU rich, the main gene associated to that uORF is localised in intracellular (or protoplasm), and the product of the main gene associated to this uORF is involved in cellular metabolic process;
8. if the main gene associated to this uORF is regulated under butanol stress, the UTR to which the uORF belongs has 4 uORFs and the UTR length is greater than or equal to 472.

search, such a combination of knowledge has never been explored for learning yeast functional uORFs.

Other machine learning work which make use of expression data and/or gene ontology include Tran et al. (2005) which uses Aleph for predicting protein-protein interaction data, Badea (2003) in which CProgol was used for differentiating two subtypes of adenocarcinoma of the lung[9], Clare and King (2003) for predicting gene function in *S. cerevisiae*, and Trajkovski, Železný, Lavrač and Tolar (2007) where Relational Subgroup Discovery (Lavrač, Železný and Flach, 2000) was used for explaining the differentially expressed genes in some diseases.

The idea of using conservation for learning functional uORFs was inspired from Zhang and Dietrich's (2005a) work. The overall goal of their study was the same as ours, that is to find additional genes potentially post-transcriptionally regulated by uORF(s). However, our way of using conservation test is rather different to theirs. They took a conventional bioinformatics approach, involving a series of multiple alignments of nucleotide sequences. In contrast, we aligned uORF sequences.

Of the 5 genes that Zhang and Dietrich (2005a) hypothesised may have functional uORFs (*RPC11*, *TPK1*, *FOL1*, *WSC3*, and *MKK1*), 3 genes which have one uORF each are predicted by our hypotheses here to have functional uORFs (*RPC11*, *TPK1*, and *FOL1*); see Table 7.14. *WSC3* is not included in our data because experimental results suggested its UTR length exceeded 1000 bases (David et al., 2006). It is also important to note here, 4 of the 5 genes that Zhang and Dietrich hypothesised may have functional uORFs were predicted to be functional by our hypotheses in Chapter 5. Our hypotheses, both in Chapter 5 and here, predict the uORF of *MKK1* to be non-functional.

The uORFs of *ECM7* and *IMD4*, which were found by Zhang and Dietrich (2005a) to have little effect on translation, are not predicted as functional by

[9]Adenocarcinoma of the lung is a type of lung cancer.

Table 7.14: Comparison between predictions made using the model in Table 7.12, in Chapter 5, and by Zhang and Dietrich for the 15 genes reported by Zhang and Dietrich (2005a).

Gene Name	Systematic Name	uORF's Position	uORF's Length	Predicted as functional in		
				Zhang & Dietrich	This study	Chapter 5
RPC11	YDR045C	-60	4	Yes	Yes	Yes
TPK1	YJL164C	-42	5	Yes	Yes	Yes
FOL1	YNL256W	-65	4	Yes	Yes	Yes
WSC3[a]	YOL105C	-50	7	Yes	No data	Yes
MKK1	YOR231W	-71	10	Yes	No	No
ECM7	YLR443W	-15	5	No[c]	No	No data
IMD4	YML056C	-99	14	No[c]	No	No data
SLM2[a]	YNL047C	-110	24	No prediction	No data	No
		-84	6	No prediction	No data	Yes
		-70	4	No prediction	No data	Yes
ARV1[b]	YLR242C	-125	12	No prediction	No data	No
		-108	3	No prediction	No data	Yes
		-40	7	No prediction	No	Yes
HEM3	YDL205C	-129	9	No prediction	No	No
AVT2	YEL064C	-11	4	No prediction	No	Yes
MBR1	YKL093W	-70	7	No prediction	No	Yes
APC2	YLR127C	-27	5	No prediction	Yes	Yes
SPE4	YLR146C	-41	6	No prediction	No	Yes
SPH1	YLR313C	-25	4	No prediction	Yes	Yes

[a] 5′ UTR length was predicted well over 1000 bases.

[b] 5′ UTR length was predicted as 92.

[c] Zhang and Dietrich found these uORFs have little effect on translation; we consider them as non-functional.

our hypotheses in this chapter. These two genes were not included in our data in Chapter 5.

By comparing the predictions for the first 7 genes in Table 7.14, it seems

that our hypotheses in Chapter 5 and in this chapter are making consistent predictions (where data are available). However, the predictions for the other 8 genes in Table 7.14 suggest that the hypotheses in this chapter are more specific than those in Chapter 5. For these 8 genes, which contain a total of 12 uORFs, Zhang and Dietrich (2005a) were only able to verify that these uORFs are real but were not able to get enough evidence to make any hypotheses whether or not these uORFs are functional. This could be an indication that many of these uORFs may be non-functional. It has to be noted that verifying a uORF as a non-functional uORF is even more difficult than verifying a uORF as a functional uORF. Surprisingly, from this set of uORFs, only 2 are predicted to be functional by our hypotheses here, but 9 were predicted to be functional by our hypotheses in Chapter 5.

In August 2007, Cvijovic et al. (2007) published their work on identifying putative regulatory uORFs in yeast. There, conservation was also used as one of the criterias for assessing whether a uORF is potentially functional or not. While we used alignments of uORF sequences, they used alignments of nucleotide sequences. Furthermore, the conservation checking in our work was done automatically, while in theirs, they manually examined the alignments to assess whether a uORF is conserved or not in other yeast species.

Beside conservation, there are also other fundamental differences between our work and Cvijovic et al.'s (2007). The computational system they used was an expert system shell DESS (Kemp, 1997) which uses a certainty factor model for representing uncertainty in both the data and the rules (i.e., the hypotheses) (Shortliffe, 1976). Unlike in our work, where the hypotheses are generated automatically, Cvijovic et al. (2007) manually constructed a rule base, and then followed a cycle of running the expert system with the input data, analysing the results, and manually modifying the rule base for better classification. Thus, when thinking of applying the method for learning functional uORFs in other organisms, applying ours will be more practical than that of Cvijovic et al.

(2007).

From the hypotheses point of view, it is interesting to notice the similarity and the differences between ours and that of Cvijovic et al. (2007). uORF length of 4 to 6 codons was considered as optimal in Cvijovic et al. (2007). In our hypotheses here, one of the rules reflects that (Rule 6 of Table 7.13 on page 123). However, in the other rule (Rule 5 of Table 7.13), "uORF length less than or equal to 23 codons" appeared; this will allow our hypotheses to cover functional uORFs which are longer than 6 codons. Some of this type of uORFs may encode bioactive peptides (Delbecq et al., 1994). Over 200 uORFs of this type have been identified and were found strongly conserved in human and mouse (Crowe et al., 2006). The rule base in Cvijovic et al. (2007) was constructed and modified mainly based on the properties of *GCN4*'s uORFs, such as uORFs are short, and thus will likely fail to identify potential functional uORFs which are quite long.

Cvijovic et al. (2007) considered 50 to 250 nucleotides as the optimal distance between uORF and the main coding sequence. While this feature appeared in the hypotheses presented in Chapter 5 (Table 5.8 on page 76), this does not appeared in our hypotheses here (Table 7.12). This indicates that Aleph considers this feature as less important. This is due to the change from using intergenic sequences to $5'$ UTR sequences. More than 88% of the 4,938 $5'$ UTR sequences used in this chapter have length less than or equal to 250 nucleotides. While the intergenic regions are generally much longer than $5'$ UTRs.

Having discussed the similarity and differences between our work and that of Cvijovic et al. (2007), we compared our predictions and theirs. Cvijovic et al. (2007) predicted 245 additional genes to have 367 new functional uORFs. Among these predictions, 34 uORFs from 32 genes were strongly predicted to be functional. Among the strongly predicted ones, only 24 uORFs from 23 genes lie within the 5'UTRs based on David et al.'s (2006) experiments. When we checked how many of these 24 uORFs from 23 genes are also predicted as

functional by our hypotheses in this chapter, we only found 8 uORFs from 8 genes (i.e., *LDB17*, *CIN8*, *BCK2*, *PMC1*, *FAS1*, *NDE1*, *CKA2*, and *ATH1*). This suggests that these 8 genes are strong candidates for lab experimental studies.

Cvijovic et al. (2007) suggested that an efficient way to experimentally verify the predictions of functional uORFs is by studying the polysomal association of microarray experiments tested under several different conditions. To us, this provides strong support for our method and our predictions.

7.9 Conclusions

We have taken a new approach to learning functional uORFs in the yeast *S. cerevisiae*. The method uses the positive-only setting of Aleph and makes use of knowledge derived from biological sequences of several different yeast species, an analysis of several publicly available expression data sets, and gene ontology annotations; this is the first time such a combination of knowledge has been explored for learning yeast functional uORFs. With only a little adjustment and provided the relevant data are available, our method can be applied to the task of learning functional uORFs in other organisms. The heterogeneous knowledge used here allows Aleph to generate a set of hypotheses with reasonably high sensitivity (81%). While the idea of using conservation for learning functional uORFs is not new, this has led us to introduce a new type of alignment, i.e., alignment of uORF sequences. This alignment allows us to check whether two uORFs from two different species share the same position in the sequence of uORFs relative to the associated coding sequence.

Furthermore, our hypotheses are simple, yet general enough to cover different types of functional uORFs. The hypotheses suggest that features like conservation in at least two other yeast species and involvement of the main gene's product in regulation of biological process, cellular metabolic process and

nucleic acid binding are important in recognising functional uORFs. When the hypotheses were used to predict novel functional uORFs from a set of unlabelled uORFs within the genome of *S. cerevisiae*, they predict 299 further genes to have 450 novel functional uORFs.

Finally, a comparison of our predictions and those of Cvijovic et al. (2007) suggests that a set of 8 predicted functional uORFs from genes *LDB17*, *CIN8*, *BCK2*, *PMC1*, *FAS1*, *NDE1*, *CKA2*, and *ATH1* are strong candidates for lab experimental studies.

Chapter 8

Conclusions and Future Work

As described in Section 1.4, the goal of this PhD project was to develop an automated learning method, using inductive logic programming (ILP) as the learning method and the yeast *Saccharomyces cerevisiae* as the model organism, to help select sets of candidate functional uORFs for experimental studies. Although the predicted functional uORFs have not been verified by lab experiments, the goal of the project has been achieved. Several approaches to learning which uORFs regulate gene expression in the yeast *S. cerevisiae* using a machine learning technique, called ILP, have been presented in this thesis. This is the first time that ILP has been explored for such task. In this chapter, we conclude the thesis by summarising the main results, discussing original contributions and suggesting some future research directions.

8.1 Summary

The characteristics of uORF data (i.e., negatives are too few, positives are not many, unlabelled are abundant) make the uORF domain very challenging. Different ILP systems, different performance measures, different evaluation methods and different types of data for deriving the background knowledge for ILP have been used during the course of this PhD project.

The main results from Chapter 5:

- The positive-only setting of an ILP system, CProgol4.4, can be used to automatically generate rules that, when used as a filter, make the search for novel functional uORFs 17 times more efficient than using random sampling;

- The properties of distance to the coding sequence was found to be particularly useful in the absence of the lengths of 5′ UTRs. ILP used the former as part of its hypotheses to filter out uORFs which are likely to be outside of the 5′ UTR region and was able to filter out almost 90% of the random examples used for prediction, of which majority would be negatives;

- In general, the rules were simple and easy to understand. However, the rules appeared to be too general. This was partly due to the limited background knowledge. All the background knowledge used in Chapter 5 was derived from intergenic sequences of the yeast *S. cerevisiae*.

The main results from Chapter 6:

- The performance of CProgol4.4 in recognising known functional uORFs in the yeast *S. cerevisiae* is significantly increased when mRNA secondary structure is added to the background knowledge (mean RA values: mean without=34.05, mean with=61.53; based on 100 times stratified 10-fold cross-validation). This conclusion still holds when performance was measured using precision, recall, specificity and F_1 score;

- The relationships between RA and other performance measures were investigated using Spearman's correlation. RA was found to have a strong positive correlation with precision, specificity and F_1 score. But, there was no significant correlation observed between mean RA and recall;

- The work described in Chapter 6 is the first machine learning work to study uORFs and mRNA secondary structures together in the context of gene regulation. Although there have been many pieces of work that have studied either uORFs or mRNA secondary structures in the context of gene regulation, we are only aware of two previous studies investigating both together; both of these studies involved wet-lab experiments;

- The rules suggest that mRNA secondary structure influences uORF's ability to regulate translation; a functional uORF may lie inside a stem-loop structure, or intersect with a stem-loop structure on the uORF's left part;

- While it may be obvious from the description of the covering approach, to the best of our knowledge there was no published work investigating the dependency between performances of the hypotheses constructed by CProgol4.4, which uses the covering approach, and the orderings of positive training examples. Experiments in Chapter 6 showed that such a dependency exists. Therefore, to ensure a fair comparison, we recommend that whenever one makes a comparison of performances from different experiments that use the covering approach, one should also check the parameter settings used and the orderings of positive training examples.

The main results from Chapter 7:

- The positive-only setting of another ILP system, Aleph, can also be used to automatically generate rules that can be used to identify functional uORFs. Aleph was used here because it provides a way of inducing hypotheses without dependency on the ordering of the input positive examples;

- For the first time, a combination of knowledge derived from biological sequences of several different yeast species, an analysis of several expression

data sets and gene ontology annotations was explored for learning yeast functional uORFs;

- In conventional bioinformatics, conservation testing is usually done using sequence alignment, where sequence refers the sequence of bases. In Chapter 7, a new type of alignment, i.e., alignment of uORF sequences, is introduced. This new type of alignment allows us to check whether two uORFs from two different species share the same position in the sequence of uORFs relative to the associated protein coding sequence;

- Based on leave-one-out cross-validation, the estimate of how well our hypotheses can correctly identify uORFs which can regulate gene expression (i.e., the sensitivity or recall of our hypotheses) is 81%;

- The resulting hypotheses are simple and informative. They are quite specific yet general enough to cover different types of functional uORFs. The hypotheses provide provisional insights into biological characteristics of functional uORFs. The biological characteristics of functional uORFs may include conservation in at least two other yeast species and involvement of the main gene's product in regulation of biological process, cellular metabolic process and nucleic acid binding;

- The hypotheses predict 299 further genes to have 450 novel functional uORFs. A comparison of our predictions and those of Cvijovic et al. (2007) suggest that 8 predicted functional uORFs from 8 genes (i.e., *LDB17*, *CIN8*, *BCK2*, *PMC1*, *FAS1*, *NDE1*, *CKA2*, and *ATH1*) are strong candidates for lab experimental studies.

8.2 Original Contributions to Knowledge

Original contributions to knowledge from this thesis are as follows.

Contributions to machine learning, particularly ILP:

1. We have introduced a novel application domain for ILP and machine learning in general, namely the uORF domain;

2. We have provided more case-studies on the application of the positive-only setting of ILP. While there have been many applications of positive-only setting of ILP to natural language processing, applications of this setting to real-world biological problems are still rare;

3. We have done the first empirical study to show that there is a dependency between the performances of the hypotheses constructed by CProgol4.4, an ILP system which uses the covering approach, and the orderings of positive training examples. While it may be obvious from the description of covering approach, to the best of our knowledge there was no published work investigating this issue and its implication empirically;

4. We have provided another case-study on the use of RA as a performance measure and an investigation of the relationship between RA and other more well known performance measures (i.e., precision, recall, specificity, and F_1 score).

Contributions to bioinformatics:

1. We have developed an automated learning method for the task of learning which uORFs regulate gene expression in the yeast *S. cerevisiae*. The method can help with the selection of sets of candidate functional uORFs for lab experimental studies;

2. We have predicted 299 further genes to have 450 novel functional uORFs. A comparison with a related study suggests that 8 of these predicted novel functional uORFs (from 8 genes) are strong candidate functional uORFs for lab experimental studies;

3. Through our ILP-rules, we have provided provisional insights into biological characteristics of functional uORFs in the yeast *S. cerevisiae*;

4. We have introduced a new type of alignment (i.e., alignment of uORF sequences rather than alignment of nucleotide sequences) which allows us to check whether uORFs from different species share the same position in the sequence of uORFs relative to the associated protein coding sequence;

5. We have done the first machine learning work to study uORFs and mRNA secondary structures together in the context of gene regulation. Although there have been many pieces of work that have studied either uORFs or mRNA secondary structures in the context of gene regulation, we are only aware of two previous studies investigating both together; both of these involved wet-lab experiments;

6. We have investigated the use of combined knowledge derived from biological sequences of several yeast species, an analysis of several expression data sets and gene ontology annotations for learning yeast functional uORFs. Such a combination of knowledge has never been explored for learning yeast functional uORFs.

8.3 Future Work

In this section, potential new areas of research that have been identified during the course of the PhD project are presented. Some of these potential new areas of research have arisen due to limitations in our work, while others are purely possible extensions or future work.

8.3.1 Future Work: Bioinformatics

Applying Our Method to Other Organisms

The learning task considered in this research was mainly focused on the uORFs of *Saccharomyces cerevisiae*. However, we believe with only a little adjustment and provided the relevant data are available, our method can be applied for learning functional uORFs in other organisms, such as human and mouse.

Using Predicted Conserved mRNA Secondary Structure

In Chapter 6, the background knowledge regarding mRNA secondary structure was derived from predictions made by RNAfold on the given *S. cerevisiae* sequences. However, the reliability of predictions made by RNAfold, and other similar softwares based on thermodynamic energy minimisation, is often questioned because each prediction is made based on a single sequence. Therefore, for future work, one could consider deriving the background knowledge from predicted secondary structures which are conserved among yeast species. This can be done by first aligning sequences of several yeast species and then using the alignment to get the predicted conserved secondary structures.

Viewing mRNA Secondary Structures from a Deeper Level

In Chapter 6, we viewed the predicted mRNA secondary structure from the highest level. This means that we did not consider a nested stem-loop as an independent stem-loop. By doing so, we limited the type of background knowledge that was derived from mRNA secondary structure. Therefore, it would be interesting to investigate the effect of including more detailed background knowledge of the mRNA secondary structure on the ILP's performance in recognising functional uORFs.

Adding mRNA Secondary Structure as Background Knowledge to Chapter 7

It was shown in Chapter 6 that adding mRNA secondary structure to the limited background knowledge used in Chapter 5 significantly increased the ILP's performance in recognising known functional uORFs in the yeast *S. cerevisiae*. Therefore, in the future, it would be worth investigating the effect of incorporating mRNA secondary structure as part of the background knowledge for the work described in Chapter 7.

8.3.2 Future Work: ILP

Investigating When the Orderings of Input Positive Examples Affect the Hypotheses Generated by an ILP System Which Uses the Covering Approach

In Chapter 6, the performances of the hypotheses generated by CProgol4.4, which uses a covering approach, were found to be affected by (dependent on) the orderings of the input positive examples. However, it has to be noted that such dependency was observed under the conditions described in Chapter 6 (i.e., the parameter settings used and that the complete searches of the hypotheses space might not always be performed). Therefore, a further investigation is needed to clarify when exactly and under what conditions the orderings of input positive examples affect the hypotheses generated by an ILP system which uses a covering approach, such as CProgol.

Choosing the Near Optimal Ordering of Positive Examples

Where the output of an ILP system is found to be dependent on the orderings of the input positive examples, finding the best ordering of positive training examples which can give the best performance is desirable. However, this task is often intractable; the number of possible orderings is $n!$, where n is the

number of positive training examples. Thus, one possible direction for future research could be to develop a new algorithm or utilise an existing one, such as a genetic algorithm (Mitchell, 1998), to choose a near optimal ordering.

Considering that CProgol has been used in many applications, a possible further extension to this new research area could be to revisit previous applications of CProgol, where such a dependency can be observed, and see whether the algorithm of choosing the near optimal ordering of positive examples could be applied generically to other domains.

Learning from Weighted Examples

Identifying definite positives or definite negatives can be very difficult and expensive in some domains, such as the uORF domain. However, some partial knowledge (i.e., biological knowledge from domain experts, in the case of uORF domain) could be used to give an insight into how likely individual unlabelled examples are to be positive or negative. Thus, one ILP direction that could be explored is to enable ILP to learn from weighted examples, instead of just definite positives and definite negatives; this work relates to probabilistic ILP (Raedt and Kersting, 2004).

8.3.3 Beyond Computation

Similar to most computational studies in bioinformatics, we would like our predictions to be tested and hopefully confirmed by wet-laboratory experiments. Such experiments might involve comparing the amount of mRNA and the amount of protein produced from the main gene, with and without the candidate functional uORF(s) under investigation. Since these experiments are expensive, we would suggest biologists to start with the uORFs from 8 genes (*LDB17*, *CIN8*, *BCK2*, *PMC1*, *FAS1*, *NDE1*, *CKA2*, and *ATH1*) that were predicted as functional by our hypotheses (Chapter 7) and by Cvijovic et al. (2007).

8.4 Conclusions

This thesis has presented an ILP approach to learning which uORFs regulate gene expression. The method that has been developed can help select sets of candidate functional uORFs for experimental studies. The predictions made have yet to be tested biologically. Positive results are certainly hoped for. However, whatever the results would be, we believe these could be used to improve the computational research as well as to advance the current knowledge in biology.

Through the work in this thesis, we have provided more case-studies on the application of the positive-only setting of ILP. While the idea of positive-only learning has long been introduced in machine learning, the use of this type of learning has been limited. In this thesis, the positive-only learning has, once again, shown its potential. We hope that this could lead to more exploration of this type of learning on a wider range of problems.

The results presented in this thesis suggest that a dependency exists between the performances of the hypotheses generated by CProgol4.4 and the orderings of input positive training examples. Beside being of interest to the ILP community, this could also be of interest to the theorists in general machine learning area. This could lead to further improvement and new developments in machine learning generally.

References

Abastado, J.-P., Miller, P. F., Jackson, B. M. and Hinnebusch, A. G. (1991). Suppresion of Ribosomal Reinitiation at Upstream Open Reading Frames in Amino Acid-Starved Cells Forms the Basis for GCN4 Translational Control, *Molecular and Cellular Biology* **11**(1): 486–496.

Allwein, E. L., Schapire, R. E. and Singer, Y. (2000). Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers, *Journal of Machine Learning Research* **1**: 113–141.

Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., Davis, A. P., Dolinski, K., Dwight, S. S., Eppig, J. T., Harris, M. A., Hill, D. P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J. C., Richardson, J. E., Ringwald, M., Rubin, G. M. and Sherlock, G. (2000). Gene Ontology: tool for the unification of biology, *Nature Genetics* **25**.

Badea, L. (2003). Functional Discrimination of Gene Expression Patterns in Terms of the Gene Ontology, *Pacific Symposium on Biocomputing*, pp. 565–576.

Baim, S. B. and Sherman, F. (1988). mRNA Structures Influencing Translation in the Yeast *Saccharomyces cerevisiae*, *Molecular and Cellular Biology* **8**(4): 1591–1601.

Bockhorst, J., Qiu, Y., Glasner, J., Liu, M., Blattner, F. and Craven, M. (2003). Predicting Bacterial Transcription Units using Sequence and Expression Data, *Bioinformatics* **19**(Suppl. 1): i34–i43.

Bratko, I. and Muggleton, S. H. (1995). Applications of Inductive Logic Programming, *Communications of the ACM* **38**(11): 65–70.

Brown, C. Y., Mize, G. J., Pineda, M., George, D. L. and Morris, D. R. (1999). Role of two upstream open reading frames in the translational control of oncogene *mdm2*, *Oncogene* **18**(41): 5631–5637.

Bryant, C. H. and Muggleton, S. H. (2000). Closed Loop Machine Learning, *Technical Report YCS 330*, University of York, Department of Computer Science, Heslington, York, YO10 5DD, UK.

Bryant, C. H., Muggleton, S. H., Oliver, S. G., Kell, D. B., Reiser, P. and King, R. D. (2001). Combining Inductive Logic Programming, Active Learning and robotics to discover the function of genes, *Electronic Transactions on Artificial Intelligence* **5**(B): 1–36.

Camacho, R. (1994). Learning stage transition rules with Indlog, *in* S. Wrobel (ed.), *Proceedings of the 4th International Workshop on Inductive Logic Programming*, Vol. 237 of *GMD-Studien*, pp. 273–290.

Clare, A. and King, R. D. (2003). Predicting gene function in Saccharomyces cerevisiae, *Bioinformatics* **19 (Suppl.2)**: ii42–ii49.

Clifton, P. F., Hillier, L. W., Fulton, L., Graves, T., Miner, T., Gish, W. R., Waterston, R. H. and Johnston, M. (2001). Surveying Saccharomyces Genomes to Identify Functional Elements by Comparative DNA Sequence Analysis , *Genome Research* **11**(7): 1175–1186.

Clifton, P., Sudarsanam, P., Desikan, A., Fulton, L., Fulton, B., Majors, J., Waterson, R., Cohen, B. A. and Johnston, M. (2003). Finding Functional

Features in *Saccharomyces* Genomes by Phylogenetic footprinting, *Science* **301**: 71–76.

Clocksin, W. F. and Mellish, C. S. (1981). *Programming in Prolog*, Springer, Berlin.

Cootes, A. P., Muggleton, S. H. and Sternberg, M. J. E. (2003). The automatic discovery of structural principles describing protein fold space, *J. Mol. Biol.* **330**(4): 839–850.

Crowe, M. L., Wang, X.-Q. and Rothnagel, J. A. (2006). Evidence for conservation and selection of upstream open reading frames suggests probable encoding of bioactive peptides, *BMC Genomics* **7**(16).

Cussens, J. and Frisch, A. M. (eds) (2000). *Inductive Logic Programming, 10th International Conference, ILP 2000, London, UK, July 24-27, 2000, Proceedings*, Vol. 1866 of *Lecture Notes in Computer Science*, Springer.

Cvijovic, M. (2005). *Comparative Genomic Study of upstream Open Reading Frames*, [Online] Masters Thesis, Chalmers University of Technology, Available from: http://www.math.chalmers.se/Stat/Bioinfo/Master/Theses/2005/2.pdf. [Accessed 3 May 2006].

Cvijovic, M., Dalevi, D., Bilsland, E., Kemp, G. J. L. and Sunnerhagen, P. (2007). Identification of putative regulatory upstream ORFs in the yeast genome using heuristics and evolutionary conservation, *BMC Bioinformatics* **8**: 295.

David, L., Huber, W., Granovskala, M., Toedling, J., Palm, C. J., Bofkin, L., Jones, T., Davis, R. W. and Steinmetz, L. M. (2006). A high-resolution map of transcription in the yeast genome, *PNAS* **103**(14): 5320–5325.

Dehaspe, L. and Toivonen, H. (1999). Discovery of Frequent DATALOG Patterns, *Data Mining and Knowledge Discovery* **3**(1): 7–36.

Delbecq, P., Werner, M., Feller, A., Filipkowski, R. K., Messenguy, F. and Piérard, A. (1994). A Segment of mRNA Encoding the Leader Peptide of the *CPA1* Gene Confers Repression by Arginine on a Heterologous Yeast Gene Transcript, *Molecular and Cellular Biology* **14**(4): 2378–2390.

Delneri, D. (2004). The Use of Yeast Mutant Collections in Genome Profiling and Large-Scale Functional Analysis, *Current Genomics* **5**(1): 1–7.

Dietterich, T. G. and Bakiri, G. (1995). Solving Multiclass Learning Problems via Error-Correcting Output Codes, *Journal of Artificial Intelligence Research (JAIR)* **2**: 263–286.

Duggan, D. J., Bittner, M., Chen, Y., Meltzer, P. and Trent, J. M. (1999). Expression profiling using cDNA microarrays, *Nature Genetics* **21**: 10–14.

Dwight, S. S., Harris, M. A., Dolinski, K., Ball, C. A., Binkley, G., Christie, K. R., Fisk, D. G., Issel-Tarver, L., Schroeder, M., Sherlock, G., Sethuraman, A., Weng, S., Botstein, D. and Cherry, J. M. (2002). *Saccharomyces* Genome Database (SGD) provides secondary gene annotation using the Gene Ontology (GO), *Nucleic Acids Research* **30**(1): 69–72.

Džeroski, S. (2001). Relational Data Mining Applications: An Overview, *in* Džeroski and Lavrač (2001), chapter 14, pp. 339–364.

Džeroski, S., Cussens, J. and Manandhar, S. (2000). An Introduction to Inductive Logic Programming and Learning Language in Logic, *in* J. Cussens and S. Džeroski (eds), *Learning Language in Logic*, Vol. 1925 of *Lecture Notes in Artificial Intelligence*, Springer, pp. 3–35.

Džeroski, S. and Lavrač, N. (eds) (2001). *Relational Data Mining*, Springer-Verlag.

Fiaschi, T., Marzocchini, R., Raugei, G., Veggi, D., Chiarugi, P. and Ramponi, G. (1997). The 5′-untranslated region of the human muscle acylphosphatase mRNA has an inhibitory effect on protein expression, *FEBS Letters* **417**(1): 130–134.

Flach, P. A. and Lavrač, N. (2003). Rule Induction, *Intelligent Data Analysis: An Introduction*, Springer Verlag, pp. 229–267.

Forster, M. J. (2002). Molecular modelling in structural biology, *Micron* **33**(4): 365–384.

Fürnkranz, J. (2001). Round Robin Rule Learning, *in* C. E. Brodley and A. P. Danyluk (eds), *ICML*, Morgan Kaufmann, pp. 146–153.

Gaba, A., Wang, Z., Krishnamoorthy, T., Hinnebusch, A. G. and Sachs, M. S. (2001). Physical evidence for distinct mechanisms of translational control by upstrem open reading frames, *The EMBO Journal* **20**(22): 6453–6463.

Gordon, J. J., Towsey, M. W., Hogan, J. M., Mathews, S. A. and Timms, P. (2006). Improved prediction of bacterial transcription start sites, *Bioinformatics* **22**(2): 142–148.

Grant, C. M. and Hinnebusch, A. G. (1994). Effect of Sequence Context at Stop Codons on Efficiency of Reinitiation in GCN4 Translational Control, *Molecular and Cellular Biology* **14**(1): 606–618.

Grant, C. M., Miller, P. F. and Hinnebusch, A. G. (1995). Sequences 5′ of the first upstream open reading frame in *GCN4* mRNA are required for efficient translation reinitiation, *Nucleic Acids Research* **23**(19): 3980–3988.

Hastie, T. and Tibshirani, R. (1998). Classification by pairwise coupling, *Annals of Statistics* **26**(2): 451–471.

Hernández-Sánchez, C., Mansilla, A., de la Rosa, E. J., Pollerberg, G. E., Martínez-Salaz, E. and de Pablo, F. (2003). Upstream AUGs in embryonic

proinsulin mRNA control its low translation level, *The EMBO Journal* **22**(20): 5582–5592.

Hinnebusch, A. G. (1997). Translational Regulation of Yeast *GCN4*. A Window on Factors that Control Initiator-tRNA Binding to the Ribosome, *J. Biol. Chem.* **272**(35): 21661–21664.

Hofacker, I. L., Fontana, W., Stadler, P. F., Bonhoeffer, L. S., Tacker, M. and Schuster, P. (1994). Fast Folding and Comparison of RNA Secondary Structures (The Vienna RNA Package), *Monatsh. Chem.* **125**(2): 167–188.

Hong, E. L., Balakrishnan, R., Christie, K. R., Costanzo, M. C., Dwight, S. S., Engel, S. R., Fisk, D. G., Hirschman, J. E., Livstone, M. S., Nash, R., Oughtred, R., Park, J., Skrzypek, M., Starr, B., Andrada, R., Binkley, G., Dong, Q., Hitz, B. C., Miyasato, S., Schroeder, M., Weng, S., Wong, E. D., Zhu, K. K., Dolinski, K., Botstein, D. and Cherry, J. M. (2007). Saccharomyces Genome Database, ftp://ftp.yeastgenome.org/yeast/. [Accessed 2-3 May 2007].

Horváth, T., Wrobel, S. and Bohnebeck, U. (2001). Relational Instance-Based Learning with Lists and Terms, *Machine Learning* **43**(1/2): 53–80.

Iacono, M., Mignone, F. and Pesole, G. (2005). uAUG and uORFs in human and rodent 5′ untranslated mRNAs, *Gene* **349**: 97–105.

Jones, D. T., Taylor, W. R. and Thornton, J. M. (1992). A new approach to protein fold recognition, *Nature* **358**: 86–89.

Kakas, A. C., Kowalski, R. and Toni, F. (1998). The Role of Abduction in Logic Programming, *in* D. Gabbay, C. Hogger and J. Robinson (eds), *Handbook of Logic in Artificial Intelligence and Logic Programming 5*, Oxford University Press, pp. 235–324.

Kakas, A. C. and Riguzzi, F. (1997). Learning with Abduction, *in* N. Lavrač and S. Džeroski (eds), *ILP*, Vol. 1297 of *Lecture Notes in Computer Science*, Springer, pp. 181–188.

Kellis, M., Patterson, N., Endrizzi, M., Birren, B. and Lander, E. S. (2003). Sequencing and comparison of yeast species to identify genes and regulatory elements, *Nature* **423**: 241–254.

Kemp, G. J. L. (1997). DESS: Demonstration Expert System Shell. Unpublished User Manual.

Kernighan, B. W. and Ritchie, D. M. (1988). *The C Programming Language*, 2^{nd} edn, Prentice Hall.

King, R. D. (2004). Applying Inductive Logic Programming to Predicting Gene Function, *AI Magazine* **25**(1): 57–68.

King, R. D., Karwath, A., Clare, A. and Dehaspe, L. (2000a). Accurate Prediction of Protein Functional Class From Sequence in the *M. tuberculosis* and *E. coli* Genomes using Data Mining, *Yeast(Comparative and Functional Genomics)* **17**(4): 283–293.

King, R. D., Karwath, A., Clare, A. and Dehaspe, L. (2000b). Genome Scale Prediction of Protein Functional Class From Sequence using Data Mining, *in* R. Ramakrishnan, S. Stolfo, R. Bayardo and I. Parsa (eds), *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 384–389.

King, R. D., Karwath, A., Clare, A. and Dehaspe, L. (2001). The utility of different representations of protein sequence for predicting functional class, *Bioinformatics* **17**(5): 445–454.

King, R. D. and Sternberg, M. J. E. (1990). Machine learning approach for the prediction of protein secondary structure, *J. Mol. Biol.* **216**(2): 441–457.

King, R. D., Whelan, K. E., Jones, F. M., Reiser, P. G. K., Bryant, C. H., Muggleton, S. H., Kell, D. B. and Oliver, S. G. (2004). Functional genomic hypothesis generation and experimentation by a robot scientist, *Nature* **427**: 247–252.

Kowalski, R. (1999). Logic Programming, *in* R. A. Wilson and F. C. Keil (eds), *MIT Encyclopaedia of Cognitive Science*, MIT Press, pp. 484–486.

Kozak, M. (1981). Possible role of flanking nucleotides in recognition of the AUG initiator codon by eukaryotic ribosomes, *Nucleic Acids Research* **9**(20): 5233–5252.

Kozak, M. (1986). Influences of mRNA secondary structure on initiation by eukaryotic ribosomes, *PNAS* **83**(9): 2850–2854.

Kozak, M. (1991). An analysis of vertebrate mrna sequences: Intimations of translational control, *The Journal of Cell Biology* **115**(4): 887–903.

Kozak, M. (1999). Initiation of translation in prokaryotes and eukaryotes, *Gene* **234**: 187–208.

Kozak, M. (2002). Pushing the limits of the scanning mechanism for initiation of translation, *Gene* **299**: 1–34.

Kozak, M. (2005). Regulation of translation via mRNA structure in prokaryotes and eukaryotes, *Gene* **361**: 13–37.

Krummeck, G., Gottenöf, T. and Rödel, G. (1991). AUG codons in the RNA leader sequences of the yeast *PET* genes *CBS1* and *SCO1* have no influence on translation efficiency, *Current Genetics* **20**: 465–469.

Kwon, H.-S., Lee, D.-K., Lee, J.-J., Edenberg, H. J., ho Ahn, Y. and Hur, M.-W. (2001). Posttranscriptional Regulation of Human *ADH5/FDH* and *Myf6* Gene Expression by Upstream AUG Codons, *Archives of Biochemistry and Biophysics* **386**(2): 163–171.

Landers, J. E., Cassel, S. L. and George, D. L. (1997). Translational Enhancement of *mdm2* Oncogene Expression in Human Tumor Cells Containing a Stabilized Wild-Type p53 Protein, *Cancer Research* **57**(16): 3562–3568.

Laso, M. R. V., Zhu, D., Sagliocco, F., Brown, A. J. P., Tuite, M. F. and McCarthy, J. E. G. (1993). Inhibition of Translation Initiation in the Yeast *Saccharomyces Cerevisiae* as a Function of the Stability and Position of Hairpin Structures in the mRNA Leader, *The Journal of Biological Chemistry* **268**(9): 6453–6462.

Lavrač, N., Flach, P. A. and Zupan, B. (1999). Rule Evaluation Measures: A Unifying View, *in* S. Džeroski and P. A. Flach (eds), *ILP*, Vol. 1634 of *Lecture Notes in Computer Science*, Springer, pp. 174–185.

Lavrač, N., Železný, F. and Flach, P. A. (2000). RSD: Relational Subgroup Discovery through First-Order Feature Construction, *in* Cussens and Frisch (2000), pp. 149–165.

Lavrač, N. and Džeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*, Ellis Horwood.

Lesk, A. M. (2002). *Introduction to Bioinformatics*, 1^{st} edn, Oxford University Press.

Lesk, A. M. (2004). *Introduction to Protein Science: Architecture, Function, and Genomics*, 1^{st} edn, Oxford University Press.

Li, H. and Jiang, T. (2005). A Class of Edit Kernels for SVMs to Predict Translation Initiation Sites in Eukaryotic mRNAs, *Journal of Computational Biology* **12**(6): 702–718.

Ling, C. X. and Li, C. (1998). Data Mining for Direct Marketing: Problems and Solutions, *Proceedings of Fourth International Conference on Knowledge*

Discovery and Data Mining, American Association for Artificial Intelligence, pp. 73–79.

Liu, H.-L. and Hsu, J.-P. (2005). Recent developments in structural proteomics for protein structure determination, *Proteomics* **5**: 2056–2068.

Mahony, S., Corcoran, D. L., Feingold, E. and Benos, P. V. (2007). Regulatory conservation of protein coding and microRNA genes in vertebrates: lessons from the opossum genome, *Genome Biology* **8**(5): R85.

Marsden, S., Nardelli, M., Linder, P. and McCarthy, J. E. G. (2006). Unwinding Single RNA Molecules Using Helicases Involved in Eukaryotic Translation Initiation, *J. Mol. Biol.* **361**: 327–335.

Mata, J., Marguerat, S. and Bähler, J. (2005). Post-transcriptional control of gene expression: a genome-wide perspective, *Trends in Biochemical Sciences* **30**(9): 506–514.

Meijer, H. A. and Thomas, A. A. M. (2002). Control of eukaryotic protein synthesis by upstream open reading frames in the 5′-untranslated region of an mRNA, *Biochem J.* **367**: 1–11.

Mignone, F., Gissi, C., Liuni, S. and Pesole, G. (2002). Untranslated regions of mRNAs, *Genome Biology* **3**(3): reviews0004.1–reviews0004.10.

Mitchell, M. (1998). *An Introduction to genetic algorithms*, MIT.

Mitchell, T. M. (1997). *Machine Learning*, McGraw Hill.

Miura, F., Kawaguchi, N., Sese, J., Toyoda, A., Hattori, M. and Morishita, S. (2006). A large-scale full-length cDNA analysis to explore the budding yeast transcriptome, *PNAS* **103**(47): 17846–17851.

Mooney, R. J. (1997). Integrating Abduction and Induction in Machine Learning, *Working Notes of the IJCAI-97 Workshop on Abduction and Induction in AI*, pp. 37–42.

Morris, D. R. and Geballe, A. P. (2000). Upstream Open Reading Frames as Regulators of mRNA Translation, *Molecular and Cellular Biology* **20**(23): 8635–8642.

Muggleton, S. (1991). Inductive Logic Programming, *New Generation Computing* **8**(4): 295–318.

Muggleton, S. (1995). Inverse Entailment and Progol, *New Generation Computing* **13**(3&4): 245–286.

Muggleton, S. (1996). Learning from Positive Data, *in* S. Muggleton (ed.), *Inductive Logic Programming Workshop*, Vol. 1314 of *Lecture Notes in Computer Science*, Springer, pp. 358–376.

Muggleton, S. (1999). Scientific Knowledge Discovery Using Inductive Logic Programming, *Communications of the ACM* **42**(11): 42–46.

Muggleton, S. and Bryant, C. H. (2000). Theory Completion Using Inverse Entailment, *in* Cussens and Frisch (2000), pp. 130–146.

Muggleton, S. and Feng, C. (1990). Efficient induction of logic programs, *Proceedings of the First Conference on Algorithmic Learning Theory*, Ohmsma, Tokyo, Japan, pp. 368–381.

Muggleton, S. and Firth, J. (2001). CProgol4.4: a tutorial introduction, *in* Džeroski and Lavrač (2001), chapter 7, pp. 160–188.

Muggleton, S. H., Bryant, C. H., Srinivasan, A., Whittaker, A., Topp, S. and Rawlings, C. (2001). Are Grammatical Representations Useful for Learning from Biological Sequence Data?-A Case Study, *Journal of Computational Biology* **8**(5): 493–521.

Muggleton, S., King, R. D. and Stenberg, M. J. E. (1992). Protein secondary structure prediction using logic-based machine learning, *Protein Engineering* **5**(7): 647–657.

Muggleton, S. and Raedt, L. D. (1994). Inductive Logic Programming: Theory and Methods, *Journal of Logic Programming* **19/20**: 629–679.

Neafsey, D. E. and Galagan, J. E. (2007). Dual Modes of Natural Selection on Upstream Open Reading Frames, *Molecular Biology and Evolution* **24**(8): 1744–1751.

Oliver, S. G., Winson, M. K., Kell, D. B. and Baganz, F. (1998). Systematic functional analysis of the yeast genome, *Trends in Biotechnology* **16**(9): 373–378.

Ong, S.-E. and Mann, M. (2005). Mass spectrometry-based proteomics turns quantitative, *Nature Chemical Biology* **1**(5): 252–262.

Pesole, G., Grillo, G., Larizza, A. and Liuni, S. (2000). The untranslated regions of eukaryotic mRNAs: Structure, function, evolution and bioinformatic tools for their analysis, *Briefings in Bioinfomatics* **1**(3): 236–249.

Pesole, G., Mignone, F., Gissi, C., Grillo, G., Licciulli, F. and Liuni, S. (2001). Structural and functional features of eukaryotic mRNA untranslated regions, *Gene* **276**: 78–81.

Pevsner, J. (2003). *Bioinformatics and Functional Genomics*, John Wiley & Sons.

Philippakis, A. A., He, F. S. and Bulyk, M. L. (2005). Modulefinder: A tool for computational discovery of cis regulatory modules., *in* R. B. Altman, T. A. Jung, T. E. Klein, A. K. Dunker and L. Hunter (eds), *Pacific Symposium on Biocomputing*, World Scientific.

Pickering, B. M. and Willis, A. E. (2005). The implications of structured 5′ untranslated regions on translation and disease, *Seminars in Cell & Developmental Biology* **16**(1): 39–47.

Preiss, T., Baron-Benhamou, J., Ansorge, W. and Hentze, M. W. (2003). Homodirectional changes in transcriptome composition and mRNA translation induced by rapamycin and heat shock, *Nature Structural Biology* **10**(12): 1039–1047.

Provost, F. J. and Fawcett, T. (1997). Analysis and Visualization of Classifier Performance: Comparison under Imprecise Class and Cost Distributions, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, American Association for Artificial Intelligence, pp. 43–48.

Provost, F. J. and Fawcett, T. (2001). Robust Classification for Imprecise Environments, *Machine Learning* **42**(3): 203–231.

Quiniou, R., Cordier, M.-O., Carrault, G. and Wang, F. (2001). Application of ILP to Cardiac Arrhythmia Characterization for Chronicle Recognition, *in* C. Rouveirol and M. Sebag (eds), *ILP*, Vol. 2157 of *Lecture Notes in Computer Science*, Springer, pp. 220–227.

Raedt, L. D. and Kersting, K. (2004). Probabilistic Inductive Logic Programming, *in* S. Ben-David, J. Case and A. Maruoka (eds), *ALT*, Vol. 3244 of *Lecture Notes in Computer Science*, Springer, pp. 19–36.

Rice, P., Longden, I. and Bleasby, A. (2000). EMBOSS: The European Molecular Biology Open Software Suite, *Trends in Genetics* **16**(6): 276–277.

Riguzzi, F. (1998). Integrating Abduction and Induction, *in* H. Prade (ed.), *Proceedings of the 13th European Conference on Artificial Intelligence*, Wiley and Sons, pp. 475–476.

Ringnér, M. and Krogh, M. (2005). Folding Free Energies of 5′-UTRs Impact Post-Transcriptional Regulation on a Genomic Scale in Yeast, *PLoS Comput Biol* **1**(7): e72.

Roberts, S. (1997). An Introduction to Progol, [Online] Manual, Available from: http://www.doc.ic.ac.uk/∼shm/Software/progol4.2/manual.ps.

Sauer, S., Lange, B. M. H., Gobom, J., Nyarsik, L., Seitz, H. and Lehrach, H. (2005). Miniaturization in functional genomics and proteomics, *Nature Reviews Genetics* **6**: 465–476.

Selpi, Bryant, C. H., Kemp, G. J. L. and Cvijovic, M. (2006). A First Step towards Learning which uORFs Regulate Gene Expression, *Journal of Integrative Bioinformatics* **3**(2): 31.
URL: *http://journal.imbio.de/index.php?paper_id=31*

Shenton, D., Smirnova, J. B., Selley, J. N., Carroll, K., Hubbard, S. J., Pavitt, G. D., Ashe, M. P. and Grant, C. M. (2006). Global Translational Responses to Oxidative Stress Impact upon Multiple Levels of Protein Synthesis, *Journal of Biological Chemistry* **281**(39): 29011–29021.

Shortliffe, E. H. (1976). *Computer-Based Medical Consultations: MYCIN*, Elsevier/North Holland, New York.

Smirnova, J. B., Selley, J. N., Sanchez-Cabo, F., Carroll, K., Eddy, A. A., McCarthy, J. E. G., Hubbard, S. J., Pavitt, G. D., Grant, C. M. and Ashe, M. P. (2005). Global Gene Expression Profiling Reveals Widespread yet Distinctive Translational Responses to Different Eukaryotic Translation Initiation Factor 2B-Targeting Stress Pathways, *Molecular and Cellular Biology* **25**(21): 9340–9349.

Specia, L., Srinivasan, A., Ramakrishnan, G. and das Graças Volpe Nunes, M. (2007). Word Sense Disambiguation Using Inductive Logic Programming, *in* S. Muggleton, R. Otero and A. Tamaddoni-Nezhad (eds), *ILP*, Vol. 4455 of *Lecture Notes in Computer Science*, Springer, pp. 409–423.

Srinivasan, A. (1999). The Aleph Manual, [online], Available from: http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/.

Srinivasan, A. (2001). Four Suggestions and a Rule Concerning the Application of ILP, *in* Džeroski and Lavrač (2001), chapter 15, pp. 365–374.

Srinivasan, A., King, R. D. and Bain, M. (2003). An empirical study of the use of relevance information in inductive logic programming., *Journal of Machine Learning Research* **4**: 369–383.

Stothard, P. (2000). The sequence manipulation suite: JavaScript programs for analyzing and formatting protein and DNA sequences, *BioTechniques* **28**: 1102–1104.

Tamaddoni-Nezhad, A., Chaleil, R., Kakas, A. and Muggleton, S. (2006). Application of Abductive ILP to Learning Metabolic Network Inhibition from Temporal Data, *Machine Learning* **64**(1–3): 209–230.

Thierry-Mieg, N. and Trilling, L. (2000). InterDB, a Prediction-Oriented Protein Interaction Database for *C. elegans*, *in* O. Gascuel and M.-F. Sagot (eds), *JOBIM*, Vol. 2066 of *Lecture Notes in Computer Science*, Springer, pp. 135–146.

Towsey, M. W., Gordon, J. J. and Hogan, J. M. (2006). The prediction of bacterial transcription start sites using SVMs, *International Journal of Neural Systems* **16**(5): 363–370.

Trajkovski, I., Železný, F., Lavrač, N. and Tolar, J. (2007). Learning Relational Descriptions of Differentially Expressed Gene Groups, *Accepted to IEEE Transactions on Systems, Man, and Cybernetics* . Special issue on Intelligent Computation for Bioinformatics.

Tran, T. N., Satou, K. and Ho, T. B. (2005). Using Inductive Logic Programming for Predicting Protein-Protein Interactions from Multiple Genomic Data, *in* A. Jorge, L. Torgo, P. Brazdil, R. Camacho and J. Gama (eds), *PKDD*, Vol. 3721 of *Lecture Notes in Computer Science*, Springer, pp. 321–330.

Turcotte, M., Muggleton, S. and Sternberg, M. J. E. (2001). The Effect of Relational Background Knowledge on Learning of Protein Three-Dimensional Fold Signatures, *Machine Learning* **43**(1/2): 81–95.

Tzanis, G. and Vlahavas, I. (2006). Prediction of Translation Initiation Sites Using Classifier Selection, *in* G. Antoniou, G. Potamias, D. Plexousakis and C. Spyropoulos (eds), *Proc. 4th Hellenic Conference on Artificial Intelligence*.

Velculescu, V. E., Zhang, L., Vogelstein, B. and Kinzler, K. W. (1995). Serial Analysis of Gene Expression, *Science* **270**(5235): 484–487.

Vilela, C., Linz, B., Rodrigues-Pousada, C. and McCarthy, J. E. G. (1998). The yeast transcription factor genes *YAP1* and *YAP2* are subject to differential control at the levels of both translation and mRNA stability, *Nucleic Acids Research* **26**(5): 1150–1159.

Vilela, C. and McCarthy, J. E. G. (2003). Regulation of fungal gene expression via short open reading frames in the mRNA 5′ untranslated region, *Molecular Microbiology* **49**(4): 859–867.

Vilela, C., Ramirez, C. V., Linz, B., Rodrigues-Pousada, C. and McCarthy, J. E. G. (1999). Post-termination ribosome interactions with the 5′ UTR modulate yeast mRNA stability, *The EMBO Journal* **18**(11): 3139–3152.

Wang, L. and Wessler, S. R. (2001). Role of mRNA Secondary Structure in Translational Repression of the Maize Transcriptional Activator *Lc*, *Plant Physiology* **125**(3): 1380–1387.

Watson, J. D., Baker, T. A., Bell, S. P., Gann, A., Levine, M. and Losick, R. (2004). *Molecular Biology of the Gene*, 5^{th} edn, Pearson Education.

Weaver, R. F. (2005). *Molecular Biology*, 3^{rd} edn, McGraw Hill Higher Education.

Whelan, K. E. and King, R. D. (2004). Intelligent software for laboratory automation, *Trends in Biotechnology* **22**(9): 440–445.

Wiestner, A., Schlemper, R. J., van der Maas, A. P. C. and Skoda, R. C. (1998). An activating splice donor mutation in the thrombopoietin gene causes hereditary thrombocythaemia, *Nature Genetics* **18**: 49–52.

Wilkie, G. S., Dickson, K. S. and Gray, N. K. (2003). Regulation of mRNA translation by 5′- and 3′-UTR-binding factors, *Trends in Biochemical Sciences* **28**(4): 182–188.

Willis, A. E. (1999). Translational control of growth factor and proto-oncogene expression, *The International Journal of Biochemistry & Cell Biology* **31**(1): 73–86.

Witten, I. H. and Frank, E. (2000). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann Publishers.

Xu, G., Rabadan-Diehl, C., Nikodemova, M., Wynn, P., Spiess, J. and Aguilera, G. (2001). Inhibition of Corticotropin Releasing Hormone Type-1 Receptor Translation by an Upstream AUG Triplet in the 5′ Untranslated Region, *Molecular Pharmacology* **59**(3): 485–492.

Zeng, F., Yap, R. H. C. and Wong, L. (2002). Using Feature Generation and Feature Selection for Accurate Prediction of Translation Initiation Sites, *Genome Informatics* **13**: 192–200.

Zhang, Z. and Dietrich, F. S. (2005a). Identification and characterization of upstream open reading frames (uORF) in the 5′ untranslated regions (UTR) of genes in *Saccharomyces cerevisiae*, *Current Genetics* **13**(294): 1–11.

Zhang, Z. and Dietrich, F. S. (2005b). Mapping of transcription start sites in *Saccharomyces cerevisiae* using 5′ SAGE, *Nucleic Acids Research* **33**(9): 2838–2851.

Zuker, A. M., Mathews, B. D. H. and Turner, C. D. H. (1999). Algorithms and Thermodynamics for RNA Secondary Structure Prediction: A Practical Guide, *in* J. Barciszewski and B. F. C. Clark (eds), *RNA Biochemistry and Biotechnology*, NATO ASI Series, Kluwer Academic Publishers, chapter 2, pp. 11–43.